Empirical Investigation of Scrumban in Global Software Development

Ahmad Banijamali(✉), Research Dawadi, Muhammad Ovais Ahmad, Jouni Similä, Markku Oivo, and Kari Liukkunen

M3S Research Unit, Faculty of Information Technology and Electrical Engineering, University of Oulu, PO box 4500, 90014 Oulu, Finland
{ahmad.banijamali,muhammad.ahmad,jouni.simila,markku.oivo, kari.liukkunen}@oulu.fi, research.dawadi@student.oulu.fi

Abstract. Scrumban combines two Agile approaches (Scrum and Kanban) to create a management framework for improving software engineering practices. Scrumban is expected to override both Scrum and Kanban, as it inherits the best features of both. However, there is little understanding of the possible impact of Scrumban on software development in prior studies. This study first makes a comparison among Scrum, Kanban, and Scrumban and then investigates the impact of Scrumban on six major challenges of global software development. This study was conducted in a distributed project at two Software Factories in two universities in Finland and Italy. The results show that Scrumban could positively affect issues such as evenness of different sites, communication, and cultural issues as well as leveraging resources among sites. However, there are still few challenges that require alternative methodologies and tools other than Scrumban to be overcome.

Keywords: Agile · Distributed software development · Kanban · Scrum · Scrumban · Software factory

1 Introduction

The success of software development projects depends heavily on the use of appropriate software development methodology. According to a report by Standish Group, 42% of project cases which have used an Agile approach were successful, which is considerably more than what has been achieved using traditional project management methods [1]. Agile methods are iterative, incremental, and enhance collaboration between self-organizing cross-functional teams [2]. Scrum is the most frequently used Agile method in software development [3]. Scrum reaches its objective through time-boxed iterations based on continuous feedback and task prioritization [4].

Kanban, on the other hand, has not been widely adopted in software development [5]. In 2004, Kanban entered into the Agile realm when David Anderson introduced it in practice while assisting a software development team at Microsoft [6].

S. Hammoudi et al. (Eds): MODELSWARD 2016, CCIS 692, pp. 229–248, 2017.
DOI: 10.1007/978-3-319-66302-9_12

Kanban proposes to defer the project commitments, set constraints on the amount of work in progress (WIP), and limit the project promises that cause project failure [7]. The high expectations for Kanban are the result of its adaptability regarding changes in requirements, its ability to visualize project processes, and its role in increasing communication and cooperation among team members [8].

A study by Ladas [9] combined Scrum and Kanban to introduce a methodology which represents the best elements of those methodologies. According to that study, Scrumban is more appropriate for teams that are already using Scrum. Scrumban applies Scrum as a prescriptive method of team-work to complete the work, while it encourages process improvements through Kanban to allow projects to continuously optimize the processes and number of tasks [10].

There are limitations with respect to Scrum which can be mitigated by using Kanban alongside it. For example, Scrum does not consider the organization as a whole during its implementation [11] and has limitations such as lack of work visibility and changing task priorities [12]. Scrumban thus inhibits companies from embracing change and establishing better relationships between business and information technology departments [13]. Scrum and Kanban can be combined for high throughput and visibility into the development process.

Geographically distributed teams with poorly planned coordination often end up with unmatched deadlines, cost overruns, and even cancelled projects [14]. Distributed software development settings (DSD) often have additional challenges such as different locations, times, cultures, and languages among team members which can add more complexities to software development [15]. The idea of utilizing manpower from different locations is tempting, but it creates excessive coordination tasks in projects makes it difficult to ensure that everyone has a clear idea of the project goals and is committed to achieving them.

A study conducted by Šmite et al. [16] compares the characteristics of Agile and DSD and declares that communication in Agile projects is informal, face-to-face, and synchronous, while DSD projects require formal, computer-mediated, and often asynchronous communication. Moreover, Agile projects apply change-driven and self-managed coordination and light-weight control. However, DSD settings need plan-driven and standardized coordination among sites, which is achieved through several command and control methods. Despite their opposing characteristics, the combination of Agile and distributed development is of high interest to companies [17].

Geographically distributed development, in itself, is a vague term because there can be different types of distributed teams based on the time difference between the involved team members [18]. Two configurations of distributed teams that can be taken into consideration are North-South and East-West. North-South distributed teams do not have a considerable difference in time-zones, while the East-West configuration involves a significant time-zone difference [18]. Our investigated software factories (described in Sect. 3.1) were distributed from the North to South of Europe; hence, the East-West setting is beyond the scope of this research.

According to Šmite et al. [16], there is limited research and understanding about the application of Agile methodologies in DSD. In addition, Scrumban is a new development approach in the software engineering domain, and existing literature provides little information on Scrumban's impact on DSD projects. Increasing interest in globally distributed software development practices has motivated us to investigate the following research question: "How does Scrumban methodology affect global software development?" As coordination among developers is a critical issue within those environments, we have mainly investigated Scrumban from this perspective.

This paper is structured as follows: Sect. 2 presents an overview of previous studies on Scrum, Kanban, and Scrumban in the context of software engineering and similarities and differences among them. Further, it discusses global software development. Section 3 provides a brief introduction to the Software Factory settings and then elaborates on the T-Bix project case which has been used for this research. Next, it shows the project coordination model and the applied research approach. Section 4 presents findings of this study, the limitations, and direction for future studies. Section 5 concludes the paper and highlights the main contributions of our work.

2 Related Works

This section discusses Scrum, Kanban, and Scrumban with respect to their similarities and differences. It also describes software development practices in a global context.

2.1 Scrum

The first implementation of Scrum in the field of software development was at Easel Corporation in the USA in 1993 [19]. Scrum advocates small teams that work independently and create more efficiency at work [20]. Scrum is an incremental Agile software development methodology that operates through a series of iterations that require continuous planning, defined roles, and project artefacts [21,22]. Scrum is the most frequently applied Agile software development method [3] for achieving small but continuous deliverables. It facilitates regular feedback after each iterative development process, called a "sprint" [23]. According to Rising and Janoff [20], Scrum is beneficial, particularly for projects in which all the requirements are not clear in advance.

Implementation of Scrum allows self-organization which can result in a high-performance team even if the team comprises average developers [19]. The most important roles in Scrum include: (1) product owner, who serves as an interface among developers and other stakeholders, (2) Scrum master, who is the person responsible for leading scrum meetings, identifying tasks to be completed within the sprints, and measuring progress [20], and (3) development team. Since companies from Western countries often tend to outsource their software development to Eastern countries [19], applying Scrum in such situations can induce independency of teams as well as increase communication and productivity.

2.2 Kanban

Kanban is a relatively new concept in the field of software engineering that was originally applied in Lean manufacturing [6]. While Scrum focuses on one iteration at a time, Kanban supports a continuous workflow [5]. Kanban provides the flexibility of managing the workflow within teams. It limits WIP in each activity to a maximum number of tasks or items at any given time. Moreover, it does not suggest strictly defined roles and sprints [24]. It provides a clear visualization of the phases in the project lifecycle.

Kanban reduces lead time and improves quality and productivity [25]. Kanban helps team members to identify constraints of a process and focus on a single item or task at a time [26]. In traditional software development methods, several works are assigned to a team member, which is defined as a push method [26]. In that case, the work to be completed is sent to the team member regardless of the status of other work. On the other hand, Kanban suggests assigning a developer to one particular job. When the work is completed, the developer can pull another task from the Kanban board and work on it. According to Polk [27], provision of a Kanban board changed the thinking of team members by making them realize that they are not just developing code but developing a complete product. With Kanban, team members, stakeholders, and customers can get a real-time view of project progress [26]. Implementation of Kanban also lowers the risk of communication and coordination breakdown [28].

2.3 Scrumban

By combining Lean and Agile methodologies, project members can receive rapid and iterative feedback while they have the ability to implement the necessary changes and respond to the feedback. The combination of Agile and Lean in co-located projects enhances coordination among team members, increases team morale, and produces better outcomes [13]. Lean increases the scale of the development process and makes it efficient, while Agile principles help to make the process flexible [11].

Table 1 summarizes the key points of using Scrum and Kanban in the same project by showing several examples from the literature. In Sect. 4.1, we will use these points for our analysis in the context of distributed software development.

Scrum and Kanban are similar in the sense that both improve transparency, aim to release software as soon as possible, work on the principle of breaking work into pieces, and continuously optimize the project plan [29]. It is argued that if Kanban is used alongside Scrum, they can complement each other [30]. Scrumban incorporates the iterative planning of Scrum but is more responsive and adaptive to changes in user requirements. Project members who have had good experience with Scrum can benefit from Scrumban, as it improves their knowledge and capabilities [30]. By combining Scrum and Kanban, researchers hope to create more flexibility in projects as well as maintaining the iterative pace that Scrum has provided [30].

Table 1. Scrum and Kanban methodological elements.

ID	Study place	Key points	Reference
P01	Vietnamese office of a Swedish software development company	Scrum: Iterative and incremental Regular feedback Strict roles and rules Kanban: Visualization Limiting WIP Scrumban: Self-organizing Collaborative teamwork	Nikitina et al. [24]
P02	Faculty of Computer and Information Science, University of Ljubljana	Scrum: Incremental and iterative Planned project Regular feedback Kanban: Maximize workflow Visualization Limiting WIP	Mahnic [5]
P03	Arrk Group: a multinational software development company	Scrumban: Limiting WIP Optimal resource utilization Collaborative teamwork Quick decisions Customer satisfaction	Joshi and Maher [31]
P04	GoGo: offers services such as Internet, entertainment, messaging, voice in the aviation market	Scrumban: Visualization of workflows Transparency Increased team participation	Brinker [32]

One factor that Scrumban inherits from Kanban is the visualization of workflows [10]. Scrum completes tasks through sprints that are planned in advance, but Scrumban allows more flexibility and planning only for the following sprint. This helps projects to limit WIP. When the limit of tasks in a particular workflow is reached, team members help each other to complete the tasks in that workflow rather than starting a new one. This increases the coordination among team members and also reduces the possibility of a bottleneck [10].

Scrumban, unlike Scrum, has no strict rules and roles and encourages self-organized teams. As a result, team members manage their tasks by themselves and make quicker decisions. According to Khan [10], Scrumban reduces the relevant tasks of planning for the whole iteration (the same as Scrum), as meetings are set only when required and tasks are changed depending on the output of the ongoing sprint.

A real case example of a company's transition from Scrum to Scrumban [4] reports that implementation of Scrumban provided a systematic improvement in the performance of developers. Additional features of Scrumban over Scrum such as WIP limit and pull-based task management are received well by the team members. Table 2 has summarized findings from Yilmaz and O'Connor [4], Reddy [7], and Ahmad et al. [33] to highlight the similarities and differences between Scrum, Kanban, and Scrumban.

Table 2. Similarities and differences among Scrum, Kanban and Scrumban [4,7,33].

Kanban	Scrum	Scrumban
No predefined roles for members	Predefined roles of Scrum master and team members	Predefined roles of Scrum master and team members may vary within project time
Continuous delivery	Time-boxed sprints	Task board-based iterations
WIP limits amount of work	Sprint limits amount of work	WIP limits amount of work
Changes can be made at any time	No changes allowed mid-sprint	Changes allowed mid-sprint
Earlier planning and documentation necessary	Planning done after each sprint	Planning on demand, also within sprints
Kanban board is persistent	Scrum board is reset after each sprint	Scrumban board is persistent
Size of task is not limited	Size of task limited to a sprint	Size of task is not limited
Pull-based work management	Sprint backlog-based work management	Pull-based work management

The implementation of Scrumban, however, presents several challenges. The flexibility regarding production changes can cause new challenges in, for example, assigning resources and developing project time-tables. Because Lean methodology calls for considering the whole organization through implementation [34], the combination of Kanban and Scrumban increases the complexities of planning for activities across the whole organization. Moreover, it is not always possible to include business personnel or management executives developing project backlogs or contributing regular feedback [11].

2.4 Distributed Software Development

Finding resources globally creates the possibility of mobility in resources and of accessing new knowledge of skilled people around the world [17]. Global software

development is applied through multi-geo, multicultural, and multi-temporal environments to benefit from access to new markets, lower costs, increased operational efficiency, improved quality, and less time to markets [35].

DSD could have different configurational characteristics, which refers to the structural properties of the global environment, different ways of distributing developers, and differences in time and physical distance. A study by Ramasubbu et al. [36] examined how configurational dimensions can affect productivity, quality, and profit outcomes of distributed projects. This study explains aspects of dispersions including spatial dispersion to measure the physical distance, temporal dispersion to measure the time-zone difference, and configurational dispersion to measure structural properties such as number of distributed sites and homogeneity of distributed people and skills across different sites.

Šmite et al. [17], Jiménez et al. [37], and Nakamura et al. [38] declare that realizing the DSD benefits come with associated challenges in terms of communication gaps between multiple sites, group awareness, software configuration management, knowledge management, flexible coordination, collaboration, project management, process support, tools support, quality management, and risk management.

Coordination is a pressing issue in global software development. People at the research and development center of Yahoo in Norway mentioned that the time-zone difference was a major cause of problems when dealing with dislocated teams [18]. According to Noll et al. [39], the main barriers to coordination in distributed projects are geographic, temporal, cultural, and linguistic differences. That study proposed that project teams should enhance site visits, use synchronous communication technology, and apply knowledge-sharing infrastructure to transform implicit knowledge to explicit knowledge [39]. Other scholars such as Mak and Kruchten [40], Redmiles et al. [41], and Sidhu and Volberda [42] have argued that coordination issues come from (1) lack of flexibility and integration, (2) poor role support, (3) decreasing informal communication and workplace transparency, and (4) limitations imposed on formal communication. However, the levels of impact of these issues are different in different dispersion configurations; for example, a study conducted by Ramasubbu et al. [36] suggests that establishing a project in an East-West geographical setting requires radically more consideration of time-zone classifications than North-South settings.

There are several instances of the application of Scrum in distributed development projects [1,16,18,43]. The American software consulting company Agile Factori implemented a successful software development project using Agile methodologies. The project was provided by Big Oil, an American company consisting of four teams, two of which were located in America and the other two in Brazil and Argentina. All four teams had a real-time video screen with audio that showed activities at the other sites. In addition, one screen at each site showed a dashboard of in-process software components. This allowed other sites visualization, increased awareness, and better coordination among teams [18].

SirsiDynix (U.S) has successfully implemented distributed Scrum since 2005 [19,43]. Using distributed Scrum, SirsiDynix collaborated with the Russian company Exigen in 2005 for a large project [43] employing more than 50 members in total and producing over one million lines of code. The output of this distributed team was estimated to be equivalent to the work of a 350 co-located-person team working in a waterfall model [43].

An international Agile software development company, Xebia, located in France, India, and the Netherlands, also implemented Scrum successfully between 2006 and 2008 [43]. Distributed Scrum was used alongside XP programming in multiple projects by Xebia, and the results showed that the distributed teams were as effective as co-located teams. These instances show that globally distributed teams can be as productive as co-located teams when Scrum is applied effectively [19,44].

3 Research Process

3.1 Software Factory

Software Factories (SF) include structured sets of related software assets to provide developers with a development setting consisting of domain-specific tools that help to transform abstract models into implementations [26,45,46]. Through the SF settings, reusable development practices such as patterns, models, guidelines, and transformations are accessible from the viewpoint of a specific aspect in the development context. This enables domain-specific validation and guidance delivery [47].

The SFs increase productivity from the business perspective, improve quality and consistency of architectures and designs, reduce development lifecycle, and consolidate operational efforts [47]. Also, SFs established in the context of universities are perfect avenues for exploiting technological research for innovation. A study by Taibi et al. [48] declares that such an SF environment benefits both business by receiving innovative, new ideas and academia by presenting new skills, frameworks, and models. Therefore, academic Software Factories can be additionally considered as a new concept of collaboration among universities and companies [48].

3.2 T-Bix Project Case

A joint five-month software development project called T-Bix was initiated between the University of Oulu, Finland and the University of Bolzano, Italy in their respective Software Factories. The aim of the project was to develop a single common platform for the time-banking system to be operational in South Tyrol in Italy. The platform allows users to register their own profiles, search for jobs and products, post jobs and products, send requests for jobs and products, and communicate their feedback. The target group was young, unemployed people as well as senior citizens hit by the socio-economic crisis.

Because T-Bix project teams were located in Europe (North-South dispersion configuration), they did not experience drastic temporal differences; however, the long physical distance and diverse cultures, languages, and social behaviors remained as challenges in the project. The Finnish team consisted of one PhD candidate and four master's degree students who were working locally in Oulu. The team from Italy had a Software Factory coordinator with a PhD degree and four master's degree students. A member of the Italian team was working remotely from Lithuania. There was one student on each team with industrial experience; however, the rest of the teams did not have prior experience in industry. Each team comprised one project manager and three developers. The Finnish team used Scrumban, while the other team used Scrum as the development methodology in Italy. Teams frequently used available tools and assets in SFs, including Rise Editor, Myeclipse, Dreamweaver, JIRA, and GitHub. There was an Italian entrepreneur (customer of the T-Bix project) who was in direct contact with both teams. The customer communicated his needs through meetings and emails; teams attempted to interpret the customer's requirements into user stories and backlogs.

The front-end of the application was developed with direct contact with the customer in Italy. The back-end, including the database development and integration of the front-end and back-end, was developed in Finland. The codes were shared on GitHub, where some feedback and comments were also shared.

An identical Kanban board was created in JIRA by the Oulu team and shared with the team members in Bolzano. The Kanban board was updated regularly, providing visibility of the board and tasks across both teams.

In addition to JIRA boards, the Software Factory in Oulu was equipped with physical Kanban boards that were utilized throughout the project lifetime. The boards were divided into four sections: backlog (features), to do, in progress (WIP), and done, and was populated with user stories planned in each sprint. Once each sprint was completed, the team in Finland updated the boards with new tasks and shifting completed jobs to the "done" section. Figure 1 shows a snapshot of the board.

Teams used collaboration tools such as Skype and Google Hangout to communicate and verify the requirements, monitor the project progress, present the sprint deliverables, discuss the challenges, and set new milestones. After each sprint, teams presented their respective outcomes and progress and received feedback from both the customer and other team members.

Finding the best time for meetings is a major concern in global software development [18]. Members from the T-Bix project met on a planned time-table which considered the temporal difference between Italy, Finland, and Lithuania. To accommodate other sites and the customer, the meetings were often held in the afternoon. This ability to hold meetings during the daytime without much time shifting is an advantage provided by North-South collaboration.

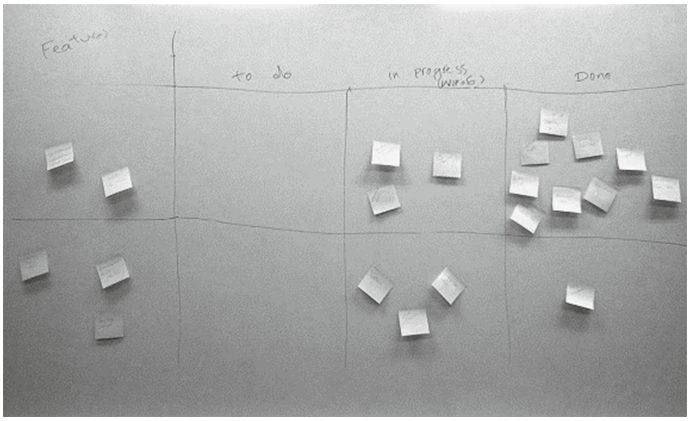

Fig. 1. A physical Kanban board in Oulu software factory.

3.3 Project Coordination Model

The project was proposed by the customer to the University of Bolzano with the aim of decreasing the rate of unemployment in South Tyrol. Subsequently, the University of Bolzano had the idea of making the project a distributed Software Factory project between the two universities.

The customer was in contact with the teams with respect to the elicitation of requirements, acceptance testing, and the validation of artefacts. The user interface of the website was designed and validated through regular meetings with the customer. The codes and designs were continuously uploaded in GitHub, in which both teams updated their latest work. The next sprint was planned according to the feedback and suggestions made by the customer and both teams. The following model (Fig. 2) shows how the project was carried out among the teams.

3.4 Research Approach

This study exploits empirical software engineering methods. The authors have applied semi-structured interviews to collect the data. The participants of this study are members of the Oulu Software Factory who were interviewed after completion of the project. The authors provided a set of open-ended questions covering the scope and objectives of this paper. Four rounds of interviews were conducted, which lasted from 45 min to 2 h. All interviews were recorded and transcribed, enabling authors to analyze them based on the needs of this study.

A semi-structured interview format was preferred, as it provides a clear set of instructions for the interviewer, who usually follows a paper-based interview guide during the interview. The availability of questions beforehand makes the interviews easier for the interviewer and the openness of this type of interview provides the interviewees with the freedom to express their views using their own terms.

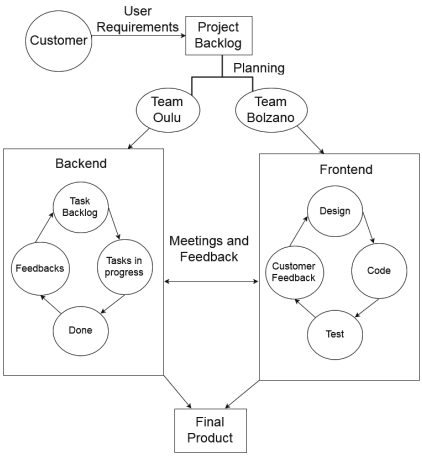

Fig. 2. Project coordination model.

Table 3. Interviewees' backgrounds.

Interviewee	Role in the project	Empirical experience	Expertise
D1	Project manager	10 years	Project management, UI Design, JIRA&GitHub
D2	Programmer	—	PostgreSQL, JIRA&GitHub
D3	Programmer	—	PostgreSQL, Java, JIRA&GitHub
D4	UI Designer	—	UI Design, Java, JIRA&GitHub

In addition, the comparable qualitative data obtained from semi-structured interviews is regarded as reliable for analysis [49]. Table 3 summarizes the roles, empirical experiences, and expertise of the interviewees.

We have defined a set of the coding categories based on the top challenges of the DSD projects. Those challenges have been extracted from prior studies in the DSD domain. The categories were discussed and confirmed by two authors of this paper. All challenges are defined in detail in the next section. Afterward, authors read through the transcripts and underlined each fragment of relevant information and specified which fragment fell into which DSD challenges categories. We have provided some examples of the specified fragments in the next section. Finally, the reliability of the results was tested separately by two authors of this paper, and they each found the same results.

4 Results

This section summarizes key findings regarding Scrumban's impact on the T-Bix project as well as the limitations and opportunities for future research.

4.1 Findings

Table 4 explains how the impact of Scrumban has been realized in the coordination between North-South distributed sites. For this purpose, we have investigated the top issues in DSD projects as already introduced by other scholars, including Carmel and Espinosa [18], Barcus and Montibeller [50], Espinosa et al. [51], and Nidiffer and Dolan [52].

Table 4. Impact of Scrumban on coordination in DSD environments.

Scrumban aspects		Issues in distributed software development				
	Strategic	Project and process management	Communication	Cultural	Technical	Security
Iterative and incremental development	Highly improved toward latest sprints	Highly improved toward latest sprints	More sprints, smoother	More iterations, fewer challenges	Slightly improved	No evidence
Predictable and well-planned project	No meaningful impact on leveraging resources at the other site	More iterations, more improvement	Effective communication for the planned tasks	No evidence	Slightly improved toward latest sprints	No evidence
Transparency	Positively impacted leveraging resources at both sites	Positively impacted task management within sites	No evidence	Slightly reduced challenges	No evidence	No evidence
Regular feedback	Slightly improved	Positively impacted task management within sites	Demands of both formal and informal feedback	Improved toward latest sprints	No evidence	No evidence
Limiting WIP	Positively impacted resource management	Decreased challenges slightly	No evidence	No evidence	No evidence	No evidence
Self-organizing	Slightly improved	Positively impacted task management within sites	Improved informal communication	No evidence	No evidence	No evidence

Strategic issues within DSD settings are concerned with the difficulty in leveraging available resources. Issues in which stakeholders can anticipate and manage risks should be identified carefully [52]. Because T-Bix was an evolutionary project done through iterative sprints, teams were able to find new ways to leverage available resources and skills. Within the initial meetings, two teams discussed the experience and expertise of their members, clarifying how the project

resources were divided between the two sites and how the project duties should be assigned.

However, the team members mentioned their increasing responsibility during the later sprints of the project. The project manager [D1] explained that they were asked to accomplish some additional work on coding. Adapting to these workflow changes made it difficult to complete the project. A developer [D4] explained that after much discussion, the two teams decided to assign additional tasks to the Oulu team, as they had more technical skills:

"After much discussion, we had to accept more work, as Bolzano was not able to complete it. We should provide more deliverables at the end of the project. We had no choice because we wanted the project done."

The teams applied JIRA to establish the project structure and define the roles of the two sites. Project tasks were assigned to the teams' members according to their roles and skills. Furthermore, JIRA created visibility in the WIP for each role compared to other developers. The project manager [D1] confirmed this:

"Using JIRA, I could monitor the progress of different completed tasks with respect to the roles. It provided me an opportunity to recognize the tasks that required extra coordination."

Project and process management in DSD involves discussing problematic situations in synchronizing work between distributed sites [50]. Integrated quality, shared workspaces for storing files, and engineering tools are potential enablers of this issue. Complexity also arises from the fact that there should be sufficient communication between two teams before they can prioritize project tasks and decide which one is to be carried out by which team, as in the case of the T-Bix project discussed. The teams had agreed upon a preliminary division of work, but additional tasks were later added to the project by the customer. The members mentioned that the added tasks caused several challenges in managing their ongoing tasks. To control the scope of the project, the involved teams should manage changes in a planned way. Any changes in the project scope may affect the priority and division of work among the sites. The project manager [D1] declared the following primary decision criterion for allocating tasks between sites:

"Consistency between the requested feature and available skills and knowledge at the sites was our decision criterion for allocating tasks to sites."

Using Kanban boards in JIRA improved the visualization and transparency of the completed, ongoing, and planned tasks. One developer [D3] mentioned the following:

"The Kanban board in JIRA was quite helpful because we could not frequently update the pictures of the physical Kanban board for the other team. We applied JIRA's Kanban board to share the tasks we had completed and planned to do."

Using JIRA and GitHub, project members received feedback on their jobs, for example, for the codes that were uploaded in GitHub. One of the developers [D2] stated the following:

"For example, when the scripts in the database had problems, one of the programmers in Bolzano was using GitHub to send feedback regarding the issues and asking for solutions."

Another developer [D4] also believed the following:

"JIRA is a tool developed for task management purposes, but you cannot upload all project deliverables into it. It is not a shared platform, so we needed to use other tools, in which we could share other data."

Communication issues are related to the lack of effective communication mechanisms. It is very important to convey information such as the current state of the project as well as project challenges, schedule, and cost. In the case of distributed projects, communication plays an important role in collaboratively planning the project stages. Along with formal communication, informal communication between team members and with stakeholders can ease the working environment and facilitate coordination among them [29]. Applying Scrumban in DSD projects demands both formal and informal styles of communication. Informal communication facilitates project implementation; however, formal communication creates a disciplined environment, which is necessary for coordination in DSD sites.

Communication was regarded as an important tool for ensuring that the T-Bix teams were placed at the same level of understanding regarding the project. A developer [D4] highlighted the following:

"The Bolzano team had their own understanding of the project and we had ours. We had discussions to resolve the discrepancies and create balance between the two teams. Scrumban provoked us to have regular meetings with team members as well as the customer. This increased the level of communication in the project." However, the project manager [D1] mentioned that different time-zones created little discomfort for arranging meetings.

Scrumban leads to a great deal of communication. One developer [D4] argued the following:

"At first, we had many problems in our communication because the project members complained that the other site hindered the project progress and was not completing its tasks well."

However, new communication channels as well as more effective planning in the project led to a higher level of communication between teams. It was claimed that:

"We had many challenges in our discussions, but since people have had more communication and became increasingly more acquainted with the way the other team works, communication became smoother."[D1]

Cultural issues involve the conflicting behavioral processes and technologies among various team members [52]. Different socio-cultural backgrounds make communication more complicated due to lack of understanding of other social behaviors, cultures, and languages. The T-Bix project shows that people have different expectations regarding working in multinational teams; for example, one developer [D2] explained the following:

"It was quite good for distributed software development to include multiple cultures. It was interesting to work with people with different backgrounds."

However, other people found multi-cultural settings more difficult than co-located projects.

Due to the nature of Software Factory projects, team members were completely new to each other and they were assigned to this project with no prior knowledge of the other team members. At the beginning of the project, they had several challenges in communicating with each other and establishing good organization for their project; however, the evolution of the teams and the feedback on the requirements and skills alleviated cultural barriers after people had met for several sprints.

Technical issues in DSD environments are related to incompatible data formats and exchanges. Creating standards and web services could be seen as potential ways to resolve this issue. The T-Bix case shows that during different sprints, teams progressively realized the technical abilities and needs of other sites. The iterative nature of Scrumban helped them to meet those needs and prepare to meet the internal project standards and agreements.

Security, on the other hand, involves ensuring electronic transmission confidentiality and privacy [52]. It can be improved through emerging standards for secure messaging. The T-Bix project did not provide meaningful evidence of Scrumban's impact on improving security issues in DSD settings. However, this study was conducted with respect to coordination issues, and other project issues are beyond its scope.

4.2 Limitations

The research was conducted in a Software Factory where we collected member experiences from a business case with a real customer outside the university. According to Fagerholm et al. [53], student selection process can serve as a limitation in Software Factory projects because universities apply different prerequisites and standards for their selection process. In our case, the two universities followed different selection methods, resulting in differences in the level of knowledge and skills between the two teams.

The authors believe that the limited number of the interviewees is the most critical limitation of this paper. The semi-structured interview design for the university environment led to both the benefits and the limitations of this research. Prior studies argue for the clear benefits of using students as empirical research subjects [54,55]; although academic projects rarely can be defined on a large scale. To that end, it is necessary to carry out further studies within a larger context in an industrial setting to verify our findings.

This research was done within three European countries. Therefore, the DSD configuration is North-South, with little difference in time-zones. We assume that management of teams with a higher level of time variation, for example East-West configuration, might have more issues. Greater differences in time-zones, for instance from India to the U.S., will cause more coordination challenges for

joint meetings and on-time responses to emails, as well as more drastic changes in culture and languages.

4.3 Future Research

There are a limited number of studies and industrial practices on the applicability, challenges and benefits of Scrumban in software engineering. The Software Factory setting is an attractive concept for testing new ideas and methodologies related to software development. However, there are challenges related to maintaining software artefacts after the completion of Software Factory projects [56]. The authors believe that the results of the T-Bix project can be maintained in future academic research, for example, by involving other Software Factories from different time-zones. It is important to continue studying the impact of Scrumban on East-West distributed teams with a greater variation in time-zones. In addition, companies with distributed sites suffer from similar challenges; therefore, they would be the best candidates to apply Scrumban and provide feedback for designing future Software Factory projects.

An interesting future study would be investigating how the results of this study would work in a large industrial environment where there are multiple sites involved at a variety of locations. We believe that those environments could nicely validate the capability of Scrumban to improve coordination in DSD projects.

This study primarily emphasized coordination aspects within distributed sites. Other issues require further investigation with respect to proper methodologies, workflows, and tools. Also, we did not consider impact parameters such as age, education, and years of experience on the use of Scrumban, which could apparently be a very good research topic.

5 Conclusions

Current software companies tend to establish their production units in different locations in order to optimize skilled workforces to produce products at higher quality and lower cost. In that regard, companies need adequate methodologies, techniques, and tools to improve efficiency and decrease challenges in DSD. Software Factory settings can reuse existing assets, architectures, knowledge, and components to develop software artefacts by imitating industrial processes.

The current study has used Software Factories to investigate how coordination among distributed sites is effected by the combination of Scrum and Kanban. Our research shows that the full extent of Scrumban capability is still unknown because it has not been researched a great deal. Therefore, the results of this research can be used as the initial steps for developing and validating an efficient methodology for software engineering practices, particularly in distributed sites.

There are different issues which should be considered before companies decide to locate their branches in various remote sites. This study argues that Scrumban could alleviate some of those challenges, but further solutions are needed to make

DSD more reasonable than co-located developments. Furthermore, companies must restructure their organizations to include proper roles and processes to improve transparency, change management, communication, coordination, and resources in DSD.

Future scholarly studies could investigate perspectives other than coordination for the usability of Scrumban. Moreover, they could propose new domain-specific tools and approaches for DSD projects which impose different constraints, for example, East-West distributed projects. However, companies, as the real users of Scrumban methodology, should be aware of its challenges as well as its benefits in planning project deliverables and coordination among teams.

Acknowledgements. This research was supported by the DIGILE Need for Speed program, and partially funded by Tekes (the Finnish Funding Agency for Technology and Innovation). We would like to thank DIGILE and Tekes for their support and the University of Bolzano for its excellent collaboration.

References

1. Schwaber, K., Sutherland, J.: Software in 30 Days: How Agile Managers Beat the Odds, Delight Their Customers, and Leave Competitors in the Dust. John Wiley & Sons Press, Hoboken (2012)
2. Alam, S.S., Chandra, S.: Agile software development: novel approaches for software engineering. Int. J. Eng. Sci. (IJES) **3**(1), 36–40 (2014)
3. Rodriguez, P., Markkula, J., Oivo, M., Turula, K.: Survey on agile and lean usage in finnish software industry. In: Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 139–148. ACM Press (2012)
4. Yilmaz, M., O'Connor, R.: A Scrumban integrated gamification approach to guide software process improvement: a Turkish case study. Tehnicki Vjesnik **23**(1), 237–245 (2016)
5. Mahnic, V.: Improving software development through combination of scrum and Kanban. Recent Advances in Computer Engineering, Communications and Information Technology, Spain (2014)
6. Ahmad, M.O., Markkula, J., Oivo, M.: Kanban in software development: A systematic literature review. In 39th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), pp. 9–16. IEEE Press (2013)
7. Reddy, A.: The Scrumban [R]Evolution: Getting the Most Out of Agile, Scrum, and Lean Kanban. Addison-Wesley Professional, Boston (2015)
8. Kniberg, H., Skarin, M.: Kanban and Scrummaking the most of both. The InfoQ Enterprise Software Development (2010)
9. Ladas, C.: Scrumban. Lean Software Engineering-Essays on the Continuous Delivery of High Quality Information Systems (2008)
10. Khan, Z.: Scrumban-adaptive agile development process: using scrumban to improve software development process. Master's Thesis, Finland (2014)
11. Rodriguez, P., Partanen, J., Kuvaja, P., Oivo, M.: Combining lean thinking and agile methods for software development: a case study of a finnish provider of wireless embedded systems detailed. In: 47th Hawaii International Conference on System Sciences (HICSS 2014), pp. 4770–4779. IEEE Press (2014)

12. Tripathi, N., Rodríguez, P., Ahmad, M.O., Oivo, M.: Scaling Kanban for software development in a multisite organization: challenges and potential solutions. In: Lassenius, C., Dingsøyr, T., Paasivaara, M. (eds.) XP 2015. LNBIP, vol. 212, pp. 178–190. Springer (2015). doi:10.1007/978-3-319-18612-2_15
13. Auerbach, B., McCarthy, R.: Does agile+ lean= effective: an investigative study. J. Comput. Sci. Inf. Technol. **2**(2), 73–86 (2014)
14. Smith, J.L., Bohner, S., McCrickard, D.S.: Toward introducing notification technology into distributed project teams. In: 12th IEEE International Conference and Workshops on the Engineering of Computer Based Systems (ECBS 2005), pp. 349–356. IEEE Press (2005)
15. Gupta, M., Fernandez, J.: How globally distributed software teams can improve their collaboration effectiveness? In: 6th IEEE International Conference on Global Software Engineering (ICGSE), pp. 185–189. IEEE Press (2011)
16. Šmite, D., Moe, N.B., Ågerfalk, P.J.: Fundamentals of agile distributed software development. In: Šmite, D., Moe, N., Ågerfalk, P. (eds.) Agility Across Time and Space. LNCS, pp. 3–7. Springer (2010). doi:10.1007/978-3-642-12442-6_1
17. Šmite, D., Moe, N.B., Agerfalk, P.J.: Agility Across Time and Space: Implementing Agile Methods in Global Software Projects. Springer Science & Business Media, Heidelberg (2010)
18. Carmel, E., Espinosa, J.A.: I'm Working While They're Sleeping: Time Zone Separation Challenges and Solutions. Nedder Stream Press, Washington, DC (2011)
19. Sutherland, J., Viktorov, A., Blount, J., Puntikov, N.: Distributed scrum: agile project management with outsourced development teams. In: 40th Annual Hawaii International Conference on System Sciences (HICSS 2007). IEEE Press (2007)
20. Rising, L., Janoff, N.S.: The scrum software development process for small teams. IEEE Softw. **17**(4), 26–32 (2000)
21. Schwaber, K.: Agile Project Management with Scrum. Microsoft Press, Redmond (2004)
22. Schwaber, K., Beedle, M.: Agile Software Development with Scrum. Pearson International Edition, New York (2002)
23. Nikitina, N., Kajko-Mattsson, M.: Guiding the adoption of software development methods. In: Proceedings of the 2014 International Conference on Software and System Process, pp. 109–118. ACM Press (2014)
24. Nikitina, N., Kajko-Mattsson, M., Strale, M.: From scrum to scrumban: a case study of a process transition. In: Proceedings of the International Conference on Software and System Process, pp. 140–149. IEEE Press (2012)
25. Sjøberg, D.I., Johnsen, A., Solberg, J.: Quantifying the effect of using kanban versus scrum: a case study. IEEE Softw. **29**(5), 47–53 (2012)
26. Ahmad, M.O., Liukkunen, K., Markkula, J.: Student perceptions and attitudes towards the software factory as a learning environment. In: Global Engineering Education Conference (EDUCON), pp. 422–428. IEEE Press (2014)
27. Polk, R.: Agile and Kanban in coordination. In: AGILE Conference, pp. 263–268 (2011)
28. Ikonen, M., Pirinen, E., Fagerholm, F., Kettunen, P., Abrahamsson, P.: On the impact of Kanban on software project work: an empirical case study investigation. In: 16th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS), pp. 305–314. IEEE Press (2011)
29. Barash, I.: Use of agile with XP and Kanban methodologies in the same project. PM World J. **2**(2), 1–11 (2013)
30. Ladas, C.: Scrumban-Essays on Kanban Systems for Lean Software Development. Modus Cooperandi Press, Seattle (2009)

31. Our Journey into Scrumban. http://www.arrkgroup.com/thoughtleadership/our-journey-into-scrumban/
32. Using Scrumban (Scrum Kanban) for Agile Marketing - Chief Marketing Technologist. http://chiefmartec.com/2014/12/using-scrumbanlean-agile-marketing/
33. Ahmad, M.O., Kuvaja, P., Oivo, M., Markkula, J.: Transition of software maintenance teams from scrum to Kanban. In: 49th Hawaii International Conference on System Sciences (HICSS 2016), pp. 5427–5436. IEEE Press (2016)
34. Karvonen, T., Rodriguez, P., Kuvaja, P., Mikkonen, K., Oivo, M.: Adapting the lean enterprise self-assessment tool for the software development domain. In: 38th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), pp. 266–273. IEEE Press (2012)
35. Sutanto, J., Kankanhalli, A., Tan, B.C.: Deriving it-mediated task coordination portfolios for global virtual teams. IEEE Trans. Prof. Commun. **54**(2), 133–151 (2011)
36. Ramasubbu, N., Cataldo, M., Balan, R.K., Herbsleb, J.D.: Configuring global software teams: a multi-company analysis of project productivity, quality, and profits. In: Proceedings of the 33rd International Conference on Software Engineering, pp. 261–270. ACM Press (2011)
37. Jiménez, M., Piattini, M., Vizcaino, A.: Challenges and improvements in distributed software development: a systematic review. Adv. Soft. Eng. **2009**, 3 (2009)
38. Nakamura, K., Fujii, Y., Kiyokane, Y., Nakamura, M., Hinenoya, K., Peck, Y.H., Choon-Lian, S.: Distributed and concurrent development environment via sharing design information. In: The Twenty-First Annual International Computer Software and Applications Conference, 1997, COMPSAC 1997, Proceedings, pp. 274–279. IEEE Press (1997)
39. Noll, J., Beecham, S., Richardson, I.: Global software development and collaboration: barriers and solutions. ACM Inroads **1**(3), 66–78 (2010)
40. Mak, D.K., Kruchten, P.B.: Task coordination in an agile distributed software development environment. In: Canadian Conference on Electrical and Computer Engineering, CCECE 2006, pp. 606–611. IEEE Press (2006)
41. Redmiles, D., Van Der Hoek, A., Al-Ani, B., Hildenbrand, T., Quirk, S., Sarma, A., Filho, R., de Souza, C., Trainer, E.: Continuous coordination: a new paradigm to support globally distributed software development projects. Wirtschafts Informatik **49**(1), 28–38 (2007)
42. Sidhu, J.S., Volberda, H.W.: Coordination of globally distributed teams: a co-evolution perspective on offshoring. Int. Bus. Rev. **20**(3), 278–290 (2011)
43. Sutherland, J., Schoonheim, G., Rijk, M.: Fully distributed scrum: Replicating local productivity and quality with offshore teams. In: 42nd Hawaii International Conference on System Sciences (HICSS 2009), pp. 1–8. IEEE Press (2009)
44. Paasivaara, M.: Coaching global software development projects. In: 6th IEEE International Conference on Global Software Engineering (ICGSE), pp. 84–93. IEEE Press (2011)
45. Abrahamsson, P., Kettunen, P., Fagerholm, F.: The set-up of a software engineering research infrastructure of the 2010s. In: Proceedings of the 11th International Conference on Product Focused Software, pp. 112–114. ACM Press (2010)
46. France, R., Rumpe, B.: Model-driven development of complex software: a research roadmap. In: 2007 Future of Software Engineering, pp. 37–54. IEEE Computer Society (2007)

47. Greenfield, J., Short, K.: Software factories: assembling applications with patterns, models, frameworks, and tools. In: 3rd International Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA). ACM Press (2004)
48. Taibi, D., Lenarduzzi, V., Ahmad, M.O., Liukkunen, K., Lunesu, I., Matta, M., Fagerholm, F., Münch, J., Pietinen, S., Tukiainen, M., Fernández-Sánchez, C., Garbajosa, J., Systä, K.: "Free" innovation environments: lessons learned from the software factory initiatives. In: 10th International Conference on Software Engineering Advances (ICSEA 2015), pp. 25–30 (2015)
49. Cohen, D., Crabtree, B.: Qualitative Research Guidelines Project. Robert Wood Johnson Foundation, Princeton (2006)
50. Barcus, A., Montibeller, G.: Supporting the allocation of software development work in distributed teams with multi-criteria decision analysis. Omega **36**(3), 464–475 (2008)
51. Espinosa, J.A., Slaughter, S.A., Kraut, R.E., Herbsleb, J.D.: Team knowledge and coordination in geographically distributed software development. J. Manag. Inf. Syst. **24**(1), 135–169 (2007)
52. Nidiffer, K.E., Dolan, D.: Evolving distributed project management. IEEE Softw. **22**(5), 63–72 (2005). IEEE Press
53. Fagerholm, F., Oza, N., Munch, J.: A platform for teaching applied distributed software development: the ongoing journey of the Helsinki software factory. In: 3rd International Workshop on Collaborative Teaching of Globally Distributed Software Development (CTGDSD), pp. 1–5. IEEE Press (2013)
54. Höst, M., Regnell, B., Wohlin, C.: Using students as subjects a comparative study of students and professionals in lead-time impact assessment. Empirical Softw. Eng. **5**(3), 201–214 (2000)
55. Madeyski, L.: Test-Driven Development: An Empirical Evaluation of Agile Practice. Springer Science & Business Media, Heidelberg (2009)
56. Chao, J., Randles, M.: Agile software factory for student service learning. In: 22nd Conference on Software Engineering Education and Training (CSEET), pp. 34–40. IEEE Press (2009)