The Practice of Not Knowing for Sure: How Agile Teams Manage Uncertainties

Denniz Dönmez and Gudela Grote

Department of Management, Technology, and Economics
ETH Zurich, Switzerland
{ddonmez,ggrote}@ethz.ch

Abstract. Uncertainties are ubiquitous in software development. They impact almost every aspect of a development project. Most uncertainties are viewed as threats to project efficiency and there are strong calls to their reduction. However, uncertainties can pose opportunities for creativity and innovation in some situations. The literature has been dominated by discussions that focus on requirements uncertainties. We aim to extend these discussions by drawing attention to additional types of uncertainties, namely resource, task, and output uncertainties. In this empirical study we investigate the potential of agile software development methods to manage these different types of uncertainties, and examine the mechanisms available to development teams. Our results reveal how some agile teams seized mechanisms to harvest positive and mitigate negative impacts of uncertainties. Drawing upon these results, we discuss several antecedents of successful uncertainty management.

Keywords: Uncertainties, Uncertainty Management, Agile Software Development Methods, Scrum, Empirical Study.

1 Introduction

In this study, we examine the potential of agile software development (ASD) methods to manage diverse uncertainties, which play a major role in software development projects [1,2,3]. Uncertainties have a significant impact on a project's performance as they result in situations that require adequate, oftentimes quick, reactions. They can be manifested in diverse conditions including unexpected events, or a lack of confidence in an estimation, and may consist of anything that is potentially important but not known for sure. Uncertainty is broadly defined as the absence of complete information [4] and linked to the inability of accurate predictions [5]. Because incomplete information can lead to costly delays, redundant work and other inefficiencies, uncertainties are unwanted impediments to most software developers and project managers. The elimination of uncertainties becomes feasible as more information becomes available during the course of a project [6,7] and has long been connected to reduced software project risks and costs [8,24]. However, in some situations, uncertainties can play an important role to foster innovation and productivity [9]. This is mostly the case when high levels of bureaucracy and exuberantly structured project management approaches suffocate creative thinking

H. Baumeister and B. Weber (Eds.): XP 2013, LNBIP 149, pp. 61–75, 2013.

and developers do not enjoy sufficient flexibility to react to dynamic problems and changing environments. Therefore, the adequate handling of uncertainties is crucial for a project's outcome. A team's capability to react adequately to unforeseen future events can result from well-managed uncertainties [10], whereas a highly structured project management approach may result in badly managed uncertainties [11]. Uncertainty management is not equivalent to the elimination of uncertainties. Uncertainties cannot be eliminated entirely in software development projects [10]. Instead, uncertainty management includes two main mechanisms; minimisation of uncertainties and coping with uncertainties [12]. In an attempt to establish the flexibility that is needed for this, many development teams have turned to agile development methods, which stress the importance of situation-dependent problem solving through an 'inspect-and-adapt' approach [10]. ASD methods, such as Scrum, deliberately encourage high flexibility and adaptability through iterative development processes, and foster communication among project stakeholders in order to enable quick and effective adaption to unexpected events. In this context, requirements uncertainties have been extensively studied, e. g. in [4], as the literature has been dominated by discussions that focus on technical aspects. Yet, little attention has been paid to identify the mechanisms that may be necessary to address different types of uncertainties, which exist in software development. We address this gap by studying additional types of uncertainties, which have been largely ignored. We aim to contribute to a more complete understanding of uncertainty management in software development by addressing four different types of uncertainties, namely resource, requirements, task and output uncertainties.

Resource uncertainties refer to incomplete information about the availability of resources that are required for the accomplishment of planned project tasks. Necessary but unavailable resources range from human resources subject to spontaneous temporal unavailability to process artefacts, such as delayed deliverables. **Requirements uncertainties** refer to ambiguous or changing customer demands. Requirements are a major source of uncertainties in software development and have been discussed by several authors [3,13,16], who argue that agile software development becomes especially important under conditions of frequent changes. **Task uncertainties** refer to a lack of clarity regarding the details of desired outcomes and appropriate solutions to problems. Uncertainty is high when tasks have unexpected dependencies or undiscovered problems with envisioned solutions exist. **Output uncertainties** result from incomplete information about the quantity or quality of product features that a team is able to implement in a given time. They are often linked to insufficient task or process knowledge resulting in unplanned delays.

The purpose of this paper is to examine the mechanisms that professional software development teams draw upon to manage these uncertainties. Thereby we shed light on different existing practices that are suggested by ASD methods, as well as established in extension to them, which are seized to approach a topic with significant influence on development projects. Apart from the identification of ASD practices, we present mechanisms that teams utilised to complement ASD methods in order to carry out their work. Our focus includes the potential of ASD to provide both the structure and flexibility necessary for the effective management of uncertainties.

2 Research Method

We applied qualitative methods for data collection and analysis, using mainly interview and observation techniques. In addition, project artefacts, such as documents and drawings, were collected during company visits. We applied observational techniques when we had the possibility to attend team meetings, witness conference calls (e. g. with clients and other project stakeholders) and conduct spontaneous informal interviews, for instance during lunch or coffee breaks. Field notes produced from these conversations were not included in the data analysis, but enriched our understanding of organisational and team processes and the projects' contextual settings.

2.1 Data Collection

Data collection took place in agile software development teams in three companies based in Switzerland. In total, 19 semi-structured interviews with individuals were conducted. Participants were members of 5 different teams and consisted of 15 men and 4 women, which reflects the teams' gender distribution. Table 1 summarises the data sources. The project teams are referred to by letters for confidentiality reasons.

We interviewed at least two and as many as five members of each team, including at least one person with official team leadership responsibility (i. e. Scrum Master, Product Owner, team or project leader). No stakeholders external to the teams were interviewed. This limitation was, however, mitigated through the fact that the teams had very close contact to them, because they worked for the same companies, and were informed about their perceptions concerning the projects. Interviews lasted from 20 minutes to one hour, with an average length of 38 minutes, and were audio-recorded. In addition, we had the chance to engage in informal conversations with almost all team members at multiple occasions as we visited each team several times.

During the interviews we focused on the interviewees' experiences with uncertain situations. We designed semi-structured interview questions that centred on different types of uncertainties. Each interview began with general questions, such as the current status of the project, and then moved to examples of recently experienced situations in which the team had to deal with incomplete information or unexpected events. We also collected experiences with negative, as well as positive outcomes of unexpected events. Interviewees were asked to describe situations they experienced (e. g., 'Could you give me an example of a situation in which your team faced the unexpected fallout of a team member?' or 'Please describe a situation in which you faced a task and it was not clear to you how to accomplish it') and how their team reacted to that particular situation. Special attention was paid to leadership and collaboration mechanisms including how decisions were made, and how communication took place. During the interviews we pursued interesting clues rather than strictly adhere to our interview guideline; we encouraged informants to wander freely in their answers and probed whenever possible.

The teams we approached apply Scrum in large company projects, mostly using sprint lengths of two weeks. According to the team members' expertise, several roles

are established with overlapping responsibilities. Most teams employ specialised developers (e.g., front and back end), testers, designers (software architects), and requirements engineers. All teams are part of the IT development departments of their respective companies, and develop software solutions that are used by other departments of their organisations. The teams differed in several important aspect including not only characteristics, such as team size, but also their choice of leadership style and coordination mechanisms.

Team A is part of a telecom company. Its 19 team members are split into two sub-teams, which are collocated and share one Product Owner and Scrum Master. The team is interdisciplinary and pair programming is used for most development tasks. At the time of data collection, the team had been working together for one year. **Teams B and C** work for a bank. While Team C has 6 collocated developers, Team B consists of 11 developers who are dispersed over three locations (two in Switzerland, one in India). **Teams D and E** are employed by an insurance company and are all collocated. While Team D is the smallest team of our study (it consists of the 4 members who were interviewed), Team E counts 9 team members.

Table 1. An overview of project, team and study participant details

Team code	Project profile	Team size	Number of interviewees incl. roles
A	The project had been set up 2 years before data collection in order to develop an application for customer order management of future products. Scrum has been used from the beginning. Sprint lengths are 2 weeks.	19 team members split into 2 functional, collocated sub-teams.	*5 interviews:* A1-A5 developers; plus several informal interviews with the Scrum Master (A6)
B	The project was started 1.5 years earlier with the aim to develop and maintain several products for internal company use. Scrum has been used from the beginning with 2 week sprints.	12 team members	*4 interviews:* B1 Product Owner B2-B4 developers
C	Releases of a company-internal application are developed in cooperation with internal clients for the last 3 years, using Scrum since 2.5 years with 4 week sprints.	10 team members	*2 interviews:* C1 Scrum Master C2 developer
D	For the previous 2 years, Scrum was used to develop new versions of a customer management system. Sprint lengths were usually 2 weeks but varied sometimes.	8 team members	*4 interviews:* D1 Scrum Master D2 developer D3 Product Owner D4 developer
E	The project serves the development of new company communication technologies and started 1 year prior to data collection using Scrum. Sprints were 2 weeks in length.	5 team members and 1 Scrum coach	*4 interviews:* E1 Product Owner E2 developer E3 developer E4 Scrum coach
5 teams			*19 interviews*

2.2 Data Analysis

Transcripts of interviews constitute the primary data in this study. All interviews were coded to reflect different types of uncertainties, which resulted from earlier ethnographically informed [14] work, and mechanisms seized for their management. We coded the data openly until no new codes emerged, i. e. theoretical saturation [15] was reached. The codes were grouped according to the types of uncertainties they addressed, and summarised into concepts regarding the underlying uncertainty management practices. Those practices that were connected to organisational rather than team or project management characteristics, such as hiring power, were ignored, as not all teams were in the position to apply them.

3 Results

We present practices that are used by teams to manage different types of uncertainties. The results are grouped according to categories of uncertainties that were identified in the interviews. Some uncertainty management practices are related explicitly to agile software development (ASD), whereas others are not. Several practices are mutually dependent or contribute to the management of multiple uncertainties. Results are summarised in Table 1 at the end of the section.

3.1 Resource Uncertainties

Resources consist of technological artefacts or infrastructure, as well as human resources required in the development process. An inadequate level of resources results either from insufficient supply or excess demand for resources. Uncertain availability of resources causes threats to the success of a project.

Unavailability of Artefacts. The unavailability of technical infrastructure, software licenses, or other artefacts can render a team unproductive. Such threats need uncertainty management practices in order to reduce the risk of the inability to complete development tasks. Teams in this study routinely applied risk analysis techniques, however, some reported not to do so systematically. Participants stated they usually thought about risk in the moment when they find themselves in need to respond to an unexpected event and faced a strategic decision, such as to reduce product features or search for substitute functionality. One participant remarked:

"detailed planning would not have helped because the unexpected events were unexpected" (C1, Scrum Master)

Decision analysis was conducted according to *"a common sense process"* (C2, developer) by thinking about opportunities and possible consequences. One common practice was to call a team meeting in order to gather possible solutions for workarounds and make due as good as possible without satisfying their excess demand for the scarce resource.

Quality of Input. Resources available only in unpredictable levels of quality were named as one major source of dissatisfaction in Team A, where deliverables from external departments often were below the team's expected quality. This was especially the case when erroneous items were received that caused unexpected additional work. Team members could not successfully manage input quality uncertainty because causes were rooted in the organisational structure and differing team cultures. The team members complained about the number of bugs in systems they relied on and which were not improved by the supplier. To our knowledge, they did not try to engage their suppliers in close collaboration in an attempt to explain their quality requirements. Moderation was sought from higher level authorities, however, addressing the problem remained a recurring task of the Product Owner.

Availability of Human Resources. One central aspect of uncertainty management in software development is the management of human resources. It ranges from hiring to training and developing, and eventually letting go team members. The duration of an onboarding process (i. e. the phase of integration until a new team member becomes productive) significantly impacts project costs. New team members are required to learn a broad range of tasks reaching from administrative work to specific development tasks. To increase knowledge transfer, documentation of procedures and pair programming sessions were used in most teams.

Knowledge transfer is also crucial when a team member leaves, or is temporarily unavailable. Breaching functional separation of roles was seen as important by all teams, however, in some cases this was not feasible either due to expertise or individual differences of team members. We found that status and roles were created according to seniority and expertise, and had consequences based on team member expectations regarding decision-making and leadership toward conflict solving.

One developer reported that informal leadership structures collapsed after one dominant decision maker had left the team, and the developer was left in the in the unwanted role of his successor because he had become the most senior team member:

"the team dynamics changed completely after [the colleague] had left the team. […] For me now the pressure is much bigger, because I am expected to take over his role now, but I am not this person. There is a lot more pressure for me, because a lot of requests come to me and I must make a lot more decisions now." (A1, developer)

When team members could anticipate their absence from work, or had regular absences because they did not work full time on the project, clusters of sub-teams were formed by the teams so that each team member had a functional substitute. Knowledge sharing and collaboration lied in the responsibilities of the team members and worked best in teams that used pair programming routinely. Team A had the policy that no pair could stay together for more than one task so that knowledge sharing would be maximised.

In one case, a developer was idle because of his inability to support his colleagues. This resulted from expertise differences and an unexpected difficulty which put his task on hold. The problem was solved by an anticipatory planning meeting with the Product Owner that was called to forecast future work packages and start anticipated tasks.

Team E reported that developers were disturbed frequently during their work by requests from other departments (in which they previously worked). The team solved this problem by extending the practice of process visualisation to include the external disturbers and transparently displaying the frequency of their requests:

"especially in the beginning of the development we were disturbed very often. That was really troublesome. […] So we made a wall with who was disturbed more than 15 minutes and counted them, and that put them off so that they didn't come any more." (E1, Product Owner)

3.2 Requirements Uncertainties

Requirements uncertainties have been identified to cause irregularities and costly delays in many projects [16,25]. ASD teams use several mechanisms to manage requirements uncertainties, the effect of which we found to depend largely on communication effectiveness. Complementary to following the suggested collaborative sessions of the Scrum framework, all teams established additional communication structures that reflected their demand for information and integration of stakeholders into the project.

Lack of Details about Demanded Functionality or insufficient understanding of business context posed problems in several cases, which were addressed by customer representatives in the teams, i. e. either Product Owner or a developer with the mandate to communicate with a certain customer. As there is no substitute for business environment knowledge, one team had business representatives integrated as team members who were, however, not working full time with the team. This interdisciplinary not only increased the team's heterogeneity but also starkly increased the availability of rapid requirement clarifications compared to the other teams. Another team had the opposite problem of a customer not stopping to add items to the list of requested features. This was solved through the Product Owner centring discussions on a prototype:

"When you ship something, people start to imagine how they can do business with it. [...] the requirements phase was never ending. Instead, with a prototype, you stop discussions and then you can say: let's focus on this part." (B1, Product Owner)

Ambiguous Information was experienced as a major cause of uncertainty by many participants. Most teams had implemented the common Scrum policy of allowing work items to be commenced only when they were declared 'ready' for development after an evaluation of their ambiguity. When this was not the case, teams prepended investigative work to the requirements until they felt to have sufficiently clear information. Participants referred to such investigations as 'Spikes' (a time period with constrained duration that is used to expand knowledge and reduce requirements or task uncertainty by investigating a specific issue).

Requirement clarification meetings served to increase communication within teams as well as with external stakeholders, and to reduce ambiguities and, hence, uncertainties regarding requirements. One participant stressed the point that, with

team autonomy, clarification lies in the responsibility of the team member. When information was required from unavailable stakeholders, other tasks needed to be turned to while awaiting reply. However, task idleness can become a process risk. One developer stressed the importance of repeatedly requesting required information:

"too many stories came in while too many others were idle. They were blocked and we did not inquire about them again, but this is important. […] it is important to ask [the informants/customers] again, and ask again, and ask again […] to keep bugging until something happens." (A2, developer)

Unexpected Requirement Changes were reported to occur seldom because of the rule that tasks are fixed during an iteration, which most teams adhered to. Requirements, such as product features, were exchanged only in emergency situations during an on-going sprint. With most teams running iterations of two weeks, project managers usually agreed to refrain from altering anything more than the priorities of tasks against the rhythm of the sprints. Instead, tasks were usually introduced through changing product backlog items and their priorities.

3.3 Task Uncertainties

Uncertainty concerning the best way to approach a task was a common theme during the interviews. The most frequently mentioned forms of task uncertainty were missing knowledge about the scope of a task, and lacking clues concerning the optimal solution, which resulted in time-consuming exploratory work.

Quality of a Solution. Finding the optimal solution to a problem requires skills, experience and oftentimes teamwork in order to pool knowledge and discuss possibilities and likely consequences. Expertise was shared with new team members through mentoring systems and pair programming sessions. Some teams by default implemented special task forces assigned to a problem, whereas others implemented frequent consultation meetings. Content specific knowledge was shared in order to qualify more team members to join discussions:

"in our team we do a bit of everything; design, development, testing… I'm just a regular team member and I have to adjust myself to every role" (A2, developer)

The functional separation of roles and responsibilities was less present than the separation according to expertise. Participants stressed that, despite the existence of distinct roles among team members, functional boundaries were often breached according to status resulting mainly from expertise:

"There are roles, sure. But sometimes they are not that strict. A tester who does only testing, a developer who does only development, a designer who does only design, these exist… but when you look at the team as a whole, then they don't – everybody here can work according to his skills" (A1, developer)

Unexpected Difficulties. Developers often got stuck due to lack of experience or task specific knowledge, or because unexpected difficulties arise with a work item.

Participants reported that clear signalling of task completion status helped within a system of transparent process visualisation (usually the task board or an online tool). When cards on a Scrum board were used, they were marked clearly as blocked. This signalled when help was needed or delays expected if demand for team member support remained unmet.

Task Sequence and Process Uncertainty results from incomplete information about a task's dependencies, which are intransparent especially in complex environments where no formal documentation is available. This requires (sometimes informal) meetings to understand the systemic environment. Participants reported that meetings, e.g. during coffee breaks and lunch, contributed significantly to their understanding of the overall process in which they worked. Consequently, some teams institutionalised common coffee breaks, e. g. once per day with the whole team.

Almost all teams tried to counteract a lack of process transparency using process visualisations including Scrum boards and keeping information visible on large sheets attached to a wall. Participants viewed it as the responsibility of the Scrum Master to prioritise tasks, sequentially order them, and make sure the tasks that are picked up belong to one story. Although team members were self-organised in their eyes, many relied on mechanisms that provided them with tasks to be completed. At the same time, developers felt they needed details about the overall process.

3.4 Output Uncertainties

The output a team can produce depends much on its resources and capabilities. Erroneous assumptions, unexpected difficulties or emergencies can force a team to deliver less than expected product features or result in diminished quality, e. g. when thorough testing is omitted.

Time Required to Accomplish a Task and Amount of Accomplishable Work. Output uncertainty was found in most teams in the form of a temporal uncertainty with regard to accomplishable work items in a given time period. Team members were stopped from working on a particular task because of unrelated emergencies that required immediate action, or because of underestimated required work effort. When tasks remained unfinished at the end of a sprint, most teams tried to break off unfinished parts into a new task, which was referred to the subsequent iteration, or they rescheduled the whole task. To mitigate the effects of distractions by emergencies, dependencies between tasks were minimised through assigning independent groups of developers to separated task bundles and formulating tasks as small as possible in size. This way, delays stemming from interdependent tasks could me avoided.

When uncertain about the amount of work that can be accomplished during an iteration, agile software developers usually rely on estimations. Estimation meetings served our participants to pool knowledge from all team members in order to predict the workload of an unknown task as accurately as possible. However, in many cases specific expertise impeded the participation of more than a few team members, while the remaining ones considered themselves disqualified for discussions and therefore

blindly trusted their peers' judgements. All teams performed estimation meetings at the beginning of a sprint, and most re-estimated tasks on a regular basis (e. g., every second day) as new information became available. Conservative estimates served as uncertainty buffers. Participants reported that the accuracy of task estimations largely depended on technological expertise, knowledge about the business environment, and team cohesion, which is related to the time a team had spent together.

The extension of an iteration in the case of unfinished work was a solution applied by one team in order not to *"drag old tasks into a new sprint"* (D1, Scrum Master), but strongly discouraged by others, including a Scrum coach.

Project Status. All teams faced uncertainty about the amount of work remaining. Mitigation was largely drawn from daily status meetings in combination with a system established to signal transparently not only the status of task accomplishment but also the backlog of unattended work items. Daily status meetings were reported to reduce output uncertainties through the frequent possibility to monitor and report current output, as every developer gave a daily account for any completed and newly commenced task. This, however, was usually limited by the temporal horizon of one iteration. Long-term output was monitored and task distribution moderated by specially designated roles, such as the Product Owner, Scrum Master, or business representatives. In some situations, team members were drawn too deeply into daily activities that the overall direction fell into oblivion. One Product Owner reported that when she and the Scrum Master both were absent for a few days, the team

"lost track over its tasks and when [they] came back the team was way out of focus […], they were taking tasks without tracking progress" (B1, Product Owner).

Quality. Variations in the delivered quality were mitigated by the attempt to involve customers in the testing, which focussed on the functionality important to them despite not being qualified software testers. Establishing mutual support among team members contributed to maintaining quality standards. When the team develops a sense of shared responsibility, developers are more likely to support their colleagues:

"I have the responsibility for the whole result and not just for my part" (A1, developer).

Code Errors were reported to be a frequent impediment. In order to avoid them, teams relied on early testing as much as possible, which was constrained by limited access to deployment systems in some cases. Several teams had successfully integrated the functional roles of developers and testers in their team, however, one interviewee warned that it might be detrimental if a developer tests his own code because he will have a narrow sense of the functionality and waste resources.

Unexpected errors during the release of a product were reported to appear less frequently when there is close collaboration with (external) stakeholders and their involvement in release planning activities. Having a release plan that is followed step-wise enables the team to identify the locus of errors quicker. One developer reported delays and decreased functionality of a released product version that could have been avoided if the affected database administrators had participated in a release kick off meeting organised by his team to have everybody on the same page.

Table 2. Uncertainties and applied practices to manage them. Practices that were employed as suggested ASD methods are denoted by (1): Scrum or (2): XP.

Type	Uncertainty or unexpected event	Uncertainty management practice
Resource	Availability of process artefacts	Discussing workarounds with the team in case necessary artefacts are missing, Analysing risks systematically.
	Quality of input	Collaborating closely with suppliers in order to develop understanding for differences [(1,2)].
	Availability of human resources	Working with redundant roles [(1)] and skills, Maintaining a knowledge base, Using transparency enhancing tools, such as publicly displayed charts or workflows [(1,2)].
	Duration for new team members to become productive	Recruiting within the organisation, Keeping documents of company and team procedures updated, Pair programming to support faster knowledge transfer [(2)].
Require-ment	Lack of details about demanded functionality	Integrating stakeholders into the project [(1,2)], Directly communicating with customers [(1,2)], Early prototyping to focus discussions [(1,2)].
	Ambiguous information	Performing investigative tasks until prev. defined clarity criteria are fulfilled [(1)].
	Unexpected changes	Allowing change requests only at certain points in time, i. e. not during sprints [(1)].
Task	Quality of a problem solution	Sharing of content specific knowledge among team to improve discussions, Pooling team members into task forces, Pair programming [(2)].
	Unexpected difficulties	Signalling of blocked tasks [(1,2)], Cultivating mutual team member support.
	Task sequence or process uncertainty	Improving processes via team reflections [(1)], Recognising the importance of informal meetings.
Output	Time required to accomplish a task	Minimising task sizes and dependencies, Matching levels of dependencies to developer availabilities.
	Amount of accomplishable work	Regularly updating task estimations [(1,2)].
	Project status	Separation of responsibilities according to temporal perspectives [(1)]; long-term planners moderate task distribution and prioritisation.
	Quality of the product	Early and frequent testing [(1,2)], Establishing shared team responsibility [(1,2)], Integration of functional roles [(1)], Involving external stakeholders through direct communication, esp. of plans [(1,2)].

4 Discussion

Agile software development (ASD) 'embraces' uncertainties by acknowledging the necessity to react flexibly to unforeseen and unforeseeable events during the course of a project. Our results reveal several mechanisms that are used by agile teams to address different types of uncertainties. The power to manage requirements uncertainties, for which ASD methods are well suited, was especially evident. In addition, ASD teams employ mechanisms to manage resource, task, and output uncertainties. However, in our study their management was complemented in many cases by practices that were not explicitly proposed by ASD methods.

Attempts to control uncertainties are less present in ASD than the emphasis on maintaining flexibility to cope with them. Flexibility is an important prerequisite for the effective management of uncertainties. Without flexibility, a team cannot mitigate the impact of unexpected difficulties or unforeseen dependencies. An inflexible team has insufficient capacity to react adequately to unexpected events, and limited access to a number of practices including the swapping of roles, or dynamically assigning the complementary skills of pair programmers to an emergent task. Positive effects of flexibility surface especially with regard to changing environments. For example, ineffective planning time is reduced when flexible task sequences allow the collection of information required for planning at the point in time at which it is needed.

Team autonomy and the redundancy of critical resources are important contributors to levels of flexibility that enable teams to become better uncertainty managers.

Team coordination and leadership style are closely connected to levels of autonomy and differ substantially between teams. Agile teams rely heavily on structures that support mechanisms for coordination and collaboration. On the basis of structural routines, ASD teams seize a variety of coordination and collaboration mechanisms that help them to collaborate on a wide range of issues [17]. Participants in our study profited from frequent status meetings, the use of physical artefacts and collocation. The benefit of having clearly defined roles and responsibilities, especially for tasks with shared responsibility, surfaced in situations of their absence. Explicitly defined routines provide important guidelines in situations where efficiency is crucial. The positive effects of collocation, and team members' redundant competence have already been discussed in previous research [18,19].

The relationship between flexibility and structure (also referred to as stability or stable structures) has been studied in the literature most prominently in connection with organisational exploration and exploitation [20]. Many authors believe that exploration and exploitation constitute opposite but complementary team characteristics. The notion of exploration refers to flexibility, the creation of knowledge and discovery of new solutions, as opposed to exploiting existing knowledge and solutions by relying on established structures.

It has been argued that a balance needs to be achieved between flexibility and structure in order to optimally address uncertainties [9]. On the one hand, flexibility is necessary to cope with the fast changes and the uncertainties that govern software development. On the other hand, structure needs to be established for efficient work processes and effective knowledge management. Teams who employ both display the ambidexterity that is important for creativity and innovation. These play a vital role in

ASD teams who are constantly pushed to deliver novel solutions. Teams must rely on the potential for creativity that is rooted in adequate collaborative processes [21]. Therefore, the same mechanisms that support the management of uncertainty foster creative thinking and create potential for innovation that requires the application of creative solutions to new problems. However, ASD methods themselves can also provide sources of uncertainties when the flexibility they create remains unmet by the establishment of adequate structures.

One way of seizing organisational ambidexterity is to move flexibly within the boundaries of structured work processes, or to use given structures that allow flexible reactions. Agile software developers routinely apply a number of concepts connected to uncertainty management that organisational researchers found in high-risk teams or action teams. For example, organisational bricolage (defined as 'making do by applying combinations of the resources at hand to new problems and opportunities' by [22]) was found in a study of fire fighters and film crews who routinely had to adapt to unexpected events [23]. One mechanism the teams used was to reorder the sequences of the work process by taking advantage of their knowledge of the work progression and how tasks fit together. Reordering the work involved changing the sequence in which pieces of the overall project were completed. Agile software teams routinely engage in similar practices by prioritising work tasks, evaluating their content and required estimated effort, and re-prioritising them in the event of an unexpected change. The sequential order of tasks is changed in case of unforeseen impediments. Formal roles exist in teams, but are not dominant. Instead, teams rely on a functional hierarchy that is characterised by informal roles according to skills and expertise, while responsibility often remains shared by the entire team. In order to ensure broad knowledge and capabilities across all team members, the breaching of formally assigned roles is common.

5 Limitations

In this study a small number of teams was studied. All teams operated in large company environments where additional contextual constraints apply that may not be generalizable to small firms with different organisational infrastructures. For example, large companies may have less difficulty to temporarily mitigate resource uncertainties in emergency situations because they have more access to resources.

Data collection was performed in a narrow period of time and did not allow us to observe the projects' developments on a larger scale. Our analysis is based mainly on interviews. In order to mitigate limiting factors to our analysis, we visited the teams in their work environments, in almost all cases more than once, and spoke with several team members and project managers to enhance our understanding of the projects.

6 Conclusion

In this paper, we addressed under-discussed aspects regarding different types of uncertainties in software development projects, and identified practices to manage them. We presented findings from an empirical study that involved five agile software development (ASD) teams. A total of 19 interviews were conducted to explore the

teams' challenges associated with uncertainties. We investigated practices to manage four types of uncertainties; resource uncertainty (concerning the availability of human resources and process artefacts), requirements uncertainties (represented by customer demands), task uncertainty (referring to unexpected problems, such as dependencies on delayed input), and output uncertainty (incomplete information concerning the deliverable schedule and scope of the product). The identification of these practices extends our understanding of systematic uncertainty management and possible strategies that are available to teams, which is important in the face of increasing complexity of software projects and, hence, increasing sources of uncertainties. A key output from this study is a set of practices for the effective management of different types of uncertainties, the antecedents of which we discussed based on our insights.

ASD methods provide powerful tools to reduce a number of uncertainties, or cope with them in case they are not eliminable. For example, mechanisms exist to support the estimation of the time to finish a task with satisfactory accuracy, or to react to frequent changes of requirements. Still, agile teams sometimes have to go beyond the possibilities provided by ASD methods in order to adequately react to uncertainties. For uncertainties, such as the time required to make a new team member productive, ASD practices offer little specific advice, and self-organising agile teams have to complement them by missing mechanisms.

The potential to manage uncertainties depends not only on the structures that ASD methods provide, but also on the organisational context and the competence of the team itself. In order to design effective uncertainty management policies one must, therefore, keep in mind mutual dependencies among different types of uncertainties. We recommend that project managers pay attention to systemic dependencies and mutual relationships of uncertainties that affect the performance of a team.

In this study, we focused on the practice of uncertainty management through the lens of four types of uncertainties that were discussed with the participants of this and other studies in order to produce a representative set of uncertainties that ASD teams face. However, the possibility exists that our list is still incomplete and further types of uncertainties are of importance for other teams. We therefore suggest that future research concentrates on producing a complete taxonomy of uncertainties.

Acknowledgements. We thank the participants of this study and their managers for the possibility to explore their projects and spending much of their time explaining and answering questions. We also thank two anonymous reviewers who provided valuable comments and suggestions for this publication.

References

1. Williams, L., Cockburn, A.C.: Agile Software Development: It's about Feedback and Change. IEEE Computer 36 (2003)
2. Nerur, S., Mahapatra, R.K., Mangalaraj, G.: Challenges of Migrating to Agile Methodologies. Communications of the ACM 48, 73–78 (2005)
3. Laplante, P.A., Neill, C.J.: Uncertainty: A Meta-Property of Software. In: 29th Annual IEEE/NASA Software Engineering Workshop, pp. 228–233 (2005)

4. Nidumolu, S.: Standardization, Requirements Uncertainty and Software Project Performance. Information & Management 31, 135–150 (1996)
5. Milliken, F.J.: Three Types of Perceived Uncertainty about the Environment: State, Effect, and Response Uncertainty. The Academy of Management Review 12, 133–143 (1987)
6. McConnell, S.: Software Estimation: Demystifying the Black Art. Microsoft Press (2006)
7. Stutzke, R.D.: Estimating Software-Intensive Systems. Addison-Wesley Professional (2005)
8. Boehm, B.: Software Engineering Economics. IEEE Transactions on Software Engineering 10 (1984)
9. Grote, G., Kolbe, M., Waller, M.J.: On the Confluence of Leadership and Coordination in Balancing Stability and Flexibility in Teams. Paper presented at the 72nd Annual Meeting of the Academy of Management, Boston (2012)
10. Wang, X., Conboy, K.: Understanding Agility in Software Development through a Complex Adaptive Systems Perspective. Presented at the European Conference on Information Systems, December 1 (2009)
11. Boehm, B.W., Turner, R.: Balancing Agility and Discipline. Addison-Wesley (2004)
12. Grote, G.: Uncertainty Management at the Core of System Design. Annual Reviews in Control 28, 267–274 (2004)
13. Maruping, L.M., Venkatesh, V., Agarwal, R.: A Control Theory Perspective on Agile Methodology Use and Changing User Requirements. Information Systems Research 20, 377–399 (2009)
14. Robinson, H., Segal, J., Sharp, H.: Ethnographically-Informed Empirical Studies of Software Practice. Information and Software Technology 49, 540–551 (2007)
15. Glaser, B.G., Strauss, A.L.: The Discovery of Grounded Theory: Strategies for Qualitative Research. Sociology Press, Aldine (1967)
16. Ebert, C., De Man, J.: Requirements Uncertainty: Influencing Factors and Concrete Improvements. Presented at the 27th International Conference on Software Engineering, ICSE (2005)
17. Sharp, H., Robinson, H.: Collaboration and Co-ordination in mature eXtreme Programming Teams. International Journal of Human-Computer Studies 66, 506–518 (2008)
18. Moe, N., Dingsoyr, T., Dyba, T.: Overcoming Barriers to Self-management in Software Teams. IEEE Software (2009)
19. Dorairaj, S., Noble, J., Malik, P.: Understanding Team Dynamics in Distributed Agile Software Development. In: Wohlin, C. (ed.) XP 2012. LNBIP, vol. 111, pp. 47–61. Springer, Heidelberg (2012)
20. Lavie, D., Stettner, U., Tushman, M.L.: Exploration and Exploitation Within and Across Organizations. The Academy of Management Annals 4, 109–155 (2010)
21. Hoegl, M., Parboteeah, K.P.: Creativity in Innovative Projects: How Teamwork Matters. Journal of Engineering and Technology Management 24, 148–166 (2007)
22. Baker, T., Nelson, R.: Creating Something from Nothing: Resource Construction through Entrepreneurial Bricolage. Administrative Science Quarterly 50, 329–366 (2005)
23. Bechky, B.A., Okhuysen, G.A.: Expecting the unexpected? How SWAT Officers and Film Crews handle Surprises. Academy of Management Journal 54, 239–261 (2011)
24. Boehm, B.W., Abts, C., Brown, W., Chulani, S., Clark, B.K., Horowitz, E., Madachy, R., Reifer, D.J., Steece, B.: Software Cost Estimation with COCOMO II. Prentice Hall, Upper Saddle River (2000)
25. Racheva, Z., Daneva, M., Buglione, L.: Supporting the Dynamic Reprioritization of Requirements in Agile Development of Software Products. In: Proceedings of the Second International Workshop on Software Product Management, IWSPM (2008)