Aalto University

School of Science

Degree Programme in Information Networks

Eevert Saukkokoski

What exactly is the use of dailies:

The daily as a practice in the context of lean software development

Master's Thesis

Espoo, 2016

DRAFT! — May 28, 2016 — DRAFT!

Supervisor: Professor Riitta Smeds

Advisor: Otso Hannula M.Sc. (Tech.)

Contents

List of Tables

Chapter I

Introduction

We want our organizations to be adaptive, flexible, self-renewing, resilient, learning, intelligent – attributes found only in living systems. The tension of our times is that we want our organizations to behave as living systems, but we only know how to treat them as machines.

(Wheatley and Kellner-Rogers, 1996, The irresistible future of organizing)

1 Background and motivation

Whence the daily? I ask this question in two senses. Where did it come from and why is it a subject of contestion, or worthy of investigation? In this chapter I put forward an argument that Scrum-inspired agile is the new norm, and thus dailies are a very commonplace occurrence. However, a new school of lean software development with practices such as kanban are beginning to supplant Scrum, calling to question the effectiveness of those methods.

I invite the reader to consider the daily as a practice in the light shed by kanban, taking a practice based research perspective as a unifying measure to the two distinct discourses.

1.1 Rise of agile and lean software development

Agile software development practices have become mainstream (West et al., 2010). Agile might be seen as having been a reaction to the failure of existing models of software development, in terms of their capabilities to both match reality and to facilitate the process of building better software. Software development is not like building a house, so attempting to understand and deconstruct the process in a similar fashion lead to less than stellar results. Such plan-driven approaches had their failings, and agile was the antidote.

Methodologies under the agile umbrella have been characterized with simplicity and ease of adaptation as key (Abrahamsson et al., 2002). Agile is represented as the antithesis of dogma, being fit for critical inspection and malleable to use-case specific needs. But if there's no definite way, knowing if you're doing the right things or going in the right direction is notoriously difficult. Agile's very nature seems to have changed with introduction to the mainstream (West et al., 2010), with Marchenko and Abrahamsson (2008) citing problems like "too many meetings" and disciplined effort required to "keep it simple" as present challenges.

There is no one single way to run a team or company "the agile way" (Kruchten, 2007). There is, however, consensus on some things. Necessity of face-to-face interaction is emphasized in agile literature as critical for transfer of ideas and achieving innovative results. (Highsmith and Cockburn, 2002) In the agile manifesto, this was considered important enough for it to take first place on a list of agile values: "individuals and interactions over processes and tools".

(Beck et al., 2001)

One thing that is undeniably part of the agility toolkit is the practice of a daily: a meeting between development team members taking place every day. “The daily scrum”, as described in the Scrum methodology, could even be construed to have become a symbol of sorts for agility itself based on its influence on the industry (West et al., 2010). There is a by-the-book description of what a daily is in the literature, yet owing to the malleability of agile, how the practice is implemented will likely vary significantly. For instance, time constraints of a daily may be relaxed or the general agenda modified if it is found that they do not aid in reaching desired goals (Marchenko and Abrahamsson, 2008).

Another wave of organisations are taking up a lean software development approach, which espouses some of the same practices but presents a different underlying set of principles. Like building a house, neither is software development like running a factory, yet a brief look into the history of lean will reveal its roots to be in the manufacturing industry. To see where lean fits in the software development scene, it could be though of as a principled way to figure out better ways to do things.

1.2 Capacity for reaction as the differentiator

One could consider an important difference between agile and lean practice to be the degree of responsivity to new input. As per Scrum, an agile team is run in *iterations* or *sprints* with a fixed agenda. This is based on the idea that *timeboxing* allows a team to focus on essential things needing to be done. A lean process, on the other hand, is built upon the concept of *pull-based flow*. Team members will pull work items for themselves when they are ready to do so, and it is the job of the system built around the team to make sure they have work items to pull in. Fundamentally, a timeboxed team will take into account new input before the start of a sprint, while every instance of a work item being *pulled* into the flow is an opportunity to reflect on new information. A visualization of the flow of work used in lean is called a *kanban board*, a concept related to Scrum's aptly named *Scrum board* which is used to detail a sprint's work items.

A question that both agile and lean attempt to answer in their own ways is "how does one operate in a changing environment". Agile takes a somewhat constrained view by recognizing that when delivering software to a customer, their needs can and will be changed. This can be by virtue of a changing business environment, better understood requirements for the system under development or any number of surprises often so characteristic of projects in real life. The reality of the situation is then accommodated for by building software such that it is amenable to changes in plans with the least amount of effort. Lean's perspective is grander in that it admits for *ways of doing* needing to change as understanding grows and situations change.

Perhaps these two pieces can be taken together to form a discourse. Agile originally offered the promise of ways to build better software, but was easily oversold as a silver bullet, leading to failures in adoption and disillusionment. Clearly there was something positive there, but how could *our* organisation make use of it? The path to agile adoption was anything from clear. Enter lean software development: with the endorsement of ideas like "start where you are" and "continuous improvement", one could conceptually figure out the *path* to enlightenment instead of the ostensible *destination.*

1.3 Why would you have a daily?

In section 1.1 we made the assumption of a team having a daily due to it being part of the package ascribed by the Scrum playbook. Indeed, cargo cults where the participants are not aware of why they choose their methods exist in the industry. However, taking a note from the previous section and admitting that ways of doing can be questioned, let's assume that the team is equipped with a sense of reflexivity sufficient to ask the very reasonable question: why would you have a daily?

It is easy to come up with arguments on how the daily might not be an effective use of its participants' time. They are synchronous, meaning participants must be present simultaneously. Though the recommended time box for a daily is 15 minutes, this length limit does not go without saying (as we will observe in section 11.1). Throughout that time, no code is written, no bugs are fixed, no features are implemented, and no customer is served. There are very real opportunity costs associated with having a daily. Making the assumption that the daily is not mandated by external factors (such as perhaps an external consultant insisting on a strict adherence to Scrum), there must be some other value that the team sees in spending that time.

It must be that the team itself has a hypothesis, an understanding of the value sought, or it would not choose to have dailies. The practice would not stick. For a team aware of the costs, making the contrary choice would be deliberately wasteful, so let us further assume that not to be the case. The people engage in something that has no value in terms of software development work done, but is ostensibly worthy of spending everyone's time on *every day*. Because no tangible "work" gets done, it follows that the time left for that work afterwards can hypothetically be spent more effectively. The daily's role must be in somehow enhancing the organisation's effectiveness, meaning capability for value-adding service delivery.

If we overlay the daily with the use of a kanban system for work visualization, we face a problem. Consider a daily with an agenda defined by the Scrum sprint's goals. We would think that when the agenda is fixed and can be known to parties beforehand, the discussion and actions undertaken will be directed towards getting those things done in the fashion the team believes to be best. "I did X yesterday, I'm going to do Y today and will be needing help with Z" would be an apt description of a common schema that each team member could use their chance of communicating at the daily for. This is, indeed, part of recommended best practice. Yet with us now having a way to tell exactly what the mentioned X, Y and Z are by taking a look at the visualization of our work, one might consider the daily to be redundant.

There is, however, another problem. Software development is not done against a static, unchanging background. This is indeed the basis on which modern methodologies build upon and where they derive their usefulness from, as discussed in 1.1. We can say, at the very minimum, that situations change in the world outside somehow, and that to optimize its behavior for yielding better outcomes an organization would do well to take into account these changes. But the world changes a lot; there is at any given point a potentially unbounded queue of input to process, against a very decidedly bounded capacity for handling that input. A perfect system will absorb any and all input to make perfectly informed decisions on what to act upon. Alas, the world is not perfect, and we will not be stopping every day to reconsider everything possible in the light of new input. Let us call this the bounded rationality dilemma.

As discussed in 1.2, a kanban system is on one respect a visualization of work and as such serves to answer the question of *how things get done.* Taking into account the bounded rationality dilemma, the system needs to be accompanied with an answer to the related but subtly different question of *what is a thing needing to get done.* Consider the crude example of a web based service's production servers being literally on fire. This is surely a situation where a team will be reaching for the fire extinguisher without the question of consulting a kanban board. Indeed, situations do arise where taking prompt action is crucial; in other situations it is not at all clear what the salient points in the input are and what there is to be done. We can see that there would be a need for the system to be able to make out literal and figurative fires from entirely benign or even meaningless events.

A daily embedded in a lean workflow would by necessity be different, when there might be any amount of new input for the team to absorb. Fresh customer insight, feature requests, bug reports and detected anomalies in running production systems are a few examples of what a team might want to decide on reacting to. The space of things to discuss and potential actions to take is vastly larger, giving rise to the possibility for a very different and arguably more interesting daily to emerge. The researcher would argue that instead of kanban making the daily redundant, the combination of both is instead worthy of investigation.

1.4 Dailies as practice

The previous section argues that the use of a daily in conjunction with kanban might have interesting features. To rephrase section 1.3, the researcher finds that a hypothesis would be that the development of the software is somehow made better through this: that there's some means, some knowledge that is gained through the engaging in a daily that at least plausibly more than makes up for the time spent. Drawing from section 1.2, if being reactive to new input is important to software development efforts, the daily may be seen as a *sensory organ* for the software development organisation to feel its way forward in a chaotic and ever-changing landscape. In other words, it helps to answer the question of *how do we know to do the right things*.

Disregarding for now the question of *whether* there's evidence for this, what is missing is a method of description for *how* the daily works to do what would be purported here. Both agile and lean discourses have their own ways of discussing and arguing about their respective benefits, of course. To avoid a circular argument or a fallacy of *begging the question*, we should avoid reaching for this description in the very literature that recommends it. We would like to come up with a way to describe the daily as a recurring event where knowledge is produced, made better or different in some way. In doing this we would find ourselves capable of not only showing how a real-life daily *is*, but deriving a description for the *use* of such a daily from first principles. On the corollary, failing to do so would shed some doubt on the fundamental usefulness of the activity.

The question of knowing how to do the right things brings us to the question of knowledge, as well as that of doing. What is knowing, and what is doing the right thing? How is the concept of the right thing to do formed and how is it put to action? In a society increasingly relying on expert knowledge, it comes as no surprise that there exist several fields of research impinging on these subjects. We will be looking at knowing and learning through the theoretical lens of the sociological concept of *practice*.

The *practice-based approach* of organizational research looks at knowledge as acquired in participation, or knowing as something that is inseparable from doing (Nicolini et al., 2003). Practicing can be viewed as a transformative process: not only is there an equivalence between knowing and practicing, but knowledge transforms itself through use (Gherardi, 2011). Practices are social institutions that are sustained by their use within a sociomaterial context. They exist in a continuous state of becoming and responding to an unfolding reality, whereby learning can be understood as a recursive process of interaction between the practice and its practitioners.

The research described in this paper aims to describe the daily as a practice in the context of lean software development. Account of the practice will be given through a case study, elucidating both how a daily happens and what the outcomes are. In other words: against the background of lean principles and through the lens of practice, *what exactly is the use of a daily?*

2 Research problem

2.1 Problem statement

In section 1 we laid out an argument on how the practice of a daily can be taken to be in conflict with the lean principle of waste reduction, if we see the use of tools such as kanban alleviating the original need to have a daily in the first place. We may however consider other uses for a daily which would mesh with lean principles. As outlined in section 1.3, the daily might be an opportunity for learning and thus improving on the work; to continuously adapt to an unfolding reality. In lean terms, this is dubbed continuous improvement. Let us pose the following research problem statement:

> **RP:** How can the practice of a daily support continuous improvement?

2.2 Theoretical research questions

The research problem leads us to a threefold investigation in order to understand the fundamentals in play.

> **TRQ1:** How does a daily in agile software development support the development team?

> **TRQ2:** How does the use of a kanban system support continuous improvement in the context of lean software development?

> **TRQ3:** How can activity theory be used to observe continuous improvement in practice?

Taking the case study in hand we are likely to face a daily quite unlike what is described in the literature. Question *TRQ1* aims at setting a point of comparison which we can take to reflect on the qualities of the daily that we will be observing. With *TRQ2* we probe the role of kanban systems within the context of lean and relate it to the lean principle of continuous improvement. Detaching ourselves from the bounds of the agile and lean discourses we find a language for describing the social process we're observing by discovering an activity theoretical perspective in *TRQ3*.

3 Research approach and scope

- Qualitative study
- Case study
- Ethnographical approach
- Audio analysis methodology
- Theoretical sampling
- Abductive reasoning

The researcher is interested in gaining a deep understanding of the practice of a daily as it occurs in the context of software development. The agile discourse has a history with ethnographically-inspired studies of agile practices (Robinson et al., 2007; Marchenko and Abrahamsson, 2008), and a similar empirical approach was chosen here. An ethnographical approach is specifically well suited for discovery of how practitioners behave within and use the practice under investigation. Given the choice of theoretical background where a sociological understanding of practice plays a significant role, and the desire to observe in context, not to interfere with and to improve, ethnography makes for a sound choice. In the sense that the research also considers parts of speech and specific manipulations of cognitive artefacts as constitutive of knowledge co-creation, the empirical study also carries microethnographical tones (Streeck and Mehus, 2005).

4 Structure of the study

The study is presented in four chapters: introduction, theoretical framework, empirical study, and conclusions. Table 1 illustrates the line of argumentation throughout the study.

I. Introduction	Background and motivation
	Research problem
II. Theoretical framework	Agile software development; dailies
	Lean software development; kanban
	Activity theory; improvement as innovation
	Theoretical synthesis
III. Empirical study	Empirical study description
	Data collection
	Findings on dailies
IV. Conclusions	Research results
	Study implications
	Evaluation

Table 1: Lines of questioning in the study.

In chapter I, the background and motivation of the study is presented. The introduction is completed by a description on the research problem undertaken, the approach and scope chosen and the scope of the study.

Chapter II, theoretical framework, encompasses a review of relevant literature for purposes of understanding the problem domain and allowing for analysis of the empirical data. The background and contents of agile software development is discussed, they are linked to lean software development, and finally an understanding of both is reached for through activity theory. The theoretical synthesis on these perspectives allows us to further research questions for the empirical study.

Chapter III describes a case study, its data gathering and analysis. Findings from the study are presented to allow for answering the empirical research questions.

In chapter IV, the empirical research questions are answered in light of the theoretical understanding gained from chapter II. The results are combined to formulate an answer to the research problem. Finally, the implications of the result are discussed and the study is evaluated.

Chapter II

Theoretical framework

5 Dailies in Agile

5.1 Discovering Scrum

"Traditional" models of software developments, well-researched and mature (Huo et al., 2004), are heavy. They assume that requirements may be discovered beforehand, that they will not undergo significant change through the duration of a project and that fundamentally, the process of developing software is predictable and repeatable (Sutherland, 2001). By these qualities they can be called *plan-driven* (Abrahamsson et al., 2002). The quintessential example of a traditional methodology[1] would be the *waterfall*[2]: gather requirements, devise a solution, program the solution, bring it to customers

[1] *Methodology* is an ascriptive description of techniques: what to do, how, when, by whom and so on.

[2] It should be noted that this simplistic construction of the waterfall is essentially a strawman argument. Winston Royce's 1970 original work detailing the "waterfall" itself remarked how a process without feedback cycles was unlikely to work. (Poppendieck and Poppendieck, 2003, p. 24-25)

(Sutherland, 2001; Huo et al., 2004). Rinse and repeat.

Turns out, this analytical view of software processes has multiple points of failure. To start with: a project's premises may change on the way (Highsmith and Cockburn, 2002), defining requirements is notoriously tricky (Lindstrom and Jeffries, 2004), and in fact it appears that plan-driven models on the whole don't reflect the reality of software development well at all – especially in the fast growth internet and mobile software industries (Abrahamsson et al., 2002).

Enter agility. The *Manifesto for Agile Software Development* (Beck et al., 2001) introduced the umbrella moniker "agile" as a word to describe a new class of models. Reality being complex and diverse, it's no surprise that the manifesto itself covers five different models (Fowler and Highsmith, 2001), and eg. Abrahamsson et al. (2002) identify eight. Agility may well be seen as an ideological movement, or a culture of which many interpretations exist (Glass, 2001; Kruchten, 2007). Abrahamsson et al. (2002) characterise agile models with the following attributes:

Incremental Delivery of software in short cycles

Cooperative Developers and their clients in unhindered, continuous communication

Straightforward Easy to learn, well documented, readily adaptable

Adaptive Options are not locked down, making alterations as needed is enabled

The term agile is thus identified as remarkably polysemous and dependant on context. To get more concrete, we will be discussing a specific agile process methodology called *Scrum*.

5.2 Overview on the Scrum methodology

Scrum is part of the original suite of methodologies giving rise to the agile manifesto (Beck et al., 2001). It has also been recognized among the most popular methodologies (West et al., 2010) and heralded as virtually a de-facto industry standard (Marchenko and Abrahamsson, 2008). Albeit no single practice is endorsed by all things agile (Kruchten, 2007), in terms of industry adoption and influence on generic expectations of what it means to be agile, there is simply no better example than Scrum.

Schwaber (1995) defines the Scrum methodology as a process of software development consisting of phases of pregame, game and postgame[3]. Planning, conceptualization, analysis, and high level architectural design are part of pregame. Postgame consists of the project's closure, when the product under development is deemed ready and will be released. Game is where the product is developed in iterative increments called sprints. It is also the part where Schwaber distinguishes the methodology from the derided waterfall; without the keyword *iterative*, one could find a very obvious mapping from the Scrum words to the world of waterfall.

A sprint is defined by Schwaber (1995) as a collaborative team performance over a preset period of time, where the processes of develop, wrap, review and adjust take place. Development "consists of the micro process of discovery, invention and implementation". Wrapping is about making sure the work is observable, eg. that the resulting software can be ran. Review means a meeting where progress is presented, problems raised and resolved, identified risks observed and responded to. Adjustment refers to taking the freshly

[3]The curious use of the term *game* stems from a sports analogy by Takeuchi and Nonaka (1986). They argued that software development is not best seen as a game of passing the baton, but as a rugby team advancing the ball by passing it back and forth between team members. It is not coincidental that Schwaber's use of the word Scrum to describe his methodology also derives from rugby.

gained information and consolidating it to shared artefacts.

Work items that describe how the product fails to satisfy current requirements are dubbed the backlog (Schwaber, 1995), from which the most important ones are prioritised for completion in a sprint in a sprint planning event (Schwaber and Sutherland, 2011). After a sprint has ran to completion, it is followed by an all-hands sprint review (Schwaber, 1995). This represents an opportunity for reflecting on what was learned in the last sprint and reprioritizing or modifying the work items accordingly (Highsmith and Cockburn, 2002).

Here we've seen an outline of Scrum and how it applies to the whole process of software development. So far our level of abstraction is at the *sprint* stage, or weeks of calendar time. But what about the day-to-day work? In the next section we will see how this is supported in Scrum by the daily Scrum meeting.

5.3 The archetypal Scrum daily

> The heart of the Scrum is the daily stand-up meeting (West et al., 2010)

The original Scrum formulation by Schwaber (1995) does not explicate the need of a daily. Scrum's co-originator (Beck et al., 2001), Jeff Sutherland, describes in his book on Scrum (Sutherland and Sutherland, 2014) that he was inspired in 1993 by a kind of warrior ceremony of the Maori people from New Zealand called the 'haka':

> The haka is a warrior dance that charges up people about to go into battle. While watching it, you can almost see the energy come out of each player and coalesce into a greater whole. With synchronized stomping and clapping and chanting—ritualized movements of cutting an enemy's throat—you can see ordinary men transform

> themselves into something bigger, something greater. They're invoking a warrior spirit that does not accept defeat or dismay.

Sutherland's team analysed the practices of successful teams and found an example in Borland Software Corporation, who gathered their team together every single day to discuss progress and resolve challenges. In trying to find a way to distill Borland's one-hour daily to its essence, Sutherland with his team ended up with the idea of *three questions* and a set of rules.

According to the rules, the daily:

1. Should be held at the same time every day and be attended by everyone.
2. Should not last more than 15 minutes.
3. Should be done standing up to help everyone participate.

During this time, the team members address three questions. The rules may change or be presented with a modicum of slack (see eg. Rising and Janoff, 2000; Yip, 2006), but the three questions tend to be cited essentially intact (Rising and Janoff, 2000; Yip, 2006; Schwaber and Sutherland, 2011):

1. What did I do yesterday?
2. What will I do today?
3. Are there impediments to progress?

As the reader might agree, dailies are very simple to describe. Indeed, that is part of their allure as part of the agile practice toolkit, second in uptake only to iterations (West et al., 2010). We have thus described the essential constituents of a daily scrum, daily stand-up, daily meeting or henceforth just *daily*: discussing progress with everyone every day to focus and empower the team. We have also provided some clarification on how the daily relates to the texture of agile software development activities as a whole.

Due to its popularity, Scrum-inspired agile is reality to many in the industry.

But we can imagine that you're not content with adhering to a pre-bundled set of methods. Is there somewhere else to go from here? In the next section we'll take a dive to the industrial heritage of agile and see if, instead of ascribed, empirically discovered methods, we can find something more fundamental.

6 The lean perspective

In section 5 we described how agile took off after the prominent failure of plan-driven methods to help with software development, and how Scrum and dailies relate to the picture in terms of everyday software development activities. In this section we're going to cover the field of *lean software development*, see where it diverges from agile and what kind of principles we can discover.

6.1 Agile's lean heritage

The seminal work on lean software development is Lean Software Development by Mary and Tom Poppendieck (2003). We'll be using their description of lean to set the stage and see how the discourse is positioned in relation to agile.

In the book's foreword, Jim Highsmith and Ken Schwaber – both prominent signatories to the agile manifesto only some years earlier (Beck et al., 2001) – both individually rejoice at the possibility of applying lean industrial practices to software development (Poppendieck and Poppendieck, 2003, p. xiii-xvi). Highsmith takes note that while agile had an heritage in lean industrial practices, they were simply unapplicable before this work. Schwaber explains that agile processes were constructed "based on experience, trial-and-error, knowledge of what didn't work, and best practices", and that these new tools would allow agile practitioners "to understand why and how the most common agile processes work, or to modify them based on a deep understanding of them, or to construct their own agile process".

What Schwaber is saying is that while he was able to put together a set of empirically discovered and validated practices, he was at a dead end when it came to finding a satisfying way of describing *why* they worked and what the constraints on their effectiveness would be. At the lack of the latter, adherents to Scrum would only have at hand a description of what to do, they'd be left fumbling if they discovered the results not to be to their liking. Indeed, you could be left with the idea that any deviations were harmful and your failure to find process nirvana was only due to failing at being strict enough. On the other hand, lean was pitched as reaching further towards the fundamentals of agile ideas, and allowing the discovery of new practices which would nevertheless serve to achieve the goals set by agile. Where Scrum had been a playbook detailing what was essentially black magic and the potential to devolve into cargo cult[4] software development, lean could serve as a process construction kit.

6.2 Lean and software development

Poppendieck and Poppendieck (2003) observe that software development might have a lot to learn from new product development in general. They leverage principles of what is called *lean development* according to an approach used by automotive manufacturers Toyota and Honda starting from the 1980s (see eg. Ohno, 1988), which they describe thusly:

[4]Cargo Cult Science being famous from Richard Feynman's commencement address at Caltech 1974. He relates a story of an indigenous people who imitate the operation of a runway in hopes of making cargo planes land, yet none do. They're missing something essential. (Feynman, 1998)

> Don't make irreversible decisions in the first place; delay design decisions as long as possible, and when they are made, make them with the best available information to make them correctly.

To these and other related qualities they attribute a significant competetive advantage exhibited by the Japanese manufacturers over U.S. ones; advantage of this kind would ostensibly be desirable to any software company as well. What's good about this approach is that it doesn't come as a bundle of practices to apply, but as principles "understood and proven by managers in many disciplines outside of software development". Principles being universal guiding ideas and insights, which are however not easy to apply to particular environments.

The *seven lean principles* forming the basis of lean thinking are defined by Poppendieck and Poppendieck (2003) as follows.

1. **Eliminate waste.** Anything in the process not contributing to value as perceived by the customer is waste.
2. **Amplify learning.** Software development is a process of learning what works, not an exercise of reducing variation; trying out different approaches and seeing what works must be enabled.
3. **Decide as late as possible.** Deferring decisions is a requirement for effective strategy in complex and evolving environments.
4. **Deliver as fast as possible.** Fast delivery is critical from both customer value and feedback cycle perspectives.
5. **Empower the team.** The people who actually do the work are the ones who understand how to achieve excellence in execution.
6. **Build integrity in.** Software must be sound conceptually, be able to maintain its usefulness over time, and altogether be fit for purpose.
7. **See the whole.** Measuring the specialized contribution of individuals or organizations leads to suboptimization, which is antithetical to integrity.

Of these principles, the first is declared most important and the rest can be seen as following from them. To complete the picture and allow the reader to apply these principles to forming agile practices applicable to their context, Poppendieck and Poppendieck (2003) present 22 different tools. Some of these tools, like the use of iterations, refactoring and testing, will be readily familiar to most agile practitioners (West et al., 2010). We will be taking a further look at something less familiar – what on this list of tools is introduced as *pull systems*, otherwise known by the Japanese name kanban.

6.3 Kanban as a way for continuous improvement

Poppendieck and Poppendieck (2003) motivate pull systems as a way to address principle 4, *deliver as fast as possible.* The argumentation is profoundly simple: people showing up for work need a way to tell what to do. The alternatives are for you to order them to do something, or give them a way to discover that for themselves. In a fast-moving and complex environment, they argue, only the second option for work coordination is viable. This is attributed to the fact that in a chain of connected events, any variation is amplified, making any predefined scheduling invalid in short order. A pull system, therefore, is devised to make work items visible and enable self-direction such that team members may make the most productive use of their time. The name *kanban* for such a practice derives from Japanese for "card" or "placard" and refers to the tokens which stand in for work items.

The conceptualization of kanban presented by Poppendieck and Poppendieck (2003) could be argued to have little more to it than simply allowing developers themselves pick items from a backlog of work items (described in 5.2). A literature review by Ahmad et al. (2013) points at the current conceptualization of kanban in the context of software development having been born in 2004. The following may be considered the kanban principles (Anderson, 2010), found to be congruent with the lean ones (Ahmad et al., 2013):

1. Visualize the workflow.
2. Limit work in progress.
3. Measure and manage flow.
4. Make process policies explicit.
5. Improve collaboratively.

The clearest benefits Ahmad et al. (2013) can find attributed to use of kanban in the literature are better understanding of whole process, improved software quality and improved customer satisfaction. Challenges, on the other hand, include the fact that kanban is not a standalone measure but requires supporting practices (eg. agile ones) and that it necessitates a difficult change in organisational culture. The potential for better understanding of the whole software development process is especially interesting, as Ikonen et al. (2010) suggest that kanban supports the ability to *see waste in the process*.

What happens when we take lean principles and find tools to implement them in practice can be described in more general terms. The drive for a collaborative and continuous elimination of waste may be taken as the defining characteristic of a process of *continuous improvement*. That definition is posed by Bhuiyan and Baghel (2005), who in their literature review place lean manufacturing in a whole class of methodologies sharing the characteristic (six sigma, balanced scorecard, and lean six sigma being the others). It would take little imagination to claim that when lean methods are applied in software development, what they serve to enable is likewise continuous improvement.

Boer and Gertsen (2003) go further to define that when the CI from manufacturing context is overlaid with learning and innovation, it instead becomes *continuous innovation*. We see how, given the view by Poppendieck and Poppendieck (2003) that software development is inherently a learning activity, it could be argued that lean software development is by necessity an exercise in continuous innovation. On the other hand, the distinction can be seen as meaningless in the context of software development. In favor of staying within the lean discourse we will opt for the term continuous improvement to characterise this process.

We have described how lean software development is formed by a collection of principles, the most fundamental of which is waste reduction. The principles taken together aim at the collaborative activity of continuous improvement. Practices derived from these principles may thus be taken to be, in effect, activities in a process of continuous improvement. Due to the nature of software development activities, this process will inherently involve learning and finding new ways of doing. Kanban is a practice the basic use of which is work visualization and self-direction, but which can also be used as a way for seeing waste and through this be an instrument of continuous improvement.

7 Improvement in practice

In what we have described so far, the theme of improvement has been most central. Practitioners of Scrum wished to discover a way of doing things better than plan-driven methods allowed for, and lean principles direct one not only towards one solution but a way of doing things such that change is not only tolerated but enabled and supported. In knowledge work organisations there is often little to force one's hand about how a process is laid out, because it exists largely in peoples' minds and not as a manufacturing pipeline. In this vein we could go deeper and attempt to find out what it is that enables a practice to support continuous improvement. How can we describe such a process of improvement and the means by which it happens? In the following we will draw from the theoretical toolbox of the practice-based approach in an attempt to find the necessary means of description.

7.1 Improvement as a social process of innovation

The improvement of ways of working requires taking novel approaches distinct from how things are done in the present state. The word 'innovation', on the other hand, is often attached to concrete objects, yet it can take other forms; the key feature is simply *novelty*. (Slappendel, 1996) Innovation can be taken as "any idea, practice, material artefact perceived to be new by the relevant unit of adoption". (Zaltman et al 1973; also Rogers and Shoemaker 1971) Crucially, innovation may also be applied to the *process* through which new ideas are generated. (Slappendel, 1996)

Should we take software development to be a collaborative activity where innovations are required to deliver novel solutions, we may also consider the process of delivering or aiding the delivery thereof as something that may be innovated on. Innovation in social interactions has been studied in *innovative knowledge communities*. Hakkarainen et al. (2004) outline three different perspectives on them:

1. **SECI model.** (Nonaka and Takeuchi, 1995) Knowledge is something that can be explicated and objectified. Innovation happens by transforming tacit knowledge to explicit.
2. **Activity theory.** (Engeström et al., 1999) Knowledge is embedded within practices. Overcoming tensions inherent in the activity is a source of innovation.
3. **Conceptual artefacts.** (Bereiter, 2002) Knowledge is expanded by manipulation of shared conceptual artefacts. The ability to extend and create novel artefacts is the source of innovation.

Out of these perspectives we should pick one suited especially for reflecting agile and lean practices. If we consider Engestrom's perspective and knowledge as being embedded in practices, then innovation must be the development of those practices. Indeed, Engestrom is specifically looking at communities where innovation is the development of practices with a shared object of activity. Engestrom (2000) claims activity theory may be used as a tool for analyzing and redesigning work. It is slated to be especially applicable when such learning occurs which has not been given as the objective from the outside, such as by management, but instead stems from contradictory demands experienced within the community. (Engeström, 2001; Engeström and Sannino, 2010)

The most prominent conjunction of activity theory and software development has been in the field of human-computer interaction research, for example Kuutti (1996) and Kaptelinin (1996). These works mainly take interest on what happens in the interface between human and computer instead of on what people do together. Barthelmess and Anderson (2002) and Adler (2005), however, take an activity theoretical view on the process of software development and show that the perspective is applicable for describing some of the industry's most problematic features. In their view, creation of software is distinct from "construction" in that it consists of conception and refinement of designed, intangible artefacts; yet the artefacts are complex and need to be specified in excruciating detail as a collaborative effort. Therefore software development deserves to be described as a knowledge based practice.

The activity theoretical approach appears to be a good fit for describing CI based on what we found out in section 6.3. We will be looking at continuous improvement as taking place within the context of an interactive, social process of innovation as occurring on the level of a group of people having the same object of activity. In the empirical study we will be taking this view and applying a strategy inspired by Nicolini (2009), where we *zoom in* to a practice and inspect it on different levels - from patterns of interactions between individuals to the wider texture of related practices. But first, let's get acquainted with activity theory and how it relates to our focus of investigation, innovation.

7.2 Activity theoretical perspective on innovation

7.2.1 Principles of activity theory

Activity theory can be seen as a general paradigm of studying transformations and social processes, where different activities are distinguished by their objects. (Kerosuo et al., 2010) The object of an activity is the reason the activity exists; without the object of work the activity would cease. (Engestrom, 2000) An activity system may be seen as a sensemaking device where the "raw" object is, as the outcome of a process, transformed into another, collectively understood object. (Engeström, 2001)

Engeström (2001) outlines the principles of activity theory as such:

1. Activity system as the prime unit of analysis
2. Multi-voicedness
3. Historicity
4. Contradictions as sources of change
5. Possibility of expansive transformations

Activity theory concerns the analysis of activity systems which are oriented towards an object of activity, mediated by artefacts and collaborative in nature. Multi-voicedness stems from the fact that in a collaborative effort there are always multiple viewpoints; division of labor serves to create different viewpoints for collaborators with different personal histories. The system itself is historical, too, with rules, conventions and artefacts to attest to that. Stemming from historicity and interactions with an environment, activity systems may be seen as residing inherently in a perpetual state of contradiction, structural tensions between aspects of the system. Finally, it's recognized that these contradictions may be aggravated such that qualitative, expansive transformations in the activity system takes place.

An activity system has a few interesting features in terms of change, learning and innovation. Historicity, the fact that before this state of the activity system there was another state that looked a bit different, implies that the system can get out of date with regard to objects and other systems it interfaces with. It is these expansive transformations which constitute innovation in the terms we lined out in section 7.1. Let us dig deeper into the terms of contradiction and how a process of expansive transformation might take place.

7.2.2 Contradictions as a source of innovation

Innovation tends to occur when there is pressure to change. Within practices, this pressure may be attributed to a "functional failure" of the practice. When the world changes, the way the practice used to work suddenly might not. (Miettinen, 2006) Engestrom (1987) describes these developing contradictions through a state of need that arises first as errors, disturbances or plain discontent with no apparent source. Disturbances are episodes of deviation from standard script (Engestrom 1996, Norros 1996) or, simply, the breaking of a habit, and serve as indication that the system exhibits change potential.

Contradictions alone are not sufficient for change. In activity systems, transformations occur by cycles of expansive learning where contradictions manifest as part of learning actions. (Engestrom, 2000; Engeström and Sannino, 2010) There are four different tiers of contradictions. Primary contradictions are latent, emergent and may appear within any nodes of the activity system. Secondary contradictions manifest openly *between* nodes. When a part of practice changes but faces difficulty with other parts, that is called a tertiary contradiction. Finally, quaternary contradictions occur when the changed practice interacts with its neighboring activity systems. We may map these to the "ideal" epistemic cycle of learning actions presented in Engeström and Sannino (2010) as follows (Engeström, 2001):

1. **Questioning.** Some parts of accepted practice are questioned, criticized, or rejected. (Stems from a *primary contradiction.*)
2. **Analyzing.** Transformation of the situation to find out causes or explanations. (*Secondary contradictions* are presented.)
3. **Modeling.** A simplified, explicit model of the explanatory relationship may be offered.
4. **Examining the model.** The model's applicability is tested.
5. **Implementing the model.** Practical applications, enrichment and conceptual extension bring the model to life. (*Tertiary contradictions* manifest as resistance from other parts of practice.)
6. **Reflection.** The reality of the implemented model's viability is observed.
7. **Consolidation.** The new practice becomes stable and is aligned with neighbouring practices. (*Quaternary contradictions* occur due to realignment.)

From this view, change in a practice amounts to the outcome of a pressure exerted by the existence of a primary contradiction. This pressure is resisted by the other contradictions. There may not be an evident transformation of the situation such that the contradiction would be resolved acceptably. Attempting to implement such a transformation might require adjustment to other parts of practice, so the group needs to undertake effort not initially visioned. Finally, even if this group of practitioners finds something that works for them, the surrounding network of activity systems will need to deal with the effects too.

In the interest of being able to see these different contradictions in the empirical data later on, we might wish to consider what an expression of them would look like. I suggest that the emotions involved with pressure and resistance expressed throughout the expansive learning cycle could be characterised, in exaggeration, like this:

Primary “This thing that we’re doing sucks.”

Secondary “Yes, but there is a reason why it has to suck like that.”

Tertiary “We made it differently and look at what happened. We should have known it would suck.”

Quaternary “Now that we made it work, the others think this sucks for them.”

Expansive learning is thus to be understood as a process of construction and resolution of contradictions that follow from each other. These actions don’t, however, need to follow each other in temporal order, and they don’t all need to be present for learning to occur. Also noteworthy is the fact that it is not *individuals* who change, but instead the object of collective activity and thus the components of the activity system linked to it. (Engeström and Sannino, 2010)

If we are to understand innovation, we must start with seeing the shared object of activity as its basis. (Miettinen, 2006) Furthermore, in order to understand the an expansive learning process in action, we need to gain an understanding of contradictions and their transformations in the activity. (Kerosuo et al., 2010) This will be the undertaking awaiting us in the empirical research chapter.

8 Theoretical synthesis

8.1 Answers to theoretical research questions

> **TRQ1:** How does a daily in agile software development support the development team?

Agile software development refers to a class of software development methodologies, which may be seen as belonging to a particular culture or perhaps even ideological movement. Agile may be characterised by incremental software delivery, cooperativity between developers and clients, straightforwardness and adaptability, and adaptivity to changing circumstances.

The daily belongs among the most popular of agile practices. It was coined by early practitioners of the Scrum methodology, also belonging to the strongest influencers in the industry. The daily is a short, all-hands meeting held every day.

Dailies occur in the context of sprints, development cycles which last for a preset time on the order of weeks. A sprint has a predefined list of work items assigned to it, representing the most important requirements to be satisfied. In the daily, participants discuss the progress of work items and whether help is needed to resolve impediments to progress.

An agile daily supports the development team by providing a daily ritual which empowers the team members to aid each other and helps them focus on the essential in the context of a sprint with defined goals.

> **TRQ2:** How does the use of a kanban system support continuous improvement in the context of lean software development?

Lean software development is an application of lean development principles, originally used in the automotive industry, to the practice of software development. The first principle, which may be taken to be the goal of lean, is the elimination of waste. Waste is everything not contributing to value seen by the customer. Software development, on the other hand, is necessarily an effort of learning or discover better ways to build things. Finally, continuous improvement is the collaborative and unending effort to eliminate waste.

In rapidly changing and complex environments, work coordination is a difficult problem. Also, to leverage input from feedback cycles for learning, delivery should be as fast as possible. For fast value delivery, it is essential that workers be able to interpret by themselves what work there is to be done. A kanban system in software development is used to visualize the workflow and manage the flow of work items through it. Such a system facilitates understanding of the process and being able to see waste in the process as it exists.

The use of a kanban system in lean software development supports continuous improvement in two ways. Firstly, it directly supports faster value delivery, which contributes to continuous improvement by enabling learning. Secondly, it supports seeing waste in the workflow, which potentially allows for addressing the generation of that waste.

> **TRQ3:** How can activity theory be used to observe continuous improvement in practice?

Improvement in general may be seen as a social process of innovation, which can apply to practices. Innovation in social interactions has been studied in innovative knowledge communities. Activity theory is a practice-based approach with which we can look into innovation.

Activity theory describes practices as historical, object-oriented activity systems. Collaboration in the system is artefact-mediated and bounded by rules and division of labor in a community of practice. Activity systems are in a perpetual state of contradiction, or structural tension between parts of practice. Innovation in activity systems is understood as expansive transformation of the object of activity, whereby these tensions are resolved.

Contradictions are the source of innovation. They arise in practice as errors, disturbances or discontent, which indicate that the system exhibits change potential. Primary contradictions are a questioning of a part of accepted practice, exerting pressure on the current state of practice by pointing out a failing. By offering explanations and analysis, secondary contradictions opposing the change pressure can present themselves. If a suitable transformation of the situation is found, tertiary contradictions occur when the new part of practice is fitted with its surrounding system. Finally, a model of action found viable will have to deal with other interfacing practices, and quaternary contradictions can be observed due to the realignment required. Expansive learning is thus a progression of constructing and resolving contradictions, but these do not need to occur in any specific temporal order.

Activity theory may be used to observe continuous improvement in practice by seeing the practice as an activity system, paying attention to developmental contradictions that occur in the practice, and investigating any resulting transformations in the object of activity.

8.2 Empirical research questions

The following research questions are posed to the empirical research data.

> **ERQ1**: How does the observed daily practice compare to the agile description of a daily?
>
> **ERQ2**: How do contradictions and their transformations manifest in a daily?

The first question, *ERQ1*, considers the activity system the daily is embedded in, the "practice-nature" of the daily. We would like to observe the constituents of the activity system and describe them in relation to our theoretical preconceptions from section 5.3. Furthermore, how can we see the role of virtual kanban boards used by the team in the practice?

With the second question *ERQ2* we ask whether we can observe the necessary elements for expansive learning to take place. Can we see contradictions emerging? Between which elements of the activity system, or between systems?

Do contradictions get resolved? When the team engages in a daily, does it evolve?

The undertaking, is, altogether, to describe the daily as a practice within software development, serving to improve on the team's understanding and being linked to the exercise of creating software. If we further show that the daily supports a process of expansive learning, we can conclude that there is use for the daily as a mechanism for the continuous improvement of a software development team – or put more simply, *improving on how the work works.*

Chapter III

Empirical study

9 Empirical study description

9.1 Research context

The research was conducted as a case study of a Helsinki-based software startup in the mobile B2B industry with a headcount of approximately 20 people and a development team consisting of at most 10 at the time of the research. The development team's habit of gathering around a collection of virtual kanban boards every morning was taken as an opportunity to learn about the practice of dailies as exercised within the context of a lean workflow. The researcher took part in the development team's dailies wearing two hats: the ordinary team member hat, which necessitated normal interaction with the team within the context of the daily, and a researcher hat which involved making observations and logging the proceedings.

9.2 Setting and participants

9.2.1 Environment

The chronological context of the development team daily was after a short whole-company standup meeting scheduled to start at 10 in the morning every day. The discussion from the standup meeting would sometimes anecdotally have an impact on the content of the development team daily, but expanding on this relation was not part of the research objectives.

The daily meetings were invariably held in a meeting room equipped with a table, a couch, a TV screen to which laptop computers could be plugged to, and two whiteboards. Attendees would customarily be seated for the duration of the daily. There was not enough room for more than 8 people to sit, so in the case of meetings which more than that amount attended, some would be forced to stand or choose to sit on the floor. On the table, two people would have their laptops open: the *driver* and the *secretary*.

9.2.2 Roles

The driver and the secretary are the only explicit roles defined in the context of the daily.

The driver connects their laptop to the TV screen, enabling the team to view the virtual kanban boards and items therein. Throughout the daily the driver will switch between kanban boards and task items according to how the team's discussion proceeds.

The secretary affects decisions made by the team. This can take the form of manipulating tasks on the boards, making modifications to board structure, and making meeting notes.

9.2.3 Tools used

The most important tool made use of was Asana, a web-based task management application. It views work as *projects*, which consist of *tasks* in a linear list, which can be delineated into segments with *labels*. Tasks consist of a title and a description, may be assigned to individuals, may be given tags, and may be commented on. The daily relied on Asana as a virtual kanban board tool. The projects are understood as *boards*, and the labels are understood as *states* that a task can have.

In addition, meeting minutes were published through Flowdock, a company-internal instant messaging and group chat application. The application structures communication in terms of *flows*, under which participants may converse textually. One of these flows was dedicated for meeting minutes, so that there was usually no other discussion besides what was logged by the secretary.

Tools for sharing knowledge in the physical environment, such as post-its or the whiteboard present in the room, were seldom touched.

10 Data collection and analysis

10.1 Research methodology

From December 2015 to January 2016, a period of time including a break for seasonal holidays, 20 individual daily events were observed and recorded. The time span was chosen both to get a representative sample of dailies and to allow for the potential discovery of a shift in the team's understanding of the system of work and the flow of work through the system.

The events observed were part of the everyday proceedings of the software development team. The team was informed beforehand of the intent to record the proceedings and that the intent would *not* be to interfere with what occurs naturally or deploy interventions for purposes of study. The author took part in all of these events and took the same role of secretary (as described in 9.2.2) in each one. The dailies took place in a meeting room that was fitted with a table, on which a mobile device was set for purposes of recording before the beginning of each daily.

Audio recordings consitute the main body of research data. After dailies, notes on what went on were made to accompany the recordings as basis for further analysis later. In addition to audio recordings and the accompanying notes, the kanban boards' structure was captured as screenshots from the virtual kanban board tool described in section 9.2.3.

Audio recordings were transferred from the recording device, an *iPhone 5S*, as *mp4* files, and played back using *VLC Player*. An initial analysis and structuring of the material was carried out for each daily by first reviewing the associated notes as a clue on possible interesting events, features, or background information useful for understanding the proceedings and then playing back the audio in a linear fashion. Notes of the audio were made in text format, logging timestamps of discussion topics and interactions that were found to be of interest. This *segment analysis* yielded an outline of dailies that is further described in section 10.2.2.

The initial analysis and segment analysis taken together formed a basis for reviewing the material and making it possible to pay more attention to specific portions of the material, some of which was transcribed for presentation. The transcriptions will follow this convention:

Timestamp in minutes:seconds "Speech in between quote marks" (with `attributions` in parentheses)
(descriptions, interpretations and characterisations of situation or action

in parentheses)
"Omissions of fragments [..] and injections [of] missing parts of speech for understandability marked by square brackets"

Timestamp for next set of actions (turns taken in continuous fashion can be transcribed under a single timestamp, but otherwise individual timestamps are presented)

Capturing the kanban boards' structure as screenshots yielded material of two different kinds. The web software Asana described in 9.2.3 presents what the researcher understood as kanban boards as a linear list. The contents of this list were extracted and are presented as indexes in appendix section B2. The individual kanban boards' structure, expressed likewise as a linear list of labels or 'states', was extracted in a similar fashion and presented fully in appendix section B3.

10.2 Overview of collected data

This section presents an outline on the data and its salient features. Table 2 is introduced as a basis for reference. When specific dailies are discussed, their index number (from `01` to `20`) will be used.

Daily index	Date	Recorded	Effective	Language	Attendance
01	8.12.2015	36:36	34:01	English	7
02	9.12.2015	48:53	47:19	English	?
03	10.12.2015	28:30	27:38	English	7
04	11.12.2015	33:51	32:25	English	9
05	14.12.2015	28:16	27:34	Finnish	7
06	15.12.2015	17:47	15:23	Finnish	5
07	16.12.2015	27:08	26:42	Finnish	5

Daily index	Date	Recorded	Effective	Language	Attendance
08	17.12.2015	20:26	19:50	Finnish	?
09	18.12.2015	19:03	17:20	Finnish	?
10	21.12.2015	31:13	30:10	Finnish	4
11	22.12.2015	11:57	11:07	Finnish	3
12	7.1.2016	34:28	32:14	English	8
13	8.1.2016	47:33	45:52	English	7
14	11.1.2016	36:06	35:16	English	10
15	12.1.2016	51:44	51:03	English	9
16	13.1.2016	1:00:21	59:36	English	8
17	14.1.2016	53:35	52:02	English	?
18	15.1.2016	40:29	40:00	English	7
19	18.1.2016	49:57	48:25	English	8
20	19.1.2016	41:11	39:24	English	8

Table 2: Overview of audio material on dailies by daily index number and date. *Recorded* is the duration of the audio recording. *Effective* is the duration of the part of the recording constituting the daily activities (see A1). *Attendance* is the number of participants present.

10.2.1 Attendance and language

The dailies gathered an attendance of three to ten participants. As table 2 shows, there is a drop in attendance halfway through the observation period.

This is attributable to national holidays and the associated vacation periods starting in a staggered fashion. The team's mean daily attendance during normal operation appears to be seven to eight participants (dailies `01` to `05` and `12` to `20`).

Attendees' overall roles in the team could be divided into four categories: *developers* working on the product directly, *quality assurance* responsible for testing the product and verifying that requirements are met, *product owners* of managerial import capable of making backlog priorizations, and *management* whose opinion carries great weight in the way of conducting operations (such as the daily itself). Developers and quality assurance were by convention compelled to take part (at the absence of a good reason to the contrary), whereas product owners and management were not. Roles, e.g. that of product owner and management, may overlap; in this case the more situationally relevant label will be used per transcribed interaction.

For the sakes of both succinctness and preservation of anonymity, when these roles are referred to in transcriptions the following labels will be used. `D` for developers, `QA` for quality assurance, `PO` for product owners, and `M` for management. Different people will be differentiated by suffixing the label with an index number, such that e.g. `D3` will refer to the third developer taking a turn of speech through the specific interaction. Indexes used are specific to the interaction, as opposed to assigned per individual.

Due to its international composition, the team handled most of the dailies in English, reverting to Finnish when team member attendance allowed for it. Seven out of the twenty dailies observed were conducted in Finnish (dailies `06` to `11`). When quotes originally in Finnish are presented in the empirical findings section, translations by the researcher are shown for accessibility.

10.2.2 Durations and segments

The 20 dailies recorded altogether consist of a total of 11 hours, 59 minutes and 12 seconds of audio. According to the segment analysis the recordings could be trimmed to a total effective duration of 11 hours, 33 minutes and 21 seconds constituting the entirety of what was considered as the daily activities. The average effective duration of a daily was 34 minutes and 40 seconds.

Metric	Effective
Total	11:33:21
Average	00:34:40

Table 3: Aggregate metrics of effective daily durations.

A total of 13 different daily segments were identified in the material. Unless otherwise specified, the segment is named by reference to a concrete instance of a board. Prominent examples include `INBOX` and `FIRES`. An overview of the segments is provided in the following.

Most dailies start with an `INTRO`. The `INTRO` is a segment where the team has engaged the daily but is not yet focused on any of the boards available. The segment can be prompted by a call such as *"So, general things"* (`01`, **02:02**), or it might begin more fluidly e.g. with a team member presenting a topic for discussion as in this exchange from daily `03`:

00:33 (unstructured discussion, indistinct background chatter)
"That's an awesome pear." (`D1` making deadpan comment, pointing out nothing useful is happening)
00:37 (the distinct sound of a fruit, supposedly said pear, being chewed)
00:39 "Should we have a new nickname for [team member]?" (`D2` presents a

topic)

Some dailies include a `BOARD OVERVIEW`. In these segments, the focus is not on individual boards, but their overall status or health, relationships and priorities between them and the arrangement of them in the list of boards. Here's an example where a developer inquires after the purpose of colored labels used for different boards (daily `03`):

26:21 "What about.. are the, uh, colors?" (`D` asks question about boards seen on screen)
"Yellow is in QA. Purple has passed QA. Then there's grey that is in smoketest and, well, you won't probably ever see the blue because I.. blue means it's.. has been smoketested and I'll archive it any moment now." (`QA` explains their methods)

Most boards used in dailies were ones of persistent nature. They are not explicitly bound to dates or weeks or deployment cadences. `CYCLES` make an exception to this. They are boards that are numbered and constitute a bundle of shippable things. Boards labelled with the same number would be deployed together, and there are a multitude of them in use at the same time. As alluded to in the quote by *QA* above, the board for a specific cycle will vanish from sight by being archived after work in it has been completed. All of these were grouped together as `CYCLES`. Discussion generally involves *what is being worked on*, *by whom* and *when it will be completed*. An extract from daily `02` demonstrates how conceptions on progress of work and implications of handovers are negotiated in context of `CYCLES`:

12:58 "So, are you still going to make it by friday?" (QA prompts)
"Yes" (`D1` responds)
"Still confident" (`QA` confirms)
"Quite. . . " (`D2` interjects)
13:05 "I have to, I'm leaving on friday" (`D2` explains)
(laughter)

13:15 "So who will be fixing your stuff on monday when it gets to QA or tuesday when it gets to QA?" (`QA` expresses concern)

Cycle is the team's name for a single development cadence. It is used for sets of features that are completed in synchrony with each other and that cannot be shipped incrementally. The opposite of this is a *continuum*, evident in boards such as `NON-CORE MODULE CONTINUUM` and `NON-FUNCTIONAL CONTINUUM`. Work done here can generally be shipped to production immediately after completion.

The use and concrete nature of these segments is described more in depth in section 11.1. Table 4 shows a summary of the data. Complete tables of the daily segments observed can be found in appendix section A1.

Table 4: Total durations of identified segments, their average length and counts of their occurrences in the material.

Segment	Sum	Average	Count
INBOX	04:04:35	00:11:39	21
FIRES	02:18:41	00:06:36	21
INTRO	01:46:12	00:05:54	18
OUTBOX	01:21:28	00:04:04	20
PRIORITY LANE	00:24:17	00:01:26	17
QUESTIONS	00:23:40	00:01:19	18
NON-FUNCTIONAL CONTINUUM	00:23:25	00:01:57	12
BOARD OVERVIEW	00:20:38	00:06:53	3
CYCLES	00:15:20	00:07:40	2
NON-CORE MODULE CONTINUUM	00:13:25	00:01:41	8
MONEYBOX	00:00:49	00:00:25	2
MODULES CONTINUUM	00:00:25	00:00:25	1
MARKETING BOX	00:00:13	00:00:13	1
WATBOX	00:00:13	00:00:13	1

Segment	Sum	Average	Count
Total Result	11:33:21	00:04:47	145

11 Empirical findings

This section presents empirical findings based on which we will answer the empirical research questions from section 8.2. To settle **ERQ1**, in 11.1 we describe the structure of a daily and link this structure to the use and arrangement of kanban boards. **ERQ2** is covered in 11.2 by investigating activity episodes where contradictions occur, and the subsequent transformations of those contradictions.

11.1 How is the daily structured?

We can observe from A1 that no two dailies followed exactly the same structure. However, clear tendencies could be observed. In this section we first describe a prototypical example of a daily and proceed to paint a picture of how the daily came to be structured the way we observed.

An overview of the structure of a daily is afforded to us by the segment analysis described in 10.1. Let us consider daily `03` as a prototypical example, as it has close to average duration, attendees and a clear segment outline. The outline can be seen in table 5.

Segment	**Duration**
INTRO	00:10:45
INBOX	00:01:45
FIRES	00:04:05

PRIORITY LANE	00:03:30
OUTBOX	00:03:20
QUESTIONS	00:02:00
BOARD OVERVIEW	00:02:13
Total duration	00:28:18

Table 5: Outline of daily `03`, presented here as a typical daily. Excerpt from appendix section A1.

11.1.1 Typical segments in depth

Introduction The daily starts with an `INTRO` segment after making sure everyone is accounted for. It's noted that a team member has been absent for a while and that is affecting the team's capability of responding to fires. The team also takes issue with the state of communication between them and another team.

02:30 "Apparently it seems to be some … hard to reach between business and marketing now, so–" (`D1`, referring to events from before the daily)
"You mean sales and marketing" (`M1`, correcting `D1`'s expression of the overall team structure)

02:41 "I'm just saying that we're not alone" (`D1` referring to difficulties shared with the team on a previous occasion)

02:58 "And I felt, it was like .. difficult" (`D1` starting to relate the experience)

03:01 "There's like .. two walls in between [..]" (`D2` continuing the sentiment metaphorically)

03:10 "So `M1` can you take down the walls or?" (`D3` asking for help in resolving the difficulties)
"Yes yes. `M2` comes back, we will shuffle the rooms anyway" (`M1`, referring

to concrete workspace arrangements)

We can observe that the exchange is rich in history and context. Discussions from what appear to be previous dailies are referenced, as well as more immediate occurrences from the same day. The team seems content to describe the situation on a very abstract level, which could point to an understanding of concrete experiences already having been shared. The tension appears to be relieved by an explicit plea for help and a corresponding assurance that very concrete actions will be taken, despite the metaphorical level of discussion so far. Taken together, we see that the team uses the opportunity for coming together to discuss their woes and to find ways for recourse.

The team proceeds through an unstructured go-through of worries until arriving at what is apparently of immediate relevance: an upcoming feature will be requiring a significant amount of quality assurance after being shipped to production. The team assesses ways of approaching this, with `QA` and `M` negotiating the scope of testing required and the dependencies between work items that will have to be resolved to get to this stage.

08:28 "Done today then?" (`M`)
"We can't do that, the thing is not in production [..]" (`QA`)
08:35 "So what is blocking that?" (`M`)
"Uhh .. the [feature] hasn't passed QA" (`QA`)
"Okay. And what is blocking that?" (`M`)
08:44 "`D` has fires." (`QA`)
"No, they're working right now, there's no [feature] related tasks" (`D`, challenging `QA`)
"Yes, there is." (`QA` restating their position)

This exchange appears to relate to the status of a specific cycle, but the team is not using the cycle's board to mediate the discussion. Thus the exchange is part of the `INTRO` segment, not `CYCLES`.

The team continues to negotiate the interrelations between work items handled by different team members and handovers required. Concern is expressed about a team member leaving for vacation after the ongoing week and some items perhaps not being completed by then, but reassurances to the contrary appear to placate everyone.

An attempt is made to start the `INBOX` segment with the prompt:

10:05 “What about starting from Asana now?” (`M`)

This is ineffective, because another team member wants to discuss handovers not yet mentioned. The secretary can be heard making meeting minutes on the subject.

Inbox The `INBOX` segment finally starts at **11:25**. The *driver* opens Asana and allows the team to view items in the inbox on a TV screen. The inbox is a board that contains new items as input to the team and to be handled in the daily proceedings. Starting from the topmost item, item labels are read out loud. Most often, a variation of the following question is used to settle the imminence of items encountered:

11:51 “Fire or not?” (`M`)

Either the item is a fire, requiring the team’s immediate attention, or it can be postponed to a later time. Fire would mean that the *secretary* moves the item for it to get handled in a later `FIRES` segment. The team members will respond with either “fire” or another board where the item would get handled in the appropriate fashion.

12:23 “Not a fire?” (`M`)
“No ..” (`D`)
“Wishlist.” (`M`)

The wishlist is a labeled section of the inbox board that is intended for a later grooming with relevant product owners involved. The secretary thus moves the item away from the inbox's topmost segment to the wishlist, and continuing in this way the inbox is gradually consumed of unhandled items.

Fires Once the inbox is empty, the driver without prompt moves the team's view to a board titled *"[FIRE]: This must be emptied"*, signaling the start of the `FIRES` segment. Again the team will proceed from top to bottom, this time discussing items that have moved between states since last observation.

Based on the board's labeling of states, (detailed in appendix B3), the items further to the top are closer to having been resolved whereas at the bottom no work is being done on them. The more than a dozen different states communicate an intricate understanding of a work item's progression from backlog to development, to testing, production and finally a state of doneness that can be agreed on by the team.

Distinguishing this segment from the previous one, items will usually not be moved by the secretary. They may, however, be marked as complete after the team agrees there's nothing more to be done.

Progress with fire extinguishing, or lack thereof, can be touched briefly:

14:32 "So `D1`, all ok, need anything?" (`QA`, referring to work item presented by driver)
"Uh ... yeah, no, just to note that-that me and `D2`, we had a, had a prepared battle plan for this specific occasion, and our first attempt at-at fixing it instantly failed. So .. I'm going to have to *actually* fix this." (`D1`)

14:52 "Can you actually fix this?" (`M` seeks confirmation)
"Yes, I can actually fix this." (`D1` concludes)

In this case the details of how the actually-fixing would be done are omitted in the discussion as something that belongs outside the scope of the daily. What gets communicated is the belief that work items will be moving today despite lack of progress thus far.

Sometimes a balance between expedited technical implementation and other rationale, such as product design, needs to be reconsidered. Here, a developer presents a caveat in their intended approach to resolve a fire:

15:08 "Most likely the fix is kind of .. people don't like it, but like that's the .. easiest way to fix" (`D`)
"What?" (`QA`, incredulously)
"You need to save the changes before you can open the [feature]" (`D`)
15:22 "Why doesn't it save it automatically? [..]?" (`M`)
"It did, but–" (`QA`)
"No, it didn't" (`D`)
"It looked like-!" (`QA`)
"Yeeah it looked like, yeah, sure. (short laughter)" (`D`)
15:33 "But why not that way?" (`M`)
"Well, we could .. do that that way" (`D`)
15:37 "Wouldn't that make more sense?" (`M`)
"Yeah." (`D`)
"Just saves it. Next." (`M`)

Negotiation on how thoroughly the board will be combed also happens. Here, a developer poses the question of whether *backlog items* (meaning items that have not yet moved since their addition to fires) should be handled at this time:

16:42 "Do we want to discuss the backlog items?" (`D1`)
"They are the same, I guess … Not moved" (`D2`)
"But no new fires. Or are these two by `D3`, are these new?" (`D1`)
"No no, they were [new] yesterday" (`QA`)

It's shortly determined that there would be nothing to gain, and the team moves on to a new board.

Priority lane The driver opens up *"Priority lane"*, commencing the `PRIORITY LANE` segment. The flow of discussion is similar, with moved work items getting discussed from top to bottom. However, the team has chosen to label the states differently. There is significantly less structure as compared to fires: **DONE**, **DOING**, **TODO** and **LONGSTANDING**. This can be attributed to the non-technical or perhaps ad-hoc nature of work items placed here.

A developer immediately starts relating the outcome of a discussion that was modelled as a work item on the priority lane and can now be seen by the team as "done":

17:15 "This was huge: [..]" (`D`, with explanation of defects discovered in production system)

Due to the critical nature of the findings, the team would expect there to have been a follow-up work item. A manager expresses concern on whether action will be taken. This gives rise to a discussion on how the flow of work is being modelled for the team's benefit in this specific occasion:

18:00 "But isn't that kind of a fire?" (`M`)
18:08 "I'm doing everything I can, I have the one other fire first." (`D`)
"Yeah but where is this kind of task?" (`M`)
"It's in the module continuum, it's being worked on .. like, right now .. there's .. this many tasks about it" (`D`)
"Okay ... but wouldn't it make sense to promote them as a fires also? And not leave them into the module continuum?" (`M`)
"Hhh if you want to-" (`D`)
"For more visibility ... because you can have task in two projects" (`M`)

A rationale of visibility is presented, along with a bargain in that the developer's existing modelling can remain essentially undisturbed. It also becomes evident how the team has a conception of fires as *urgent-but-unplanned* versus *planned-and-prioritized*, which results in a curious tension:

18:48 "Yeh ... which one do you want [to see in fires]?" (`D`)
"Everything that's considered a fire [and is] in module continuum ... because that's essentially blocking the module continuum for continuing ... on the actual *planned* features" (`M`)

19:04 "These are the planned features .. and fixes. These are what, uh, what we have prioritized with `PO`" (`D`)

The tension is resolved by refining the modeling to include the "fire-nature" of the work items and showing them in fires. The tension's root cause, of the model apparently omitting the `PO`'s existing priorization, was also discovered.

The team finishes up the priority lane by speeding through a few work items in the "todo" state, confirming them to be still relevant but undone. Once the list is combed, it's time for the next board.

Outbox The driver opens up *"OUTBOX (ska diskuteras snart)"* and commences the `OUTBOX` segment at **20:45** with a callout from the secretary. As per the Swedish language subtitle of the board, it is used to keep track of discussions that should take place.

`M1` immediately calls for a comment to be added on an existing work item that there's new evidence, referring to the `INTRO` segment discussion about a team being difficult to reach. This is apparently done for the benefit of a discussion to be had with `M2`, who is not present. Another discussion item is called out as being "longstanding", and is moved to that section of the board by the secretary.

Due to the excursion by `M1` of bringing a task in the middle of the board to the team's attention, the linear top-to-bottom fashion of reading the board that was seemingly faithfully observed before has broken down at this point, but is resumed. To-be-had discussions are gone through and confirmed relevant in much the same fashion as the todo items in the previous segment.

In the following excerpt there's again a reference to the difficulties discussed in the `INTRO`, now accompanied with a desire to not only have a discussion but to *keep track* of whether a problem was in fact solved for good:

22:15 " 'Tech team doesn't see documentation being handled.' Really don't!" (`M` quoting work item title)
"Well .." (`QA`)
"Again. This morning." (`M`)
"I have bumped and it's-" (`QA` stating a discussion was had)
22:25 "Yeah but until we can really say that it's now being properly handled . . . we see a few cycles where it just .. goes well" (`M`)

Ostensibly this would mean that the discussion item will be held in the "longstanding" section for evaluation, even if such a discussion as intended by the item did in fact take place as claimed by `QA`.

Lastly, we observe a callback to the issue with fires and work in the module continuum. There's an unhandled item for which the discussion has already taken place, but the team would need to be informed about the outcome.

23:36 " 'Ability to track module fires' " (`M` reading out item title as a prompt)
"Yeah, `PO` requested that-that any uhh module fires also be added to the module continuum so you can take a look at the module continuum afterwards and see which have been completed" (`D` relating the result of the discussion)
"Yes yes, very good" (`M`)

The PO's request itself happens to be an insignificant detail from the daily's perspective, because as shown in A1, the module continuum is never combed through in the daily. It provides another rationale for the tension observed earlier, however: the PO's wishes about work visualization in the continuum might have contributed to a conflict with the team's needs about transparency.

The "longstanding" discussion items are skipped with a callout from `M` getting an affirmative response from the secretary. This concludes the `OUTBOX` segment.

Questions By this point, the team is evidently anxious to move forward: several members in succession call out the name of the next board, *"QUESTIONS"*. The purpose of the board is to make sure that information needs of parties, internal and customer alike, are not neglected. Concretely, there are both ongoing lines of inquiry and recurring "household keeping" themed tasks. Only the topmost tasks, belonging to the recurring category, are briefly confirmed as having been done.

24:39 "So ..." (`D1`)
24:41 "So ..." (`D2`)
24:43 "Let's not be done yet." (`M`)

Team members express desire to move on and perhaps finish the daily, but the manager wants to be more thorough. With that note, the driver shows the next board.

Non-functional continuum The non-functional continuum board consists of small tasks that have only superficial effect on the product and can thus be shipped immediately. The structure and sectioning is similar to "fires", but the work reflected therein is considered to be of less urgency by nature.

The manager appears to have had a clear agenda in moving to this board, because he proceeds to press on an uncompleted task.

24:48 “There’s [item description]” (`M`)
“Has `D1`’s face on it [..]” (`QA`)

25:02 “So `D1` could you add these things? There’s a huge organisational waste when we have done [website content] already .. and they are not linked in the UI” (`M`)

25:12 “Yeah yeah” (`D1`)
“This is almost becoming a fire” (`M`)
“… Yeah.” (`D1`)

25:21 “But like, we had this discussion before I left.” (`M`)
“Yeah yeah I can add” (`D1`)
“But .. really” (`M`)
“Yeah yeah yeah yeah yeah” (`D1`)

25:29 “And… Start using the ‘my tasks’ thing [..]” (`M`)
“That’s how I do it cause otherwise I just don’t remember what I …” (`D2`)

From the manager’s perspective, the team appears to have failed in accounting for the timeliness requirement of the task being discussed. Proximally, the reason is pinned on `D1` not using the team’s common tooling for supporting this by seeing that the task exists and is assigned. However, the team members seem to acknowledge that it is not trivial to make good priorizations of when these individual tasks would best be done, as they are unrelated to the team’s larger deliverables.

Board overview The `NON-FUNCTIONAL CONTINUUM` segment fluidly transforms into `BOARD OVERVIEW` when the manager again prompts:

26:05 “So what’s the status of these numbered things? `QA`?” (`M`, referring to ongoing cycles)

No action is needed from the driver as the discussion items of this section are continuously visible on screen. `QA` elaborates on the statuses from their perspective. A developer is curious about the color coding `QA` uses for signifying project statuses, and gets an explanation. This is immediately taken into use by another developer by describing the state of the next project on the list as "green", meaning "in progress".

26:50 "What's the definition of 'API'?" (`M` referring to cycle name)
"It was more that I didn't want to use the 'partner integration funnel' for my tasks-" (`D`)

27:01 "Okay.. could this be renamed so that it still means something? Like ... put in parenthesis that 'partner integration funnel'? Or something, because nobody has idea what's 'API' [in] this context" (`M`)

Manager calls for cycles being more clearly defined bundles of tasks, because a vague umbrella topic does not aid in understanding the work. The developer admits to not having been wanting to do this before. Perhaps the more specific name does not entirely describe the referred cycle's scope, or a better understanding has only recently arisen and the project's name is lagging behind.

Also of note is that the developer expresses having a sense of ownership over "his" tasks. There's apparently a domain that the developer personally is responsible for and this cycle would lie within that domain.

28:15 "Alright ... Now we're done." (`M`)

Nobody objects; the daily is concluded.

11.1.2 Relation between daily structure and kanban boards

The previous section answered the question of how the daily happened. To support an argument about a relation between the daily and its sociomaterial context, in this section we attempt to reach for an understanding on how the daily could have come to take this form.

Observing table 6 where the segments undergone in the daily and the kanban boards have been overlaid, we find that the boards are in the same order as the team uses when proceeding through the daily. This gives rise to the idea that the structure of a daily and the structure of the boards are interrelated. Indeed, this is not an uncommon occurrence: `INBOX` – `FIRES` - `PRIORITY LANE` - `OUTBOX` - `QUESTIONS` is a typical sequence through the observation period.

Segments	Kanban board titles
INTRO	-
INBOX	INBOX (Composer 2)
FIRES	[FIRE]: This must be emptied
PRIORITY LANE	Priority lane
OUTBOX	OUTBOX (ska diskuteras snart)
QUESTIONS	QUESTIONS
NON-FUNCTIONAL CONTINUUM	CONTINUUM: Non-functional
BOARD OVERVIEW	-

Table 6: List of daily segments and the kanban board list for daily `03` overlaid.

In support of this deduction we can can consider table 7. The segments `INBOX`, `FIRES`, `PRIORITY LANE`, `OUTBOX` and `QUESTIONS` occur from 17 to 21 times. Most other segments only get a handful of visits, with the runner-ups being `NON-FUNCTIONAL CONTINUUM` at 12 and `NON-CORE MODULE CONTINUUM` at 8 occurrences respectively. We could consider these top segments to form the stable skeleton of a daily.

Is it reasonable to infer a causality? Is the team's daily structure indeed encoded in the kanban boards' layout, or are the boards merely arranged in imitation of the daily? To answer this question, we may consider what happens when the boards are disturbed. Table 7 gives us a hint about where to look for such a disturbance: segments `MONEYBOX`, `MARKETING BOX` and `WATBOX` appear for the first time at the end of the observation period but are nowhere to be seen by daily `20`.

Table 7: First and last occurrences of segments through observation period with the count of occurrences in between.

Segment	First	Occurrences	Last
FIRES	1	21	20
INBOX	1	21	20
INTRO	1	18	20
OUTBOX	1	20	20
PRIORITY LANE	1	17	20
QUESTIONS	1	18	19
CYCLES	2	2	18
NON-CORE MODULE CONTINUUM	2	8	20
BOARD OVERVIEW	3	3	13
NON-FUNCTIONAL CONTINUUM	4	12	20
MODULES CONTINUUM	7	1	7
MONEYBOX	17	2	19
MARKETING BOX	19	1	19

Segment	First	Occurrences	Last
WATBOX	19	1	19
Total Result	1	145	20

How did the aforementioned "box" segments end up in the daily? To corroborate a hypothesis about a relation between the daily and the boards, a first order explanation of a change having been made to the boards would suffice. What kind of modifications were made to the board layout then, if any?

Prior to `19`	`19`
INBOX (Composer 2)	INBOX (Composer 2)
[FIRE]: This must be emptied	[FIRE]: This must be emptied
Priority lane	Priority lane
OUTBOX (ska diskuteras snart)	OUTBOX (ska diskuteras snart)
-	MONEYBOX
QUESTIONS	QUESTIONS
-	MARKETING
CONTINUUM: Non-functional	CONTINUUM: Non-functional
CONTINUUM: Modules	CONTINUUM: Modules

Table 8: Difference in board layout for daily `19`.

Boards associated with the segments `MONEYBOX` and `MARKETING BOX` appear in the layout among the most commonly used boards by daily `19`. `WATBOX` seems to be an anomaly, because although it is exhibited in the daily segment after `MONEYBOX` the board itself is still out of the way and would not even be visible on the driver's screen. Apparently there was a commonality between the three, however, because in daily `20` the team has again groomed the boards for `MONEYBOX` and `MARKETING BOX` outside the group of most common boards and together with `WATBOX`.

One interpretation for this series of events is that there occurred a disturbance which necessitated the provision of new places to put things for them to get done eg. at the appropriate timeliness. The boards are perused once or twice in dailies until the team figures that it's not getting any use out of doing this commonally, at which point the boards are moved out of the way of the ordinary daily flow.

From the looks of things, it does stand to reason that the structure of the board and the daily go hand in hand. If this is a credible statement, it should be a hint that the daily and its sociomaterial context are engaged in a recursive loop of unfolding driven by tensions introduced and resolved during the practice of a daily. It's too early to make that claim, however. Of course, we're only investigating the matter on the level of segments and boards and ignoring most of the story. In the next section we will take the chance to have a closer look at the dynamics and nature of this processual evolution more closely.

11.2 What a difference a daily makes

In the previous section we looked into how a daily happens and what kinds of activities a daily constitutes. We were left with the open question of *change*. If the daily is to be considered a practice, engaging in a daily should somehow inform the next occasions this engagement is taken and manifest as observable change in the practice. In this section we will lay the ground for answering **ERQ2** by investigating whether and how this change happens.

Our inquiry will be twofold. First we will check the material for episodes of questioning, disturbance and contradiction to see if the system evidences potential for change. Second, we will see if this change potential is realised as transformations. Within practices, contradictions unfold and improvement tends to happen slowly and must be observed over a duration of time. Our observations on change will take the form of journeys of transformation, taking place over multiple dailies. Let's proceed with a brief overview on how the stories were collected and structured overall, before looking at the stories one by one.

11.2.1 Episodes of contradiction

11.2.2 Journeys of transformation

In the introductory section the usefulness of a daily was set to question by the fact that it is not a software development activity in itself. To reiterate, no features are delivered, no bugs fixed and no customer served. Therefore, if a use for the daily is to be found, in that it somehow aids software development activities, the use must lie in the relation between the daily and other practices that *do* constitute software development. From the episodes delineated in 11.2.1, a set was chosen such that their orientation is *not* toward the daily and what happens within but impactful on the outside world. The most important 'story arcs', if you will, were extracted from their original context and presented here compressed form such that we may observe the structural tensions reflected and their resolution via a process of transformation.

We are swamped with fires, but are they really all equally important?

Minor change requests get in the way of overall progress, yet they're important

There is more than one kind of wishlist

The module release process is not working

Chapter IV

Conclusions

12 Results

12.1 Answers to the empirical research questions

> **ERQ1**: How does the observed daily practice compare to the agile description of a daily?

A scrum daily is: - short - determined in its goals by the sprint agenda - aimed at enabling progress with the currently known work items (section 5.3)

The purpose of the observed daily is to - React to new input - Address urgent, non-planned work - Reflect on overall status - Ensure necessary discussions are had outside

Scrum daily is for ensuring that the ball is passed around and advanced; the observed daily is for ensuring any new balls get to the playing field if necessary and that no balls are dropped because nobody is watching

12.2 Answer to the research problem

13 Implications of the study

13.1 Practical implications

13.1.1 The daily reflect's the team's understanding of the process of software development

13.1.2 The daily can be used as a tool for continuous improvement

13.1.3 The practice of a daily can inform software development activities

13.2 Theoretical implications

13.2.1 Overcoming tensions vs extending knowledge objects

In 7.1 we delimited the scope of this study to the activity theoretical perspective, in which we consider the process of innovation as that of collaborative contradiction resolution. This was a choice fitting for the investigation of the daily practice as the focal element, but there is another option. Consider the possibility where we wanted to instead investigate the specific way the team uses the kanban boards as a knowledge object and to refine their understanding of the activities which are placed outside the daily but modelled within. In this case we could've done well by picking the *conceptual artefact* based model of knowledge building by Bereiter (2002), contrasted to Engestrï¿œm's model we used here by Hakkarainen et al. (2004). A study where the conceptual artefacts are taken in focus could, for instance, look deeper into the team's use of their tools and how they reflect the conceptual models in use.

14 Evaluation

14.1 Limitations of the study

Appendix A

Dailies

A1 Daily segment lengths

Table 9: Identified segments in dailies by their timestamp in the recording and duration.

Daily index	Segment	Timestamp	Duration
1	INTRO	00:02:02	00:12:28
	INBOX	00:14:30	00:13:30
	FIRES	00:28:00	00:04:40
	PRIORITY LANE	00:32:40	00:01:20
	OUTBOX	00:34:00	00:00:20
	QUESTIONS	00:34:20	00:01:43
	DONE	00:36:03	00:00:33
	RECORDING ENDS	00:36:36	
2	INTRO	00:01:11	00:08:59
	NON-CORE MODULE CONTINUUM	00:10:10	00:02:30
	CYCLES	00:12:40	00:03:00

Daily index	Segment	Timestamp	Duration
	INBOX	00:15:40	00:13:20
	FIRES	00:29:00	00:13:55
	PRIORITY LANE	00:42:55	00:02:30
	OUTBOX	00:45:25	00:01:25
	QUESTIONS	00:46:50	00:01:40
	DONE	00:48:30	00:00:23
	RECORDING ENDS	00:48:53	
3	INTRO	00:00:40	00:10:45
	INBOX	00:11:25	00:01:45
	FIRES	00:13:10	00:04:05
	PRIORITY LANE	00:17:15	00:03:30
	OUTBOX	00:20:45	00:03:20
	QUESTIONS	00:24:05	00:02:00
	”””BOARD OVERVIEW””?”	00:26:05	00:02:13
	DONE	00:28:18	00:00:12
	RECORDING ENDS	00:28:30	
4	INTRO	00:00:00	00:04:40
	INBOX	00:04:40	00:12:10
	FIRES	00:16:50	00:09:40
	OUTBOX	00:26:30	00:03:20
	QUESTIONS	00:29:50	00:00:50
	NON-FUNCTIONAL CONTINUUM	00:30:40	00:01:45
	DONE	00:32:25	00:01:26
	RECORDING ENDS	00:33:51	
5	INTRO	00:00:19	00:07:56
	INBOX	00:08:15	00:07:25
	FIRES	00:15:40	00:05:50
	PRIORITY LANE	00:21:30	00:00:20

Daily index	Segment	Timestamp	Duration
	OUTBOX	00:21:50	00:02:20
	QUESTIONS	00:24:10	00:03:43
	DONE	00:27:53	00:00:23
	RECORDING ENDS	00:28:16	
6	INBOX	00:00:07	00:07:23
	FIRES	00:07:30	00:04:50
	PRIORITY LANE	00:12:20	00:01:40
	OUTBOX	00:14:00	00:01:10
	QUESTIONS	00:15:10	00:00:20
	DONE	00:15:30	00:02:17
	RECORDING ENDS	00:17:47	
7	INTRO	00:00:13	00:03:17
	INBOX	00:03:30	00:13:00
	FIRES	00:16:30	00:05:10
	PRIORITY LANE	00:21:40	00:01:05
	OUTBOX	00:22:45	00:00:55
	QUESTIONS	00:23:40	00:01:45
	NON-FUNCTIONAL CONTINUUM	00:25:25	00:01:05
	MODULES CONTINUUM	00:26:30	00:00:25
	DONE	00:26:55	00:00:13
	RECORDING ENDS	00:27:08	
8	INTRO	00:00:20	00:06:25
	INBOX	00:06:45	00:07:00
	FIRES	00:13:45	00:02:15
	PRIORITY LANE	00:16:00	00:02:00
	QUESTIONS	00:18:00	00:00:30
	NON-FUNCTIONAL CONTINUUM	00:18:30	00:01:40
	DONE	00:20:10	00:00:16

Daily index	Segment	Timestamp	Duration
	RECORDING ENDS	00:20:26	
9	INTRO	00:00:30	00:01:40
	INBOX	00:02:10	00:05:20
	FIRES	00:07:30	00:03:05
	OUTBOX	00:10:35	00:01:10
	QUESTIONS	00:11:45	00:02:15
	NON-FUNCTIONAL CONTINUUM	00:14:00	00:01:10
	BOARD OVERVIEW?	00:15:10	00:02:40
	DONE	00:17:50	00:01:13
	RECORDING ENDS	00:19:03	
10	INTRO	00:00:25	00:05:55
	INBOX	00:06:20	00:05:39
	FIRES	00:11:59	00:10:11
	PRIORITY LANE	00:22:10	00:02:35
	OUTBOX	00:24:45	00:05:15
	QUESTIONS	00:30:00	00:00:35
	DONE	00:30:35	00:00:38
	RECORDING ENDS	00:31:13	
11	INBOX	00:00:15	00:05:30
	FIRES	00:05:45	00:02:00
	PRIORITY LANE	00:07:45	00:00:30
	OUTBOX	00:08:15	00:00:10
	QUESTIONS	00:08:25	00:02:57
	DONE	00:11:22	00:00:35
	RECORDING ENDS	00:11:57	
12	INTRO	00:00:08	00:07:52
	INBOX	00:08:00	00:03:20

Daily index	Segment	Timestamp	Duration
	FIRES	00:11:20	00:09:36
	PRIORITY LANE	00:20:56	00:01:29
	OUTBOX	00:22:25	00:09:35
	QUESTIONS	00:32:00	00:00:22
	DONE	00:32:22	00:02:06
	RECORDING ENDS	00:34:28	
13	INTRO	00:00:58	00:00:47
	BOARD OVERVIEW	00:01:45	00:15:45
	INBOX	00:17:30	00:20:18
	FIRES	00:37:48	00:05:32
	PRIORITY LANE	00:43:20	00:00:28
	OUTBOX	00:43:48	00:00:32
	QUESTIONS	00:44:20	00:00:20
	NON-FUNCTIONAL CONTINUUM	00:44:40	00:00:35
	NON-CORE MODULE CONTINUUM	00:45:15	00:01:35
	DONE	00:46:50	00:00:43
	RECORDING ENDS	00:47:33	
14	INTRO	00:00:26	00:02:07
	INBOX	00:02:33	00:18:52
	FIRES	00:21:25	00:04:15
	PRIORITY LANE	00:25:40	00:00:35
	OUTBOX	00:26:15	00:06:15
	QUESTIONS	00:32:30	00:00:45
	NON-FUNCTIONAL CONTINUUM	00:33:15	00:01:45
	NON-CORE MODULE CONTINUUM	00:35:00	00:00:42
	DONE	00:35:42	00:00:24
	RECORDING ENDS	00:36:06	
15	INTRO	00:00:25	00:01:35

Daily index	Segment	Timestamp	Duration
	INBOX	00:02:00	00:05:55
	NON-FUNCTIONAL CONTINUU	00:07:55	00:05:05
	INBOX AGAIN	00:13:00	00:14:55
	FIRES	00:27:55	00:06:40
	PRIORITY LANE	00:34:35	00:02:40
	OUTBOX	00:37:15	00:08:05
	QUESTIONS	00:45:20	00:02:25
	NON-FUNCTIONAL CONTINUUM	00:47:45	00:01:15
	NON-CORE CONTINUUM	00:49:00	00:02:28
	DONE	00:51:28	00:00:16
	RECORDING ENDS	00:51:44	
16	INTRO	00:00:12	00:12:33
	OUTBOX	00:12:45	00:20:05
	PRIORITY LANE	00:32:50	00:00:30
	OUTBOX	00:33:20	00:07:13
	INBOX	00:40:33	00:15:30
	FIRES	00:56:03	00:03:45
	DONE	00:59:48	
	RECORDING ENDS	01:00:21	
17	INTRO	00:01:11	00:01:54
	FIRES	00:03:05	00:12:47
	INBOX	00:15:52	00:22:08
	FIRES	00:38:00	00:00:25
	PRIORITY LANE	00:38:25	00:01:05
	OUTBOX	00:39:30	00:04:00
	MONEYBOX	00:43:30	00:00:45
	QUESTIONS	00:44:15	00:00:20
	NON-FUNCTIONAL CONTINUUM	00:44:35	00:05:45
	NON-CORE MODULES	00:50:20	00:02:53

Daily index	Segment	Timestamp	Duration
	DONE	00:53:13	00:00:22
	RECORDING ENDS	00:53:35	
18	INTRO	00:00:15	00:03:20
	CYCLES	00:03:35	00:12:20
	INBOX	00:15:55	00:12:50
	FIRES	00:28:45	00:06:20
	PRIORITY LANE	00:35:05	00:00:15
	OUTBOX	00:35:20	00:02:25
	QUESTIONS	00:37:45	00:00:40
	NON-FUNCTIONAL	00:38:25	00:00:25
	NON-CORE MODULES	00:38:50	00:01:25
	DONE	00:40:15	00:00:14
	RECORDING ENDS	00:40:29	
19	INTRO	00:01:15	00:09:55
	INBOX	00:11:10	00:20:20
	FIRES	00:31:30	00:12:45
	OUTBOX	00:44:15	00:01:18
	MONEYBOX	00:45:33	00:00:04
	WATBOX	00:45:37	00:00:13
	QUESTIONS	00:45:50	00:00:30
	MARKETING BOX	00:46:20	00:00:13
	NON-CORE MODULE CONTINUUM	00:46:33	00:00:57
	NON-FUNCTIONAL CONTINUUM	00:47:30	00:02:10
	DONE	00:49:40	00:00:17
	RECORDING ENDS	00:49:57	
20	INTRO	00:00:16	00:04:04
	INBOX	00:04:20	00:18:25
	FIRES	00:22:45	00:10:55

Daily index	Segment	Timestamp	Duration
	PRIORITY LANE	00:33:40	00:01:45
	OUTBOX	00:35:25	00:02:35
	NON-FUNCTIONAL CONTINUUM	00:38:00	00:00:45
	NON-CORE MODULE CONTINUUM	00:38:45	00:00:55
	DONE	00:39:40	00:01:31
	RECORDING ENDS	00:41:11	

Appendix B

Kanban boards

B2 Lists of kanban boards

Table 10: Partial outlines of the kanban boards used by the team ordered by date.

Capture date	Boards
2015-12-08	INBOX (Composer 2) [FIRE]: This must be emptied Priority lane OUTBOX (ska diskuteras snart) QUESTIONS CONTINUUM: Modules CONTINUUM: Non-functional (omitted)
2015-12-16	INBOX (Composer 2) [FIRE]: This must be emptied Priority lane

Capture date	Boards
	OUTBOX (ska diskuteras snart)
	QUESTIONS
	CONTINUUM: Non-functional
	CONTINUUM: Modules
	#15c: iOS 4.0.8
	#17 Android 4.2.x
	#19 Composer 2
	#20 Partner API
	#20 Partner integration funnel
	#20 Release Wizard
	#21 Logic editor
	#?? DreamFactory Funnel
	#?? Composer 2
	#?? iOS
	#?? Android
	#?? Payments
	#?? Dolan
	#2X API and Friends
	(omitted)
2016-01-18	INBOX (Composer 2)
	[FIRE]: This must be emptied
	Priority lane
	OUTBOX (ska diskuteras snart)
	MONEYBOX
	QUESTIONS
	MARKETING
	CONTINUUM: Non-functional
	CONTINUUM: Modules
	#17 Android 4.2.0 + 4.2.1
	#22 Composer 2

Capture date	Boards
	#23 Release Lifecycle
	#23 Composer 2
	#23 API and ACL
	#24 Release Lifecycle
	#24 Logic editor
	#24 Composer 2
	#?? iOS 4.1.0
	#?? Android 4.3
	#?? Payments
	#?? Dolan
	#?? Module core
	#?? API and friends
	#?? Push Notifications
	#?? SandboxDB 2
	INBOX (Kisko)
	Exception Box
	—
	#NN Descriptive Theme Name
	#NN Release Lifecycle
	—
	BACKLOG: Modules
	—
	MINOR Roadmap
	SOLUTION Roadmap
	Composer 2 bug backlog
	Toolchain bug backlog
	SORT THIS AWAY PLS
	NEXT
	SOON
	LATER
	WAT

Capture date	Boards
	HARD AND SCARY
	WATBOX
	TECH WISHING WELL
	(omitted)
2016-01-19	(omitted)
	INBOX (Kisko)
	MONEYBOX
	MARKETING
	TECH WISHING WELL BOX
	WATBOX
	EXCEPTION BOX
	BACKLOG: Modules
	(omitted)

B3 Lists of kanban board structures

Table 11: Partial outlines of the structures of the kanban boards used by the team ordered by date of capture.

Capture date	Board name	Labels
2015-12-16	CONTINUUM: Modules	(unlabeled)
		!! NOT AN INBOX !!
		SMOKETEST FAILED (in production)
		DEPLOYMENTS (make sure all subtasks a
		WAITING FOR DEVQA (in devgyver)
		DEVQA REJECTED (in devgyver)
		WAITING FOR MERGE (in branch)
2015-12-16	[FIRE]: This must be emptied	(unlabeled)

Capture date	Board name	Labels
		UNHANDLED FIRES INBOX AWAITING SMOKE TEST (in production) SMOKETEST FAILED DEPLOYMENTS (make sure all subtasks a WAITING IN DEVGYVER FOR MERGE IMPEDED IN DEV QA BACKLOG (waiting to be picked on the tab WAITING FOR REPRO LONGSTANDING (extinguished on its own
2015-12-16	INBOX (Composer 2)	(unlabeled) INBOX TUTORIAL Inbox Needs something Discuss later Wishlist inbox Wishlist (rest omitted)
2015-12-16	OUTBOX (ska diskuteras snart)	(unlabeled) DONE TODO IMPEDED Longstanding (WIP limit: 18)
2015-12-16	Priority lane	(unlabeled) DONE DOING TODO LONGSTANDING

Capture date	Board name	Labels
2015-12-16	QUESTIONS	(unlabeled) Inbox Updates asked for Seen Badly needs a repro Client Plugins Misc CLI Needs repro
2015-12-21	[FIRE] This must be emptied	(omitted) WAITING IN DEVGYVER FOR MERGE IMPEDED IN DEV QA UNICORN BACKLOG (wip: 1) JUNIPER BACKLOG (wip: 1+1) BACKLOG (wip: infinite) WAITING FOR REPRO LONGSTANDING (extinguished on its own
2015-12-21	OUTBOX (ska diskuteras snart)	(unlabeled) INBOX ABOVE DONE ALMOST-BUT-NOT-QUITE-DONE TODO IMPEDED Longstanding (WIP limit: 18)
2016-01-07	OUTBOX (ska diskuteras snart)	(unlabeled)

Capture date	Board name	Labels
		INBOX ABOVE
		DONE
		ALMOST-BUT-NOT-QUITE-DONE
		TODO
		IMPEDED
		MISSING IN ACTION
2016-01-12	MONEYBOX	(unlabeled)
		PENDING CUSTOMER ACTION
2016-01-12	OUTBOX (ska diskuteras snart)	(omitted)
		DONE
		ALMOST-BUT-NOT-QUITE-DONE
		TODO
		BOUNCED
		NEEDS PLANNING
		IMPEDED
		MISSING IN ACTION

Bibliography

P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta. *Agile software development methods. Review and analysis.* VTT Technical Research Centre of Finland, 2002.

Paul S Adler. The evolving object of software development. *Organization*, 12 (3):401–435, 2005.

M Omair Ahmad, Jouni Markkula, and Markku Oivo. Kanban in software development: A systematic literature review. In *Software Engineering and Advanced Applications (SEAA), 2013 39th EUROMICRO Conference on*, pages 9–16. IEEE, 2013.

David J Anderson. *Kanban.* Blue Hole Press, 2010.

Paulo Barthelmess and Kenneth M. Anderson. A view of software development environments based on activity theory. *Computer Supported Cooperative Work (CSCW)*, 11(1-2):13–37, 2002.

K. Beck, M. Beedle, A. Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, B. Kern, J Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas. Manifesto for agile software development. 2001. URL `http://agilemanifesto.org`.

C Bereiter. Education and mind in the knowledge society, 2002.

Nadia Bhuiyan and Amit Baghel. An overview of continuous improvement: from the past to the present. *Management Decision*, 43(5):761–771, 2005.

Harry Boer and Frank Gertsen. From continuous improvement to continuous innovation: a (retro)(per) spective. *International Journal of Technology Management*, 26(8):805–827, 2003.

Yrjo Engestrom. Learning by expanding. *Helsinki: Orienta-Konsultit Oy*, 1987.

Yrjo Engestrom. Activity theory as a framework for analyzing and redesigning work. *Ergonomics*, 43(7):960–974, 2000.

Yrjö Engeström. Expansive learning at work: Toward an activity theoretical reconceptualization. *Journal of education and work*, 14(1):133–156, 2001.

Yrjö Engeström and Annalisa Sannino. Studies of expansive learning: Foundations, findings and future challenges. *Educational research review*, 5(1): 1–24, 2010.

Yrjö Engeström, Reijo Miettinen, and Raija-Leena Punamäki. *Perspectives on activity theory*. Cambridge University Press, 1999.

Richard P Feynman. Cargo cult science. *The Art and Science of Analog Circuit Design*, page 55, 1998.

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

Silvia Gherardi. Organizational learning: The sociology of practice. *Handbook of Organizational Learning and Knowledge Management*, 2:43–65, 2011.

Robert L. Glass. Agile versus traditional: Make love, not war! *Journal of Information Technology Management*, 14:12(12):s. 12–17, 2001.

Kai P Hakkarainen, Tuire Palonen, Sami Paavola, and Erno Lehtinen. Communities of networked expertise: Professional and educational perspectives. 2004.

J. Highsmith and A. Cockburn. Agile software development: The business of innovation. *IEEE Computer*, 34:9(9):s. 120–127, 2002. ISSN 0018-9162.

M. Huo, J. Verner, L. Zhu, and M.A. Babar. Software quality and agile methods. In *Proceedings of the 28th Annual International Computer Software and Applications Conference*, 2004.

Marko Ikonen, Petri Kettunen, Nilay Oza, and Pekka Abrahamsson. Exploring the sources of waste in kanban software development projects. In *Software Engineering and Advanced Applications (SEAA), 2010 36th EUROMICRO Conference on*, pages 376–381. IEEE, 2010.

Victor Kaptelinin. Activity theory: Implications for human-computer interaction. *Context and consciousness: Activity theory and human-computer interaction*, (1):103–116, 1996.

Hannele Kerosuo, Anu Kajamaa, and Yrjö Engeström. Promoting innovation and learning through change laboratory: An example from finnish health care. *Central European Journal of Public Policy*, 4(1):110–131, 2010.

P. Kruchten. Voyage in the agile memeplex. *ACM Queue*, 5:5(5):s. 38–44, 2007. ISSN 1542-7730.

Kari Kuutti. Activity theory as a potential framework for human-computer interaction research. *Context and consciousness: Activity theory and human-computer interaction*, pages 17–44, 1996.

L. Lindstrom and R. Jeffries. Extreme programming and agile software development methodologies. *Information Systems Management*, 21:3(3):s. 41–52, 2004.

A. Marchenko and P. Abrahamsson. Scrum in a multiproject environment: An ethnographically-inspired case study on the adoption challenges. In *Agile, 2008. AGILE '08. Conference*, pages 15–26, Aug 2008. doi: 10.1109/Agile.2008.77.

Reijo Miettinen. The sources of novelty: A cultural and systemic view of distributed creativity. *Creativity and Innovation Management*, 15(2): 173–181, 2006.

Davide Nicolini. Zooming in and out: Studying practices by switching theoretical lenses and trailing connections. *Organization Studies*, 30(12): 1391–1418, 2009.

Davide Nicolini, Silvia Gherardi, and Dvora Yanow. *Knowing in organizations: A practice-based approach.* ME Sharpe, 2003.

Ikujiro Nonaka and Hirotaka Takeuchi. *The knowledge-creating company: How Japanese companies create the dynamics of innovation.* Oxford university press, 1995.

Taiichi Ohno. *Toyota production system: beyond large-scale production.* crc Press, 1988.

Mary Poppendieck and Tom Poppendieck. *Lean Software Development: An Agile Toolkit: An Agile Toolkit.* Addison-Wesley, 2003.

Linda Rising and Norman S Janoff. The scrum software development process for small teams. *IEEE software*, 17(4):26, 2000.

Hugh Robinson, Judith Segal, and Helen Sharp. Ethnographically-informed empirical studies of software practice. *Information and Software Technology*, 49(6):540 – 551, 2007. ISSN 0950-5849. doi: http://dx.doi.org/10.1016/j.infsof.2007.02.007. URL `http://www.sciencedirect.com/science/article/pii/S0950584907000110`. Qualitative Software Engineering Research.

K. Schwaber. Scrum development process. In *OOPSLA Business Object Design and Implementation Workshop*, pages 10–19, 1995.

Ken Schwaber and Jeff Sutherland. The scrum guide, 2011.

Carol Slappendel. Perspectives on innovation in organizations. *Organization Studies*, 17(1):107–129, 1996.

Jürgen Streeck and Siri Mehus. Microethnography: The study of practices. *Handbook of language and social interaction*, pages 381–404, 2005.

Jeff Sutherland. Agile can scale: Inventing and reinventing scrum in five companies. *Journal of Information Technology Management*, 14:12(12):s. 5–11, 2001.

Jeff Sutherland and JJ Sutherland. *Scrum: the art of doing twice the work in half the time.* Crown Business, 2014.

Hirotaka Takeuchi and Ikujiro Nonaka. The new new product development game. *Harvard business review*, 64(1):137–146, 1986.

Dave West, Tom Grant, M Gerush, and D D'silva. Agile development: Mainstream adoption has changed agility. *Forrester Research*, 2:41, 2010.

Margaret J Wheatley and Myron Kellner-Rogers. Self-organization: The irresistible future of organizing. *Strategy & Leadership*, 24(4):18–24, 1996.

Jason Yip. It's not just standing up: Patterns of daily stand-up meetings. 2006.