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

Marko Ikonen¹ Elena Pirinen² Fabian Fagerholm¹ Petri Kettunen¹ Pekka Abrahamsson¹
¹Department of Computer Science
University of Helsinki, Finland
firstname.lastname@cs.helsinki.fi
²Project Management Office
Aalto University, Finland
firstname.lastname@aalto.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 projects. 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. The impact of Kanban is evaluated from nine theoretically derived perspectives. The results highlight new findings regarding the application of Kanban in the software context. This bears managerial implications, which is addressed. The key implications of the findings suggest that Kanban and its inherent simplicity motivate the workers and control the project activities.

Keywords—software development, Kanban, project management, process model

I. INTRODUCTION

The principles of Lean thinking are grounded the ways 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. Eliminating waste is the most important principle of Lean thinking. Lean thinking requires continuous growth and commitment from personnel as well as management. This is similar to the concept of learning organization [2], [3], [4] where the organization continuously develops itself by facilitating learning among its members.

While Lean thinking has turned out to be a success in manufacturing [1], it has recently been applied to the area of software engineering, as well [5]. The basis of the repeatable manufacturing process, however, differs from the idea of traditional software projects. Project Management Institute in its PMBOK Guide [6] defines project as “a temporary endeavor undertaken to create a unique product, service, or result”. Despite this difference, Lean thinking can be applied, among other areas, to software development projects [7], [8], [9]. In addition to waste elimination, Lean software development principles can be defined as follows: build quality in, create knowledge, defer commitment, deliver fast, respect people, and optimize the whole [10]. Moreover, the creativeness and innovativeness of the workers should be utilized [8]. In a wider context, the production should be conceptualized as a flow [11]. The Kanban method is often presumed to make this attainable [5], [12].

There is a strong practitioner-driven movement supporting the idea of the use of Kanban in software engineering. It is evident that the outcomes of applying Kanban are expected to be high in software development, as they have been in manufacturing. This is not yet, however, confirmed by means of empirical studies: only a few studies have explored empirically the dynamics of Kanban from the viewpoint of software development. This lack serves as an important source of motivation for this study.

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” [13]. Howell et al. [13] 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 the 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 literature-based framework modeling effects of Kanban on software project work. This framework recognizes nine perspectives which are then evaluated by conducting a case study in an experimental software R&D setting that produced a business prototype on a proof-of-concept level. The results from the empirical evaluation support some arguments of the literature. In addition, some new findings, such as task assigning depending on the different motives of workers, are encountered.

II. RELATED WORK

This section reviews the background and related work of project management (Section II-A) and Kanban software development (Section II-B). Moreover, Section II-C develops a research framework.

A. Software Project Management Challenges

Since software processes are largely dynamic and cooperative activities, the impact of software process technology with...
its modeling languages, editors, and interpreters has only been small [14]. In the 1980s, crucial support processes, such as learning, technical communication, requirements negotiation, and customer interaction, were poorly described in software process models [15]. The reasons for most project failures were seen in the managerial, not in the technical sector [16], [17] and can often be traced to poor team performance. This is caused by inadequate attention to people and teamwork issues [18] since people have a tendency to concentrate on the technical aspects (hardware and software) rather than the peopleware and impermanence [19].

Boddy and Macbeth [20] 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 [21]. Regarding the group formation, the key elements are team-based learning, group behavior, and ways in which people create purposes and communicate [22]. In software development, Moe et al. [23] address five dimensions in order to improve the teamwork: shared leadership, team orientation, redundancy, learning, and autonomy. The most salient problems reported in projects concerning additional efforts or mistakes include communication and coordination breakdowns [15], [18], [24]. Teasley et al. [25] 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 [26] 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 [16] 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 smooth task flow and short feedback loops (ensuring that tasks are kept small), the schedule and budget can be specified based on facts, not on guesses. Regarding the scope risk, Kanban has been designed for continuous requirement changes. At the beginning of a project, determining all scope risk, Kanban has been designed for continuous requirement changes. At the beginning of a project, determining all assignments can be reorganized faster, which increases control with 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 [21]. Regarding the group formation, the key elements are team-based learning, group behavior, and ways in which people create purposes and communicate [22]. In software development, Moe et al. [23] address five dimensions in order to improve the teamwork: shared leadership, team orientation, redundancy, learning, and autonomy. The most salient problems reported in projects concerning additional efforts or mistakes include communication and coordination breakdowns [15], [18], [24]. Teasley et al. [25] 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 [26] 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 [16] ranked these two factors among the top six of the software risk factors.

Boddy and Macbeth [20] 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 [21]. Regarding the group formation, the key elements are team-based learning, group behavior, and ways in which people create purposes and communicate [22]. In software development, Moe et al. [23] address five dimensions in order to improve the teamwork: shared leadership, team orientation, redundancy, learning, and autonomy. The most salient problems reported in projects concerning additional efforts or mistakes include communication and coordination breakdowns [15], [18], [24]. Teasley et al. [25] 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 [26] 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 [16] 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 smooth task flow and short feedback loops (ensuring that tasks are kept small), the schedule and budget can be specified based on facts, not on guesses. Regarding the scope risk, Kanban has been designed for continuous requirement changes. At the beginning of a project, determining all assignments can be reorganized faster, which increases control with 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 [21]. Regarding the group formation, the key elements are team-based learning, group behavior, and ways in which people create purposes and communicate [22]. In software development, Moe et al. [23] address five dimensions in order to improve the teamwork: shared leadership, team orientation, redundancy, learning, and autonomy. The most salient problems reported in projects concerning additional efforts or mistakes include communication and coordination breakdowns [15], [18], [24]. Teasley et al. [25] 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 [26] 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 [16] 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 smooth task flow and short feedback loops (ensuring that tasks are kept small), the schedule and budget can be specified based on facts, not on guesses. Regarding the scope risk, Kanban has been designed for continuous requirement changes. At the beginning of a project, determining all assignments can be reorganized faster, which increases control with 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 [21]. Regarding the group formation, the key elements are team-based learning, group behavior, and ways in which people create purposes and communicate [22]. In software development, Moe et al. [23] address five dimensions in order to improve the teamwork: shared leadership, team orientation, redundancy, learning, and autonomy. The most salient problems reported in projects concerning additional efforts or mistakes include communication and coordination breakdowns [15], [18], [24]. Teasley et al. [25] 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 [26] 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 [16] 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 smooth task flow and short feedback loops (ensuring that tasks are kept small), the schedule and budget can be specified based on facts, not on guesses. Regarding the scope risk, Kanban has been designed for continuous requirement changes. At the beginning of a project, determining all assignments can be reorganized faster, which increases control with 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 [21]. Regarding the group formation, the key elements are team-based learning, group behavior, and ways in which people create purposes and communicate [22]. In software development, Moe et al. [23] address five dimensions in order to improve the teamwork: shared leadership, team orientation, redundancy, learning, and autonomy. The most salient problems reported in projects concerning additional efforts or mistakes include communication and coordination breakdowns [15], [18], [24]. Teasley et al. [25] 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 [26] 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 [16] ranked these two factors among the top six of the software risk factors.

B. Kanban Software Development

The Japanese word kanban refers to a signboard. When the term is used in manufacturing, it means a scheduling system that hints what, when, and how much to produce [27].

Fig. 1. An example of one implementation of the Kanban board used in a software development project.

It is claimed that Kanban is one of the important elements that executes Lean thinking in practice [28], [29]. Kanban is also one of the key operation management tools in Lean manufacturing [8]. 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) [30].

Despite the demand to visualize the workflow, there are no particular rules concerning how to implement the content of the Kanban board. In their basic form in production environments, Kanban controls are typically implemented with physical index cards (usually called tickets) moving along with the material. The cards then act as the flow-control tickets between the different work stations or processes. Figure 1 illustrates one realization of Kanban as a table (such as a wallpaper with sticky notes). Each project task (card) flows from one state to another (from left to right) as it progresses. Hence, the overall project situation can be seen at a glance while the dynamic moving of the task cards indicates the project progress (or blocking) over time.

Benefits of Kanban scheduling are reduced inventory (simultaneous WIP), improved flow, prevented overproduction, operations-level control, visualized schedule and management of the process, improved responsiveness to changes in demand, minimized risks of inventory obsolescence, and increased ability to manage the supply change [31]. As a result, Kanban attempts to lower production costs, increase quality, and accelerate cycle time [8]. Meanwhile, inventories and problems caused by sudden changes will become more apparent.

However, while Kanban attempts to clarify workers’ awareness of the current production issues and forthcoming tasks, it does not recommend any particular project phases, milestones, or partitioning tasks. Due to this liberty, it is up to a project team to build and customize the appropriate practices for its project. Some such practices have been suggested in the literature. Ladas [12], for example, suggests that the amount of tasks in progress simultaneously should be adjusted to the reasonable capacity in use. Middleton [9] claims that this amount should be minimized in order to keep quality high. Shinkle [5], 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 to do, design, and coding, gets its own “ready” stage, the blockages
in the workflow become more visible [12]. Alwardt et al. [32] state that tasks should be prioritized. Moreover, laborious tasks should be partitioned before setting them as assigned [5].

While the operations management field has investigated Kanban-based production for years, not much empirical evidence has been published in software development. It follows that the presumed advantages and suggested control rules, such as WIP sizes, remain largely anecdotal and not necessarily generally applicable.

Figure 1 demonstrates the meaning of WIPs. In the “Code Review” column, the WIP number has been set at two. This limit means that no more than two tickets are allowed to be in the column simultaneously. If a task being code reviewed were problematic, carrying it out without the assistance of others would restrain the flow. Other people, once they have finished their tasks, cannot put another ticket into the column because it is already full (due to the WIP limit). They rather have to help with the problematic ticket in order to free some space in the column for new tickets. In this way, the flow is supposed to be smooth and bottlenecks avoidable.

C. Research Framework

In order to analyze the effects of Kanban upon software project management, we develop a research framework. This framework (Table I) comprises nine literature-based aspects of project work. It is expected, based on the literature, that the use of the Kanban process model influences all aspects of the project work. Next, the influence of the nine aspects is reflected on the Kanban process model. In order to ground them, the aspects are additionally contrasted with the waterfall model baseline [33]. This is because of its historically strong and long-term impact on software development projects. In addition to the waterfall model, the values of agile approach (see Agile Manifesto [34]) is taken into account. The reason for this additional viewpoint is that agile methods [35] have been considered to provide a solution for many problems derived from the plan-driven, conventional software development method, such as the waterfall model.

The first aspect of work is the amount and type of documentation. Software domain and criticality have a great influence on what needs to be documented in software development. The waterfall model typically implies a lot of documentation [36] since every phase produces documents for the next phase. Agile Manifesto [34], instead, prefers “working software over comprehensive documentation”. This implies in practice that the documentation needs are identified in close collaboration with the customer. In Kanban, the customer pulls results from the software developers rather than the developers pushing results to the customer [36]. Therefore, it could be expected that documents are not produced in Kanban without the need. Yet, good practice in software development expects at least the documentation of critical system design aspects alongside the architectural issues as well as the following of coding conventions including code commenting guidelines.

Regarding problem solving, Sommerville [37] suggests that the waterfall model may hide problems. Problems surface late in the development process and they are dealt with by a workaround or even ignored. In contrast, agile principles [34] refer to delivering working software and to “welcome changing requirements, even late in development”. Lean thinking is even more concrete since problems are required to be solved immediately and completely in order to prevent the same kinds of problems in the future [8]. Using Kanban, problems can be found almost immediately after their occurrence [12].

Visualizing the development process helps the developers to see the state of the development process: how much work has already been done and how much is yet to be done [36]. The waterfall model is visual because after each phase a tangible document is made which shows the amount of work [37]. The agile principles do not emphasize visualization. Even though the agile Scrum process model [38], e.g., asks to monitor and update the progress, its visualization focuses on the time-limited sprints. The Kanban method, instead, visualizes all work flowing through work stages on the Kanban board [36]. By looking at this board, everyone can see how a single task as well as the whole project as a whole is progressing.

The waterfall model supports understanding the whole since the system is known after the specification phase [33]. Again, the agile principles explicitly ignore this despite that customer satisfaction, one key of Agile Manifesto [34], is a part of seeing the whole [39]. On the other hand, the Lean principle of “optimize (or see) the whole” has been noticed in Kanban since 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 [36].

The waterfall model does not encourage informal communication [36]. Each phase, instead, produces explicit documentation which the developers follow in the next phase. Agile Manifesto [34], instead, emphasizes communication. In Kanban, communication is important and it should be free and open, which follows Lean thinking [40].

Poppendieck and Poppendieck [36] suggest that if a process model being used 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 [41]. 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 [41], [42]. Larman [43] 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, typically need some earlier experience from the developers, before reaching an efficient way of carrying out the progress. Larman [43], for example, mentions common mistakes and misunderstandings related to the lack of experience. The Kanban method is intuitive to understand and gives rather free hands to the developers to do their work [30]: the only rules regarding this are that the workflow must be visualized, cycle time measured, and work-in-progress limitations enforced.

In the waterfall model, giving feedback may be limited except between the phases [37]. Feedback is considered to be a threat to the predefined plan because corrective actions would change the plan. Agile Manifesto [34], as stated above, welcomes 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 [36].

The approval process is often overly strenuous when using the waterfall model [36]. After each phase the documentation is approved by management. Agile Manifesto [34] 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 [36]. 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 should be selected. Both agile and Lean thinking prioritize the satisfaction of the customer through a valuable product. In the Kanban method, developers can select their own work according to Lean thinking [36]. There, the work is self-organizing.

III. Empirical Research Design

This section describes the research setting for the case study (Section III-A) and the research methods used (III-B).

A. Research Environment and Case Project

Software Factory is a novel software engineering research and education setting at the University of Helsinki [44]. It is basically an industry-oriented R&D laboratory environment for conducting software business projects. It also has an educational purpose and provides computer science students with an opportunity to put software engineering and project management theory into practice. 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 a Smartboard and configurable visualization screens. 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 [45]. The Department of Computer Science hosts an initial reference implementation, with more locations coming up globally.

The case project produced a business prototype on a proof-of-concept level. This product was a Web application for an imaginary market. Most of the team members had work experience in programming and project work from a couple of months to a couple of years. The project contained 13 persons: 12 Master students including the team leader and one Bachelor student. Excluding the use of Kanban, no 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. One of the authors of the paper acted as one of the customers of the project but without steering the use of Kanban in the project while another furnished the other customers, launched, and finished the project.

The team formed the columns (i.e., workflow phases for the tasks) on the Kanban board at the beginning of the project. Later on, the team altered the columns and set up WIP limits in order to get a smoother workflow. The tasks were typically chunked into ca. half-day size, i.e., three or four hours.

During the 7-week project, the team had one-week iterations with weekly customer demos and retrospectives. Such a sprint-based, iterative Kanban is called Scrumban [12] and it provides a structured schedule also for the customer. Daily stand-up meetings (similar to the short (max. 15 minutes) project status meetings in Scrum) were held, as well. The customers were experienced technically and in the customer role. They were also obligated to commit themselves with time and interaction and they were reachable also outside the demos so the team could check and ask about things. IT support and an external technical consultant were available throughout the project. Excluding the author operations mentioned in this section, the team was on their own.

B. Research Methods

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 both video and direct observation [46], and thematic interviews [47]. The direct observation that was performed by being physically present in the Factory but without participating in the project work, lasted eight days. The video observation, in its turn, was focused on a particular session involving the development team and the customer representative working together including the Kanban board.

After the project, 8 software developers including the team leader 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 five 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. The customer interview
data gathered in the concurrent study [48] was used to support
the team members' side of the story. The answers related to
customer demonstrations, time between demonstrations, and
communication with the team.

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 inter-
view recordings. For identification the developers have been
numbered from D1 (developer #1) to D8 (developer #8). D1
was the team leader. The customer representatives have been
marked C.

Documentation for the customer was minimal; the cus-
tomer was not interested in documents so the team only
did the documents they needed for their own work, such as
instructions for the developers to install the local software
environment. A member of the project stated:

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

Documentation in Kanban should be used only for produc-
ning value. This need for documentation was not noticed in
customer demonstrations either. Nevertheless, our observation
revealed that the team wrote necessary documents for internal
use but did not waste its time in preparing, e.g., massive
requirement documents.

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 since this task had a high priority and delaying
the tests would have restrained the ticket flow on the Kanban
board. Comprehensive integration testing that was performed
during the last week of the project detected only a few defects,
as a member stated:

“We made quite comprehensive integration testing
during the last week. Hardly any traditional bugs
were detected. The coverage of the [code] lines was
about 80%. Rather, we detected more bugs in the
interface.” [D1]

This low detection rate follows the ideal situation in Kanban:
since the defects are solved right after their occurrence, they
cannot accumulate at the end.

If a developer had a problem with his or her tasks, it
had to be solved before taking on a new task. Therefore,
problems appeared as blockages in the workflow. The ability
of Kanban to illuminate problems appeared when some tasks
were assigned, as a member said:

“One developer was about to take a new ticket,
but the work-in-progress limit did not allow this.
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]

Instead of hiding the problems, the team brought them to
the surface immediately, which enabled solving them quickly.
Related to Lean thinking, problems should always be solved
at the root cause. The root cause is then eliminated in order
to prevent problems in the future. The team seemed to follow
this suggestion since the comprehensive integration testing
did not detect many defects from the code (but rather from
the interface that is harder to test automatically). While we
cannot connect Kanban to this root-cause elimination, Kanban
definitely drove the team to solve problems immediately in
order to prevent blockages on the Kanban board.

Visualization. The Kanban board was visual and the de-
velopers found it useful and informative, as members confirmed:

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

TABLE II
THE OPEN-ENDED QUESTIONS USED IN THE THEME INTERVIEW.
“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... If there were many tickets, for example, in the Quality Assurance column waiting for processing, why don’t we start working on them so they could flow forward?” [D1]

“The board functioned very well. If the information of the Kanban board is only in a computer system, it does not communicate so well.” [D4]

Visualizing also motivated the team greatly. 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 in the former columns 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.” [D4]

The visualization also helped the team to select features for the customer demos, as a member stated:

“Selecting things for the demos was easy. You just picked up the appropriate ones from the right side of the Kanban board [i.e., the Verified column into which the tickets were accumulating].” [D1]

Altogether, the visualization of Kanban was supported in the 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, motivated them, and helped them in preparing the customer demos.

**Understanding the whole.** The team found some ways 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 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, 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, which eased seeing the whole:

“We did not have each person doing a small piece of the system but anyone might do anything. Everyone saw the whole system all the time.” [D2]

According to the evidence, 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. In the case study, the developers attained seeing the whole by being present at the customer demos, by being able to carry out a variety of tasks, and by exploring the market environment by investigating the competitors’ websites. Nevertheless, the study did not show whether this attempt to see the whole relates to Kanban.

**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. As I said, the Kanban board functioned very well in communication. An [electrical] information system may be hidden because it locates somewhere and then it does not communicate with anyone.” [D4]

Such an atmosphere is a signal of trust in the team. In these positive circumstances, visualized problems on the Kanban board launched some discussion inside the team as we found in the observation: A task had been in the Working column for two days. Then 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.

Visualizing problems does not solve them without the team’s communication ability so the team used communication in order to find ways for improvements in the flow. Seeing a problem typically encourages motivated people to solve it on their path to good results, as occurred in the case project. On the other hand, following the idea of communication and rapid feedback, as the team did, prevented errors and mistakes from accumulating and growing to be more serious.

**Embracing the method.** The Kanban method was found 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 it was found that 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 then 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.

Structuring the Kanban board is not prescribed in the literature (Section II-B). This encourages thinking:

“You must proactively think of how the workflow is going.” [D4]

The columns changed during the project: the To Do list was removed from the board during the second week of the project.
but on the third week most important tasks were added to the Right-now column. At the beginning, the team did not limit the work-in-progress but during the second week the WIPs were added. A member confirmed our observation that the method and altering the Kanban board was felt to be useful:

“The Kanban method fitted into this project. The columns were altered appropriately during the project.” [D8]

Overall, the developers found Kanban to be a very intuitive method as advocated in the literature. However, adjusting the actual Kanban board correctly for the project took time. As an advantage, the team was itself forced to think about how to best adjust Kanban for the project. We found that at the beginning it was well worth the time to address the structure of the board properly: adding the Code Review column (see “Feedback” below), e.g., increased feedback and setting the WIPs prevented task switching and made the flow smoother.

Feedback from the team was constant and abundant. The team itself had defined some possibilities for feedback in their development process; there was Code Review and Quality Assurance columns for every task. In those phases, a junior found a senior to go through his or her work and give comments.

“Our code review phase was a significant source of feedback.” [D1]

Kanban encourages keeping plausible iterations short so the feedback is expected to be instant. The team met with a customer approximately once a week to get customer feedback but the feedback from the customer would have been needed more often:

“The demos gave us feedback from the customer about whether we are going in the right direction but we would have needed the customer more.” [D1]

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

During the seventh week, the progress was accelerated since the customer was present daily. The customer stated:

“When in the last week I was present all the time and visited there many times a day, many things that could have waited a few days more for the traditional customer demo, were solved there.” [C]

This indicates that communication with the customer and getting more feedback from the customer were not up to Kanban but the team. After all, the customer agreed about the importance of the demonstrations:

“We had these customer demos that became critical communication points... What is the direction and something needs to be done to get something else working. So we asked for a lot of opinions but it was self-organizing.” [D4]

However, the Kanban method did not offer any guidelines for feedback according to our findings. On the other hand, Kanban did encourage the team to chop tasks into small pieces, which enabled short feedback cycles. As it turned out, the code review phase was a significant source of feedback. As stated above, the atmosphere was positive and Kanban launched some communication. This contained feedback, as well.

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 (in the sense described in Section III-A). 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 [the 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]

This statement emphasizes how Kanban enables keeping the flow smooth: First, the work phases for the tickets are clear and logical. Second, if all other members are busy, the tickets cannot pass the phase of code review or quality assurance. If the tickets continue accumulating in those columns, e.g., it forces the team to smooth the flow.

The customer liked the approach, wherein the features were asked to be approved in the customer demonstrations:

“The leader asked what they should concentrate on during the next week... I felt it was actually a good thing to do. It makes you summarize the themes into a couple of sentences.” [C]

If the customer wanted changes to the working feature, the changes were sent to the To Do column.

“All that was in the demos was basically a working product. Sometimes, some tickets came to be corrected during the next week.” [D4]

Since the best expertise lay within the current reviewers and implementers of the tasks, getting approvals from management was not needed. Kanban supported also, at least to some extent, the final approval process made by the customer since approvals were made in the demonstrations weekly.

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 due to Kanban’s liberty:

“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]

Overall, while Kanban did not offer tools for prioritization of the tasks, the interview answers revealed that selecting work assignments can be done differently depending on the motives of the workers. This is an example of how motivation can be supported by Kanban which allows carrying tasks out freely.

V. Discussion

Table III summarizes 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 summary table categorizes the evaluation by coarse-level ordinal factors.

A. Implications

The case study evidence (Section IV), summarized in Table III, supports the research framework (Table I) quite well. In total, seven of the nine team-related aspects were supported. This raises the question why Kanban was able to influence these aspects in software development.

One explanation is the team’s autonomy. Self-organizing teams have been reported to bring advantages to organizations [49], [50], [51]. Positive outcomes are often related to performance effectiveness and members’ attitudes and behavior [52]. The decision-making in the problem level (not, e.g., in the upper management level of the organization) enables the quick reaction ability of self-organizing teams [53]. Instead of waiting for a manager’s approval, the team has the authority to take necessary actions by itself [54].

Another is that Kanban seems to support four practices explaining successful implementation of projects. These are goals, resources, structures, and controls [20] that guide the focus on (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 [21]. Kanban encourages people to use each of these except obtaining resources that does not relate to daily team practices.

Finally, Kanban takes into account the dynamic nature of software development including unrealistic schedules and continuous requirement changes. Through decades, these two factors have been ranked up to the most frequent factors explaining why software projects fail [16], [26].

The Kanban-related literature recommends an approach wherein the problems are surfaced immediately in order to solve them quickly [12], [36]. Self-organizing teams have such a quick reaction ability [53]. Regarding Lean thinking, Liker [8] suggests that problems should always be solved at the root cause. The root cause is then eliminated in order to prevent problems in the future.

Our study supports the literature that assumes Kanban to be an intuitive method. Based on the empirical evidence, we can additionally infer that the most significant influences stem from the inherent visualization of Kanban. Besides, in order to function well, Kanban, in contrast to Scrum, requires visualizing the progress. This visualization was found to motivate people and to help in controlling the project activities in flexible yet coherent ways by relying on the intuition of the team members and emergence. Work assignments, e.g., were found to be appropriate to select differently depending on the motives of the workers. Apparently, such a motive-based assigning has not been discussed in the literature. It is also argued that motivation improves the results [55]. While Kanban is not seen as a people-oriented approach, the elements of empowerment, trust, self-organization, and setting (including Kanban) were found to function as motivators for the case team. Whether this attributes to Kanban can be questioned.

Another supporting trait is that the non-prescribed structure of the Kanban board encouraged the team members proactively to think of what kind of workflow it should be. The simplicity of the Kanban model allows situational adaptation, which is crucial in today’s dynamic environment of continuously and rapidly changing software development.

However, we can also see that Kanban 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. Middleton and Joyce [56] report supportive information radiators, such as ideation pipeline and team performance indicators.

B. Validity of the Study

Our 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. It covers only part of the work a software developer does by describing nine aspects based on 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. The project was of reasonable size and we were able to collect a lot of material to conduct our study. Altogether, the results of the study provide a base for further studies and applications in industrial settings.

The Kanban method was new to every member of the case study project. Therefore the methods of work as well as the 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, in addition to interviews, we also used the direct and video observation that were compared with the interviews to confirm our understanding of the project. Besides, one of the interviewees had over two years of experience and had used both the waterfall model and agile process models several times.
We did not find any ready-made model from the literature to test empirically. Regardless, the existing research about software process models enabled us to build our own model (Table I) about the work of a software project based on the established groundings. Now, when the laboratory results are available, further research can compare the obtained results with similar projects where other methods are used.

Regarding the use of students as study objects, it is quite well established that when one seeks to establish a trend, their use is quite acceptable. Tichy [57] uses a method comparison as a specific target of study where the use of students is a valid approach. Others have made similar suggestions. Höst et al. [58], for example, conclude that students are indeed relevant when considering experimentation in software engineering. Kitchenham et al. [59] remind us not to look down on studies focusing on gathering empirical data from student experimentation or projects. Madeyski [60] also argues strongly for the benefit of using students as study objects. That specific study focused on test-driven development and pair programming. In addition, Arisholm and Sjoberg [61] argue that the programming skills of Master students 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 in practice.

The investigation indicated considerable benefits of the Kanban process model including motivation and control over the project activities. These, on their parts, benefit 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. For industry, the great value of Kanban lies in its real-time visualization. However, it is important to remember that visualization alone does not replace concrete actions or guarantee success. Nevertheless, being a relatively basic control tool, Kanban needs to be supported with additional practices and connections.

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 Kanban has been used for decades in the manufacturing environment. Despite this, the knowledge-intensive nature of software development work must now be taken into account.

Altogether, the more Kanban is being studied in the field of software development, the more evidence seems to be found of its adaptability into software engineering projects. Impacts of Kanban on software development work is an area of research wherein the potential still lies without having utilized it widely.

ACKNOWLEDGMENTS

This work was supported by TEKES as a part of the Cloud Software program of Tivit (Finnish Strategic Centre for Science, Technology and Innovation in the field of ICT). The authors would also like to thank the case project team members for their cooperation, Henri Karhatsu for his contribution, reviewers for their comments, and the staff of the computing facilities at the Department of Computer Science, University of Helsinki and Helsinki Institute of Information Technology, who operate the Software Factory technical environment.

REFERENCES
