On the Impact of Kanban on Software Project Work: An Empirical Case Study Investigation

Marko Ikonen  Elena Pirinen  Fabian Fagerholm  Petri Kettunen  Pekka Abrahamsson
Department of Computer Science
P. O. Box 68, FI-00014 University of Helsinki, Finland
{firstname.lastname}@cs.helsinki.fi

Abstract—The pertinent mission of software project management is to continuously achieve more and more successful projects. In the field of software development, the Kanban method has gained momentum recently, mostly due to its linkages to Lean Thinking. However, only a few empirical studies investigate the dynamics and impacts of Kanban on project success. The aim of this study is to improve the understanding on how Kanban impacts on software project work. For the purpose of the study, a framework is developed and empirically investigated in an experimental software R&D setting called Software Factory in a single case setting. The impact of Kanban is evaluated from nine theoretically derived perspectives. The qualitative findings indicate considerable positive support for the application of Kanban in the case setting. This bears direct managerial implications, which are addressed. The key implications of the findings suggest that Kanban and its inherent simplicity increase project visibility and thereby improve the ability to steer the project better.

Index Terms—software development, Kanban, project management, process model

I. INTRODUCTION

The principles of Lean Thinking are grounded in the ways of how to add value for the customer [1]. Such an increase in value comes from the features and quality of products to the ability to deliver and adapt quickly to changes. In brief, every non-value-adding activity (NVA), variations (in process quality, cost, delivery), and unreasonableness (overburden) should be eliminated. In common, these things are considered waste. The other way down, that relates to the most important principle of Lean Thinking: eliminating waste.

The Lean philosophy requires continuous growth and commitment from personnel as well as management. This is similar to the concept of learning organization [2] wherein the success and surveying of the organization is based on commitment and lifelong learning.

While the Lean philosophy has turned out to be a success in manufacturing [1], it has recently been applied to the area of software engineering, as well [3]. The basis of the repeatable manufacturing process differs from the idea of traditional software projects 1. Despite this difference, the Lean philosophy can be applied, among other areas, to software development projects [5], [6], [7]. In addition to waste elimination, Lean Software Development principles can be defined as follows: amplify learning, decide as late as possible, deliver as fast as possible, empower the team, build integrity in, and see the whole [8]. Moreover, the creativeness and innovativeness of the workers should be utilized [6]. In a wider context, the production should be conceptualized as a flow [9]. The Kanban method is often presumed to make this attainable [3], [10].

Following the Lean Manufacturing practices, the outcomes of applying Kanban are expected to be high in software development, as they have been in manufacturing. If it is really so, the use of Kanban would benefit software engineering management as well. This belief is, however, yet unconfirmed. Only a few studies exist where empirical investigations into the dynamics of Kanban have been explored from the viewpoint of software development.

Following this line of thinking, this paper investigates how Kanban influences software development and, consequently, project work. More specifically, the management-related research question is set as follows: What are the perceived impacts of Kanban on software project work? This is particularly important as projects “are conceived and completed by people” [11]. Howell et al. [11] argue that we still lack the theoretical foundation that connects leadership and people aspects holistically. Reaching an optimal flowing state in the development process and its tasks means that the amount of waste, such as unnecessary waiting and non-value adding extra work, has been minimized. Kanban, as a method, provides a way to visualize this flow of tasks and, thereby, attempts to reveal problems in the flow. Kanban empowers people with a minimum set of required rules to follow.

The study conducts an empirical investigation related to Kanban software development. Based on the existing literature, the paper builds a theoretical framework modeling effects of Kanban on software project work. This framework recognizes nine perspectives which are then evaluated by conducting a case project in an experimental software R&D setting where a product was intended for real markets. The results from the empirical evaluation support some arguments of the literature. In addition, some new findings are encountered. Moreover, the results show that the framework (Section II-C) offers a valid base for exploring the impacts of Kanban on software project work.

The rest of the paper is organized as follows. Section II reviews the background and related work of Kanban software development. Moreover, it proposes a research framework.

---

1 A temporary endeavor undertaken to create a unique product, service, or result.” [4]
Section III then describes our case-study research setting and the research method. Section IV documents the empirical results while Section V analyzes them. Finally, Section VI draws conclusions with pointers to further research.

II. RELATED WORK

A. Kanban Software Development

Kanban executes the Lean philosophy in practice [12], [13] and is one of the key operation management tools in Lean manufacturing [6]. It drives project teams to visualize the workflow, limit work in progress (WIP) at each workflow stage, and measure the cycle time (i.e., average time to complete one task) [14].

Kanban-driven operations help the teams to keep inventories (simultaneous WIP) under control, which, on its part, helps to balance the overall production flow. As a result, the Kanban method attempts to lower production costs, increase quality, and accelerate lead time [6]. Meanwhile, inventories and problems caused by sudden changes will become more apparent.

However, while Kanban enables us to clarify workers’ awareness of the current production issues and forthcoming tasks, it does not recommend any particular project phases, milestones, or partitioning tasks. Because of its closeness to the “do whatever” rule, a project team has to build and customize the appropriate practices for its project. When successful, the impact on the project is supposed to be positive.

Some such practices have been suggested in the literature. Ladas [10], for example, suggests that the amount of tasks in progress simultaneously should be adjusted to the reasonable capacity in use. Middleton [7] claims that this amount should be minimized in order to keep quality high. Shinkle [3], instead, argues that minimizing this amount is not the best solution.

Project flow stages wherein the tasks progress from stage to stage have been suggested as well. If each stage, such as todo, planning, design, and coding, gets its own “ready” stage, the blockages in the workflow become more visible [10]. Alwardt et al. [15] state that tasks should be prioritized. Moreover, laborious tasks should be partitioned before setting them as assigned [3].

While the operations management field has investigated Kanban-based production for years, not much empirical evidence is published in software development. It follows that the presumed advantages and suggested control rules (e.g., WIP sizes) remain largely anecdotal and not necessarily generally applicable. The purpose of this study is to shed light on this gap in a well-defined, controlled software project environment.

B. Software Project Management Challenges

Boddy and Macbeth [16] address four practices explaining successful implementation of projects: goals, resources, structures and controls. These practices create the need for managers to focus on the following five areas: (1) ensuring agreement with goals, (2) obtaining resources, (3) monitoring and learning, (4) exercising influence (using individual initiative and creating appropriate structures), and (5) ensuring effective communication [17].

Since software processes are largely dynamic and cooperative activities, the impact of software process technology with its modeling languages, editors, and interpreters has been limited [18]. In the 1980s, crucial support processes, such as learning, technical communication, requirements negotiation, and customer interaction, were poorly described in software process models [19]. The reasons for most project failures were seen in the managerial, not in the technical sector [20], [21]. Project failures can often be traced to poor team performance. This is caused by inadequate attention to people and teamwork issues [22] because people have a tendency to concentrate on the technical aspects (hardware and software) rather than the peopleware and impermanence [23].

Team-based learning, group behavior, and ways in which people create purposes and communicate are key elements of the group formation [24]. In software development, Moe et al. [25] address five dimensions in order to improve the teamwork: shared leadership, team orientation, redundancy, learning, and autonomy. When enabling the group to execute its process, efficient information sharing emerges [24].

The most salient problems reported in projects concerning additional efforts or mistakes include communication and coordination breakdowns [19], [22], [26]. Teasley et al. [27] even show that the productivity in teams can be doubled by easing their access to each other and by making work artifacts visible to all. Moreover, Addison and Vallabh [28] identify that two of the most frequent risk factors in software projects are unrealistic schedules and budgets, and continuous requirement changes. Even 11 years before this identification, Boehm [20] ranked these two factors among the top six of the software risk factors.

In Kanban-driven software development, the threat of these risks is lower: Related to the schedule risk, once the duration of a project has been determined, the customer of the project prioritizes (typically in cooperation with the developers) the goals. Due to the reasonably fast task flow and short feedback loops (ensuring that tasks are kept small), the schedule and budget can be specified early. Regarding the scope risk, Kanban has been designed for continuous requirement changes. At the beginning of a project, determining all the requirements precisely would be hard or even impossible. Thereby, one principle of the Lean philosophy, “decide as late as possible”, is to make the decisions with more fact-based information and with less guessing [8]. When a project moves on, more knowledge is attained, which helps the customer decide on current, true needs instead of “just-in-case” features. Due to the small tasks, the requirement changes of the project are better under control.

C. Research Framework

In order to analyze the effects of Kanban upon software project management, a research framework is constructed. The framework (Table I) comprises nine literature-based aspects of
The waterfall model supports understanding the whole since the system is known after the specification phase [29]. Again, the agile principles explicitly ignore this despite that customer satisfaction, one key of Agile Manifesto [30], is a part of seeing the whole [34]. One of the Lean principles, in contrast, is seeing the whole and the Kanban board shows the work in progress, what needs to be done, and what has been done already. However, the board does not tell us how much work there is altogether [8].

The waterfall model does not encourage informal communication [8]. Each phase, instead, produces explicit documentation which the developers follow in the next phase. Agile Manifesto [30], instead, emphasizes communication. In Kanban, communication is free and open according to the Lean philosophy [35]. Communication is regarded as a good thing and attempts are made to remove all the obstacles in its way.

Poppendieck and Poppendieck [8] suggest that if a process model is not easy to adopt, the developers will do their work in their own way and ignore the method. The waterfall model is considered simple to explain and recall but it gives the illusion of an orderly, measurable, and accountable process [36]. Related to embracing the method, the waterfall method requires deep knowledge of the problem field, process, and software being built in order to be successful [36], [37]. Larman [33] agrees and argues that the waterfall method is suitable for producing products in which the process is highly identical. Agile process models, such as Scrum [33], typically need some earlier experience from the developers, before reaching an efficient way of carrying out the progress. Larman [33], for example, mentions common mistakes and misunderstandings related to the lack of experience [33]. The Kanban method is intuitive to understand and gives rather free hands to the developers to do their work [8]; the only rule regarding this is that the workflow must be visualized.

In the waterfall model, giving feedback may be limited except between the phases [32]. Feedback is considered to be a threat to the predefined plan because corrective actions would change the plan. Agile Manifesto [30], as stated above, welcomes even late changing requirements in development. Moreover, it encourages collaborating with the customer rather than negotiating contracts. In Kanban, feedback from the customer as well as from the other developers is frequent and regular [8].

The approval process is often overly strenuous when using the waterfall model [8]. After each phase the documentation is approved by management. Agile Manifesto [30] stresses the meaning of self-organized teams. In a similar way, Lean
principles emphasize that the best expertise is at each workstation and no massive approval processes need to be performed [8]. There is neither need nor time to carry commands and instructions back and forth in the organization.

Finally, the waterfall model does not give instructions on how the work tasks of work are selected. Both agile and Lean philosophy prioritize the satisfaction of the customer through a valuable product. In the Kanban method, developers can select their own work according to the Lean philosophy [8]. The work is self-organizing.

III. EMPIRICAL RESEARCH DESIGN

This section describes the empirical case study conducted. The research environment (Section III-A) is an academic but industry-like R&D laboratory devised for empirical software engineering research. In this particular case project the product was targeted for real markets. By applying the research framework defined in Section II-C, the case project was investigated with observations and interviews (Section III-B).

A. Research Environment and Case Project

Software Factory is a new software engineering research and education setting at the University of Helsinki [38]. It is basically an industry-oriented R&D laboratory environment for conducting software business projects. The concept comprises a physical laboratory environment coupled with a novel operational model. The laboratory room is equipped with sophisticated computer and monitoring equipment and equipment for software development, such as Smartboards. These high-end facilities make it possible not only to conduct actual software engineering work in a modern fashion but also to collect online research data automatically (e.g., logs). Moreover, the facility enables capturing rich insights into the human-related aspects of software development [39]. The Department of Computer Science hosts an initial reference implementation, with more locations coming up globally. This facility has been operating since the beginning of 2010.

Software Factory runs projects continuously in seven-week cycles. In this study, a participant observation in one particular project was conducted. The observation was performed by being physically present in the Factory but without participating in the project work. The project developed a Web application for real markets. Most of the team members had experience working in industry. The project team size was 13 persons. In addition to technical knowledge, each team member had experience in working in a team-based environment. Besides the use of Kanban, no other particular development method was insisted upon. The team had close to full control within the R&D setting to decide upon the practices used. The team leader was not appointed by external mandate; rather, he took his role as a part of the self-organizing activities of the team. More experienced members (called seniors) took responsibility for designs. They also assisted less experienced members (called juniors) in technical and practical issues and gave useful advice. In short, the project team was self-organized.

B. Research Method

After constructing the research framework (Table I) based on the literature, the case project data were used for its empirical evaluation. The research methods in the case study were direct observation, video observation, and thematic interviews [40]. The direct observation lasted eight days while the video observation was focused on a particular session involving the development team and the customer representative working together with the Kanban board.

After the project, six software developers including the team leader of the 13 team members were interviewed. The case study focused on the team level because the work aspects (Table I), such as communication or feedback, relate mostly to the team level and are part of interpersonal interaction. The developers were chosen on the ground of their roles in the team: one was the team leader, two were senior developers who had more experience of software development, and three were junior developers. Each interview lasted about one hour. The observations were recorded in a diary and the interviews were taped. Open-ended questions and the semi-structured theme interview technique were used. The questions were compiled following the nine aspects of work (Table I) and are presented in Table II. The reasoning behind the questions was to find out how each particular aspect was experienced and done in the project.

IV. RESULTS

This section evaluates the research model (Table I) with the empirical case study data regarding the impacts of Kanban on the work of a software developer as stated in Section II-C. All nine aspects of the model are involved. The main theme questions used for the interviews are presented in Table II. Moreover, the questions of the thematic interviews steered the video observation by focusing the scope on the nine aspects. The quotations presented below come from the interview recordings. For identification the developers have been numbered from D1 (developer #1) to D6 (developer #6). Developer #1 was the team leader.

Documentation. Documentation was minimal; the customer did not express any need for documents so the team did only the documents they needed for their own work, as a member of the project stated:

“The customer was not interested in reading any documents so we had no need to make them.” [D2]

Problem solving. Problems were solved as soon as they occurred, as the observation revealed: A developer came to work at 8:45 AM and asked right away: “Why are the tests failing?” Then, another developer started to help him immediately.

If a developer had a problem with his or her tasks, it had to be solved before he or she could take on a new task. Therefore, problems appeared as blockages in the workflow. The ability

---

2Not to be confused with the early ‘soft脱颖而出 factories’ in Japan and in the USA of 1960–1980s.
TABLE II
THE OPEN-ENDED QUESTIONS USED IN THE THEME INTERVIEW.

<table>
<thead>
<tr>
<th>Aspect of Work</th>
<th>Questions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documentation</td>
<td>How would you describe the documents that were made?</td>
</tr>
<tr>
<td></td>
<td>How much documents? Why so little?</td>
</tr>
<tr>
<td>Problem solving</td>
<td>What caused problems in the development?</td>
</tr>
<tr>
<td></td>
<td>How did the problems occur?</td>
</tr>
<tr>
<td></td>
<td>How did you deal with them?</td>
</tr>
<tr>
<td></td>
<td>What unexpected things did you face?</td>
</tr>
<tr>
<td></td>
<td>How did you deal with them?</td>
</tr>
<tr>
<td>Visualization</td>
<td>How would you describe your understanding of the working process?</td>
</tr>
<tr>
<td></td>
<td>How would you describe the workflow?</td>
</tr>
<tr>
<td></td>
<td>What do you think about the workflow being on the board?</td>
</tr>
<tr>
<td></td>
<td>Did you watch the board?</td>
</tr>
<tr>
<td></td>
<td>Why did the board change?</td>
</tr>
<tr>
<td>Understanding the whole</td>
<td>How did you try to understand the whole picture, the whole domain?</td>
</tr>
<tr>
<td></td>
<td>What made it easier to understand the whole?</td>
</tr>
<tr>
<td></td>
<td>What made it more difficult?</td>
</tr>
<tr>
<td>Communication</td>
<td>How much opportunities did you have for discussions with others?</td>
</tr>
<tr>
<td></td>
<td>Why was there a lot of communication?</td>
</tr>
<tr>
<td>Embracing the method</td>
<td>How did you learn the development method?</td>
</tr>
<tr>
<td></td>
<td>What have you learnt about it?</td>
</tr>
<tr>
<td></td>
<td>Does the board help you to learn the method? Why?</td>
</tr>
<tr>
<td></td>
<td>How do you think everyone was interested in developing and using the method?</td>
</tr>
<tr>
<td>Feedback</td>
<td>How much support and encouragement did you get? By whom?</td>
</tr>
<tr>
<td></td>
<td>How did the team members support each other?</td>
</tr>
<tr>
<td></td>
<td>Did you get feedback? When?</td>
</tr>
<tr>
<td>Approval process</td>
<td>How did things get approved?</td>
</tr>
<tr>
<td></td>
<td>Who approved taking the next task?</td>
</tr>
<tr>
<td></td>
<td>Who approved that the task is done?</td>
</tr>
<tr>
<td>Selecting work assignments</td>
<td>How would you describe selection of the tasks?</td>
</tr>
<tr>
<td></td>
<td>On what grounds did you select your tasks?</td>
</tr>
</tbody>
</table>

of Kanban to illuminate problems appeared when some tasks were assigned, as a member said:

“One developer had two tickets and was about to take the third, but the limit of the work-in-progress was two so he could not take the new ticket. I understood then the point of limiting the work-in-progress, the blockages must be removed before you can take on a new task.” [D1]

The team tried to solve the problems first within the team:

“Well, we were 13 of us in a room so you needed only to open your mouth and it was likely that someone could assist you.” [D5]

**Visualization.** The Kanban board was very visual and the developers found it very useful, as a member confirmed:

“To me, the most important thing about Kanban was that I could see the board every morning and get an idea of who is doing what and if there is something important waiting to be done.” [D1]

Visualizing also gave a great motivation to the team. The developers could see from the movement of tickets in the verified column of the board what progress had been made.

“It was not so motivating to move those tickets but when the verified column began to pile up, it motivated me.” [D2]

“The board motivated me to work faster. Once, for example, we had a lot of tasks in the verified column and one single ticket on the Kanban board which was my task. So I could look and see... oh, everyone has finished the tasks... and you are in the working column, so go on and do it. So that was a motivating thing.” [D3]

**Understanding the whole.** The team found some tools for understanding the whole according to the corresponding Lean principle; the developers tried to use the competitors’ websites and everyone was present at the customer demonstrations. Using the competitors’ websites helped the developers to bring new ideas to the project. The following statements describe the situation.

“The team had to pick up a website, make an analysis of it. Otherwise people come across a set of ideas, but then again, the ideas are one and the same all the time. Certain things impressed me on those websites. I made a list of them and then I thought, how can I get this into my project. So I had suggestions and after the approval we thought that ok, this can satisfy the customer.” [D3]

“I think the best way was to attend the demo. The customer raised a lot of questions, you knew what they really want, and you got a general, a big idea of the project.” [D6]
Moreover, everyone in the team could take any of the tasks:
“...We did not have each person doing a small piece of the system but anyone could do anything. Everyone saw the whole system all the time.” [D2]

Communication. Communication was informal and free:
“We had a pretty good amount of communication within the team, we felt we were a single entity. The communication was extremely flexible.” [D3]
“I feel that communication was highly agile, free. Everyone could comment on anything they wanted to.” [D4]

In addition, the importance of communication was found in the observation:
“A task had been in the working column for two days. Now the team found a way to slice it into smaller tasks. Note the communication: if your task is too big, someone else can find a way to slice it up.”

Embracing the method. The developers found the Kanban method to be intuitive. After only a short briefing the developers found they understood the basics of how to use Kanban.
“At the beginning, they [more experienced members] explained how to use the Kanban board briefly. It was clear to me and everything could be seen on the Kanban board.” [D6]

At the beginning of the observation, the team leader used the Kanban board to teach the process:
“A developer was doing something else than what was shown on the board. The team leader called the developer over to the board and told him that the board must show what he is actually doing. If something is finished, he must move it to the next column and take a new task. The Kanban board was used for teaching the process.”

Structuring the Kanban board is not prescribed. The board also changed during the case project. The team constructed the columns of the Kanban board at the beginning of the project:
“You must proactively think of how the workflow is going.” [D4]

The columns changed during the project: the todo list was removed from the board during the second week of the project but on the third week most important tasks were added to a right-now column on the board. At the beginning of the project, there were no limits to the work-in-progress but during the second week they were added as found in the observation:
“The limits were added when a developer already had two tickets but was about to take a third nice-to-do [i.e., low-priority level] ticket.”

Feedback. Feedback from the team was constant.
“We had a good atmosphere, everyone was talking to each other and if you had done something well you got a few positive words.” [D2]

“When you had thought about doing something in a certain way you would knock on someone’s shoulder, is this ok, does this look all right.” [D5]

The team itself had defined some possibilities for feedback in their development process: there was a code review and quality assurance for every task. In those phases, a developer found a senior developer to go through his work and give comments.
“Our code review phase was a significant source of feedback.” [D1]

The team met with a customer approximately once a week to get customer feedback.
“The demos gave us feedback from the customer about whether we are going in the right direction but the team would have needed the customer more.” [D1]

The team would have needed feedback from the customer more often:
“The demos were unfortunately the only chances for us to actually talk with the customer.” [D4]
“When we got something done we had to wait for the next demo to get feedback from the customer.” [D4]

Approval process. The approval process was lightweight. The expertise was within the team: there were no higher-level or external authorities to tell the team what they could or should do. As for single tasks, the team itself made rules for approving the task to be ready to move to the next column. A developer had to find another developer to approve his or her tasks.
“When you thought it [task] was ready, you moved it to the code review column and found a reviewer for your task. Then, when he said that it was ok, the task moved to the quality assurance column and you got someone to perform the quality assurance and when he said that it was ok then it was ready.” [D4]

The customer approved the features in the customer demos. If the customer wanted changes to the working feature, the changes were sent to the backlog column.
“Everything that was in the demos was basically a working product. Sometimes, some tickets came to be corrected during the next week.” [D4]

Selecting work assignments. The developers could choose their work tickets independently as long as they followed the priority order.
“We selected the tickets ourselves but we always asked the team if there is something that is more important if the priority is the same. There can be some dependencies that you cannot see, maybe something needs to be done to get something else working. So we asked for a lot of opinions but it was self-organizing.” [D4]
The developers had their own motives for taking the tickets:

“You took the most important ticket that you were interested in.” [D1]

“You could take an easy ticket to get into the system and then some more different stuff so you learn all the time.” [D2]

“Something was related to my previous tasks so I took it” [D6]

“If I picked up a task, how could it help another individual so that it could reduce the massive coding part or something.” [D3]

V. DISCUSSION

A. Findings

Table III analyzes the case study findings presented in Section IV with respect to the expected influences of the Kanban process model as hypothesized in the research framework (Table I). Since the research data collection is qualitative, the evaluation scale is coarse.

Poppendieck and Poppendieck [8] state that Kanban documentation should be used only if a customer needs or wants it, which was also supported in our case study: the customer did not express a need for any particular design or user documents so the team did not produce them.

Instead of hiding the problems, the team brought them to the surface immediately, which enabled solving them quickly. The literature recommends this kind of an approach [8], [10]. Related to the Lean philosophy, Liker [6] suggests that problems should always be solved to the root cause by asking ‘why’ five times. The root cause is then eliminated in order to prevent problems in the future. His proposition was, however, not supported in our empirical study: problems were disposed of to the extent necessary for the project but not all the way to the root cause. One may ask whether it is necessary in the field of software development to always find the root cause of a problem.

The visualization of Kanban was supported in our case study. The Kanban board helped the developers to become aware of the state of the product, especially the existence of problems. In addition, visualization helped the developers to understand their work process. Moreover, we found that the communicativeness of the Kanban board motivated the developers and helped them in picking up new tasks.

As proposed in the literature [8], a constant drive to see the whole was also present in our study. The literature, however, does not specify guidelines of how to actually make this happen. We found that the developers attained seeing the whole by being present at the customer demos, by being able to carry out any task, and by exploring the market environment by investigating the competitors’ websites.

Communication was free in the project as proposed in the literature [8]. This led to a good atmosphere and made the team feel as a single unit.

Related to embracing the method, the developers found Kanban to be a very intuitive method like advocated in the literature [8]. However, adjusting the actual Kanban board correctly for the project took time. We found that at the beginning of the project it was well worth the time to address the structure of the board properly.

Feedback from the team members was constant and abundant. However, the developers found feedback from the customer to be too sparse and the Kanban method did not offer any guidelines for it. As Korkala and Abrahamsson [41] state, the lack of an onsite customer is a problem in most software development projects. According to our findings, Kanban did not offer a solution either.

The approval process was lightweight as proposed [8]. Here again, the developers would have needed comments from the customer more often.

Related to selecting work assignments, the developers could freely pick up their tasks as long as they followed the priority order, which supports the proposition of Liker [6] that the best expertise is at each workstation. However, Kanban did not offer tools for prioritization of the tasks.

Based on this case study evidence, we can infer that the most significant influences stem from the inherent visualization of Kanban. That helps in controlling the project activities in flexible yet coherent ways by relying on the intuition of the team members and emergence. Another supporting trait is that the simplicity of the Kanban model allows situational adaptation.

However, we can also see that Kanban is not all-encompassing, and it is not sufficient for managing all the dimensions of software projects. For instance, while the Kanban board helps to detect potential problems and bottlenecks early, it requires additional practices to actually solve them. Moreover, “seeing the whole” may still be difficult in particular with larger, complex system projects with a simple Kanban board alone. That is, Kanban needs supportive practices and contextual linking, for example, to incorporate customer feedback.

Empirical evidence clearly hinted that the adoption of Kanban in use was quite straightforward even if continuous modifications and adjustments to the actual use of the board were done. While Kanban offers means for coordination purposes, the inherent simplicity of the approach supports the perception of creative freedom necessary in software development. The need for such freedom is often undermined or overlooked by project management tool providers.

B. Validity of the Study

Our model of the work of a software developer is not necessarily sufficient to cover all aspects of work; there are probably other perspectives than what we selected for our model. The model covers only part of the work a software developer does. In our model we describe nine aspects based on plenty of descriptions in the literature.

Even though our results may not be directly generalizable, they may give hints on how Kanban can be used in real-life software projects. Because of the team of thirteen developers, the project was of reasonable size and we were able to collect...
we then tested this model with the direct observation, video not been studied much. In this paper, we proposed a research process model on the work of a software project itself have software development processes. Moreover, the impacts of the software process method. Although Kanban is well known in programmers in the industry can be considered equal. This kind of sample population used in the study and senior Arisholm and Sjøberg [46] argue that programming skills of not have much experience in developing software systems.

Kanban board evolved during the project. However, Kanban as a method is flexible and does not require strict ways to work. Moreover, three of the developers we interviewed did not have much experience in developing software systems. Their opinions of the impacts of work were based only on this project. However, we used, in addition to interviews, the direct and video observation. One of the interviewees had many years of experience and had used both the waterfall model and agile process models several times.

Our framework model (Table I) of the work of a software project is not necessarily sufficient to cover all aspects of work; there are probably other perspectives than what we selected for the model. This model covers only part of the work a software developer does. Nevertheless, the model includes nine aspects described widely in the literature. There was no ready-made model to test empirically. However, a lot of research about software process models existed so we were able to build our own model about the work of a software developer based on the established groundings.

Finally, the approach of the experimentation strategy used in this study has been favored [42], [43]. The use of the sample population (i.e., the team members of the project) in this study has been shown as a valid approach in the literature, for example by Höst et al. [44] and Madeyski [45]. In addition, Arisholm and Sjoberg [46] argue that programming skills of this kind of sample population used in the study and senior programmers in the industry can be considered equal.

VI. Conclusions

There is not much empirical research about Kanban as a software process method. Although Kanban is well known in production fields, it has not yet been widely investigated in software development processes. Moreover, the impacts of the process model on the work of a software project itself have not been studied much. In this paper, we proposed a research framework for those purposes. In the empirical case study, we then tested this model with the direct observation, video observation and thematic interviews.

The investigation indicated considerable benefits of the Kanban process model, which, on its part, benefits software project management. This is an advantage for practitioners: by being aware of the nine aspects of work suggested in the study, management can use them as lenses to concretize the basically abstract environment of the software development process. It is important to take care of each work aspect supporting the project goals (i.e., adding value directly or indirectly to the customer) and being utilized by the efficient use of Kanban. These should provide more benefits and less waste in order to make projects more successful. Regardless, being a relatively basic control tool, Kanban needs to be supported with additional practices and connections.

Following that line of reasoning, this study leads to some further research work: refining the research instrument (tables I and II), conducting more empirical investigations also in industrial settings, and elaborating the research model to capture quantitative performance effects of the Kanban development.

More generally, there could also be some potential for more cross-disciplinary learning since the Kanban model has been used for years in manufacturing environments. However, the knowledge-intensive nature of software development work must be taken into account. Altogether, the more Kanban is being studied in the field of software development, the more evidence is found of its adaptability into software engineering projects. Comparisons of Kanban made, including this paper, with conventional and agile software process methods emphasize the potential that Kanban provides for the software field. Impacts of Kanban on software development work is an area of research wherein the potential still lies without having utilized it widely.

ACKNOWLEDGEMENTS

(will be written after the review process)

REFERENCES


<table>
<thead>
<tr>
<th>Aspect of Work</th>
<th>Expected Influences of Kanban (Table I)</th>
<th>Evidence (Case Study)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documentation</td>
<td>Only if the customer needs and favors it</td>
<td>supported</td>
</tr>
<tr>
<td>Problem solving</td>
<td>Problems are found quickly on the Kanban board; they are thus tackled immediately</td>
<td>somewhat supported</td>
</tr>
<tr>
<td>Visualization</td>
<td>The work is visualized on the Kanban board</td>
<td>supported</td>
</tr>
<tr>
<td>Understanding the whole</td>
<td>One of the Lean key principles; Kanban does not offer tools though</td>
<td>somewhat supported</td>
</tr>
<tr>
<td>Communication</td>
<td>Rapid and plenty</td>
<td>supported</td>
</tr>
<tr>
<td>Embracing the method</td>
<td>An intuitive method (e.g., visualization)</td>
<td>supported</td>
</tr>
<tr>
<td>Feedback</td>
<td>Rapid and plenty; supports also regular meetings with the customer</td>
<td>not supported</td>
</tr>
<tr>
<td>Approval process</td>
<td>Best expertise is at each workstation or developer; no complex approval processes</td>
<td>somewhat supported</td>
</tr>
<tr>
<td>Selecting work assignments</td>
<td>Developers select and pull their work voluntarily and individually</td>
<td>supported</td>
</tr>
</tbody>
</table>

TABLE III
EMPIRICAL EVALUATION OF THE KANBAN-RELATED IMPACTS WITH RESPECT TO THE RESEARCH FRAMEWORK.