Using Simulation for Understanding and Reproducing Distributed Software Development Processes in the Cloud

M. Ilaria Lunesu[1], Jürgen Münch[2], Michele Marchesi[3], Marco Kuhrmann[4]

[1]Department of Electrical and Electronic Engineering, University of Cagliari, Italy

[2]Herman Hollerith Center, Böblingen & Reutlingen University, Germany

[3]Department of Mathematics and Computer Science University of Cagliari, Italy

[4]Clausthal University of Technology, Institute for Applied Software Systems Engineering, Germany

Corresponding Contact:
E-Mail: ilaria.lunesu@diee.unica.it

Using Simulation for Understanding and Reproducing Distributed Software Development Processes in the Cloud

M. Ilaria Lunesu[a,*], Jürgen Münch[b], Michele Marchesi[c], Marco Kuhrmann[d]

[a]*Department of Electrical and Electronic Engineering, University of Cagliari, Italy*
[b]*Herman Hollerith Center, Böblingen & Reutlingen University, Germany*
[c]*Department of Mathematics and Computer Science University of Cagliari, Italy*
[d]*Clausthal University of Technology, Institute for Applied Software Systems Engineering, Germany*

Abstract

Context: Organizations increasingly develop software in a distributed manner. The Cloud provides an environment to create and maintain software-based products and services. Currently, it is unknown which software processes are suited for Cloud-based development and what their effects in specific contexts are.
Objective: We aim at better understanding the software process applied to distributed software development using the Cloud as development environment. We further aim at providing an instrument, which helps project managers comparing different solution approaches and to adapt team processes to improve future project activities and outcomes.
Method: We provide a simulation model, which helps analyzing different project parameters and their impact on projects performed in the Cloud. To evaluate the simulation model, we conduct different analyses using a Scrumban process and data from a project executed in Finland and Spain. An extra adaptation of the simulation model for Scrum and Kanban was used to evaluate the suitability of the simulation model to cover further process models.
Results: A comparison of the real project data with the results obtained from the different simulation runs shows the simulation producing results close to the real data, and we could successfully replicate a distributed software project. Furthermore, we could show that the simulation model is suitable to address further process models.
Conclusion: The simulator helps reproducing activities, developers, and events in the project, and it helps analyzing potential tradeoffs, e.g., regarding throughput, total time, project size, team size and work-in-progress limits. Furthermore, the simulation model supports project managers selecting the most suitable planning alternative thus supporting decision-making processes.

Keywords: Scrum, Kanban, Process Simulation, Comparison.

1. Introduction

Being able to collaborate effectively has become a crucial factor in software development and maintenance. Organizations increasingly develop software in a distributed manner by appointing external developers and development teams, who collaboratively work at different sites utilizing a multitude of communication tools (Bird et al., 2009; Portillo-Rodríguez et al., 2012). Literature shows distributed software

*Department of Electrical and Electronic Engineering, University of Cagliari, Italy
Email address: ilaria.lunesu@diee.unica.it (M. Ilaria Lunesu)

development being challenged by many factors, e.g., distance in language, culture, time and location, coordination of distributed (virtual) teams, and lack of trust among developers (Sengupta et al., 2006; Herbsleb and Mockus, 2003). Notably agile software development constitutes a challenge, as agile software development relies on a set of principles and values that put the people and close collaboration and interaction in the spotlight. It is crucial to understand how agile methods "behave" in distributed software development as adapting and deploying an agile method to a project spanning several sites bears some risk (Lous et al., 2017).

A simulation-based approach grounded in statistical data from previous projects can help analyzing risks and evaluating different process variants (Kellner et al., 1999; Wakeland et al., 2004), but also helps evaluating decisions and potential effects on a project (Armbrust et al., 2005). Moreover, a process simulation offers insights faster than a full case study (Fagerholm et al., 2017, pp. 11–13). In particular, a *simulation model* can be modified and the results quickly provide indication whether or not modified parameters affect a project and how—so-called "what-if" analyses (Zhang et al., 2008). For example, while it is hard to modify the team in a "real" project, in a simulation, modifying the *team size* parameter helps analyzing the impact, e.g., on work-in-progress (WIP), lead/cycle time, and team productivity. Furthermore, a simulation model provides flexibility to allow for configuring different process models, running simulations on a shared dataset, and to compare and study aspects of interest of different process models. For instance, project managers interested in minimizing cycle times can use a simulation to compare the behavior of Scrum- and Kanban-based processes to pick the process variant promising the best performance. In this regard, a simulation can be utilized to modify parameters, find relations between parameters, and study complex processes over time. According to Kellner et al. (1999) and Armbrust et al. (2005), a simulation used this way can help reproducing a real system, compare variants, identify bottlenecks, and so forth. Hence, a process simulation is a tool to help project managers analyzing different actions, evaluating impact, and eventually selecting those actions best fitting a particular situation (Lunesu et al., 2017).

Problem Statement. Even though globally distributed software development (also called Global Software Development; GSD, or *Global Software Engineering; GSE*) is around for years, still, practitioners struggle with effectively adapting agile methods (Lous et al., 2017). In this context, the *Cloud* provides a highly flexible environment offering a variety of services. However, little is known which processes are used for distributed development using the *Cloud as software development environment*, how these processes are used and customized, and how they might differ from other approaches.

Objective. Our overall objective is to better understand the software process applied in GSE settings, notably settings using the Cloud as development environment. Based on real project data[1], a simulation-based approach was chosen to improve the understanding of such processes and to support project managers to select and tailor software processes for Cloud-based distributed software development. Hence, an objective of the presented work is also to show feasibility/reliability of using simulation models, e.g., for projects in the *Software Factory* environment. Finally, we aim at providing an instrument, which helps project managers comparing different solution approaches and to adapt current team processes to improve future project activities and outcomes.

Contribution. An event-driven simulator (Anderson et al., 2012) was configured using a Scrumban process with the number of user stories and their effort and priority in the backlog as input. The simulator helps

[1]For seven weeks, six developers in Finland and six in Spain, located at three sites (two in Spain and one in Finland) worked on a project developing a SmartGrid system. See Section 4.1 for further details.

reproducing activities, developers, user stories and events in the project, and it generates statistics, e.g., on throughput, total time, and lead and cycle time. The resulting simulation model can be customized to simulate different processes. Specifically, in addition to the Scrumban process, we also modeled "pure" Scrum and Kanban processes to allow for comparing the different processes with regard to project performance thus supporting project managers in selecting the best-fitting development approach for a specific scenario.

Outline. The remainder of the article is organized as follows: Section 2 provides an overview of related work. In Section 3, we describe the research design including research questions, simulation variables, and the specification and implementation of the simulation model. Section 4 presents the results from the different simulations. We conclude this article in Section 5.

2. Related Work

Software Processes and GSE. Globally distributed software development has become commodity, and it was showcased that distributed teams and even outsourced teams can be as productive as small collocated teams (Sutherland et al., 2007), which, however, requires a full implementation of Scrum along with good engineering practices. Paasivaara et al. (2009) state that agile methods can provide a competitive advantage by delivering early, simplifying communication and allowing the business to respond more quickly to the market by changing the software. To support this claim, authors present a multi-case study on the application of Scrum practices to three globally distributed projects discussing challenges and benefits. In this regard, Phalnikar et al. (2009) propose two team structures for implementing Scrum in a distributed setting. However, deploying agile methods to a GSE-setting is challenging for several reasons, such as demanding communication in a distributed setup, challenges related to coordination, and collaboration (Alzoubi et al., 2016; Vallon et al., 2017; Lous et al., 2017), and there is yet no agreement on generalizable solution approaches. For instance, while Vallon et al. (2017) discuss how agile practices can help improving or resolving such issues and found Scrum the most promising/successful development approach, Lous et al. (2017) found GSE challenging Scrum, especially when it comes to scaling the process in the context of (large) distributed settings. Wang et al. (2012) state that using agile methods helps mitigating challenges in co-located as well as in distributed teams, e.g., responding to fast-paced changes that occur in software projects. All the factors above influence the way in which software is defined, built, tested, and delivered. Ramesh et al. (2006) discuss how to integrate and balance agile and distributed development approaches to address such typical challenges in distributed development.

Complementing the "pure" agile approaches, *Lean* approaches have gained significance in the software industry, and they are used in co-located and distributed settings alike. Such approaches focus on eliminating waste, e.g., (Mujtaba et al., 2010), yet, these approaches are still under study, notably with regards to the question if and how these approaches help mitigating the various challenges in GSE. For instance, Tanner and Dauane (2017) study Kanban and highlight those elements that can help alleviating communication and collaboration issues in GSE. Kanban is a development approach, which applies Lean principles (Ahmad et al., 2013; Ikonen et al., 2011; Ahmad et al., 2016) and is becoming increasingly popular as an effective extension of Scrum and other agile methods. However, even though Kanban's popularity is increasing, many questions regarding its adoption in software development remain open. Practitioners face serious challenges while implementing Kanban, since clear definitions of its practices, principles, techniques, and tools are missing. In response, distributed teams use a plethora of specific tools to facilitate collaborative work Portillo-Rodríguez et al. (2012). However, different studies suggest the projects' processes being selected in a pragmatic rather than in a systematic manner (Vijayasarathy and Butler, 2016; Theocharis et al., 2015; Kuhrmann et al., 2017), and studies also suggest agile methods stepping into the background when it comes to define proper tool support (Femmer et al., 2014). On the other hand, GSE is a discipline that is maturing, as for instance Šmite et al. (2010) show in their discussion of available empirical evidence in the field or Ebert et al. (2016) who discuss the impact of GSE-related research to industry. That is, there is a variety of software processes and support tools used in practice. Such combinations are usually made in response to the respective project context (Kuhrmann et al., 2017), which gives project managers a hard time picking the most efficient process-tool combination for a project.

Software Process Simulation. Software Process Modeling Simulation (SPMS) is presented as a promising approach suitable to address various kinds of issues in software engineering (Kellner et al., 1999). Martin and Raffo (2001) present the simulation of a practically used software processes with the purpose of evaluating a potential process change to mitigate risks coming along with process change. Their model simulates discrete activities within the context of an environment described by a system dynamics model. A systematic review by Zhang et al. (2008) showed that especially risk management is one of the key objectives of SPMS. Liu et al. (2009) conducted a systematic review on risk management and SPMS concluding that the number of studies has been increasing gradually and that *discrete-event simulation* and *system dynamics* are the most popular simulation paradigms. For instance, examples for discrete-event simulations of agile practices are presented by (Melis et al., 2006; Turnu et al., 2006). Cao et al. (2010) present an approach based on system dynamics to study the complex interdependencies among the practices used in agile development. However, discrete-event models have to be considered critical as they use simple building blocks and tend to be fairly basic, and such models face problems concerning the discretization of time and insufficient detail of parameters and variables. An analysis of the dynamic behavior of a Scrum and Kanban variant has been conducted by Cocco et al. (2011). Turner et al. (2012) simulate the process performance of shared systems engineering services. They developed a specific Kanban-based process to support software development in rapid response environments, simulated this process using three modeling approaches (system dynamics, discrete events, and agents), and compared it to a simulated traditional process to determine if there were gains in effectiveness and value over time. Their overall goal was to study whether organizing projects as a Kanban-based scheduling system (KSS) leads to a better project performance. Tregubov and Lane (2015) presented a simulation model designed to explore effects of using KSS in multilevel systems. Their model implements a discrete-event simulation of the software-intensive system engineering processes for the purpose of estimating how KSS-scheduling can achieve predicted benefits, i.e., delivered value over time and schedule. Other than in the predictive simulation approach, Ali et al. (2015) use simulation as a tool to support reflections and discussions. They found simulations substantially contributing in identifying opportunities, e.g., reduction of idle times and improvement of the workflow in a process. Simulation was found beneficial in reasoning about and selection of alternative practices to steer process improvements. Finally, the suitability of software process simulation and an agenda for advancing reciprocity among research and industrial practice is presented by Houston (2012), who also shows the hurdles coming along with process simulation.

Software Process Simulation and GSE. Globally distributed projects that are conducted in an agile way can be characterized as human-intensive endeavors, yet, simulating humans and their behavior is difficult. However, empirically proven models for simulating complex behaviors exist, e.g., in the field of psychology. While modeling of human behavior is not in the scope of the presented work (and this should be considered when using the models), Armbrust et al. (2005) provide a discussion on human resource modeling in software development. Nevertheless, using process simulation for distributed projects is considered a promising route towards prediction and fast evaluation of process change, as several aspects can be analyzed quickly and without utilizing long-lasting thus expensive case studies or limited student lab experiments (Fagerholm et al., 2017). Although contributing to the body of knowledge, case studies describe context-specific approaches and, therefore, transferring the outcomes to another context usually requires setting up a new case study. A process simulation as presented in this article helps improving decision-making processes by constructing a parameterized simulation model, which allows for modeling the intended process (or a set of alternatives), feeding the simulation with (empirical) data from past projects, calibrate the simulation, and eventually conclude a feasible solution; a procedure that was, so far, successfully applied to other fields, e.g., risk management in distributed projects as presented by Lamersdorf et al. (2012).

The work presented in this article emerges from the various difficulties regarding the use of simulation models to reproduce real case studies from China and India (Concas et al., 2013; Anderson et al., 2012, 2011; Lunesu, 2013). This article thus contributes to the body of knowledge by presenting a simulation-based approach that can help reflecting on past projects and selecting and evaluating process alternatives to improve the GSE development approach.

Cloud-based Development and GSE. So far, in literature, few reports on using process simulation of agile methods in GSE using the Cloud as major development environment are available. Due to its *economies of scale*, Cloud computing has become the norm for consuming computing resources. While the potential for using the Cloud for GSE has been investigated in the literature, Alajrami et al. (2016) go one step further and propose a Cloud-based software process enactment architecture which utilizes the Cloud elasticity, accessibility and availability to facilitate distributed development, and to overcome some of the associated technical and communication challenges. Yara et al. (2009) present a Cloud-based platform that addresses core problems, e.g., computing capacity, bandwidth, storage, security, and outline a generic Cloud architecture and an initial implementation in the context of GSE. Nevertheless, even though companies have implemented GSE, they still face challenges in the development lifecycle. Hashmi et al. (2011) provide a synopsis of Cloud computing, GSE challenges, and discuss the problem that "Cloud" denotes a process *and* a product alike. Therefore, Hashmi et al. (2011) especially support our motivation to use Cloud technologies in GSE.

This article thus contributes to the body of knowledge by providing a study on the Cloud as development environment for GSE. Our study addresses the issues above by using a simulation-based research approach. Grounded in historical data, we provide a means to model distributed projects and simulating them in order to investigate the various challenges and effects coming along with using agile and Lean software development approaches in GSE.

Previously Published Material. The article at hand is an extended version of Lunesu et al. (2017) in which we compared three process simulations with the original process, Kanban, and Scrum to study the methods' impact on performance, total time, and throughput. In this extended article, we added a fourth research question (Table 1) to our previously published conference paper with which we extend our analysis by a comparison of the different processes. Accordingly, related work as well as the result presentation and discussion have been extended.

3. Research Design

This section presents the research design for the study. The development of the simulation model and the execution of the simulations followed the approach described in Rus et al. (2003) and Armbrust et al. (2005). The overall research objective and research questions studied are presented in Section 3.1. Section 3.2 describes the goals and requirements. The simulation model as key element of the study is specified in Section 3.3, and its implementation is presented in Section 3.4.

3.1. Research Questions

To study distributed software development in the Cloud and make a comparison among Lean Agile processes using an adapted simulation model, we formulate the research questions in Table 1.

Table 1: Summary of the research questions addressed in the study at hand.

	Research Question and Rationale
RQ_1	*How does the simulation model need to be calibrated, such that it reflects the particularities of the distributed project?* The first research question aims at extending a previously defined simulation model (Anderson et al., 2012), such that it covers the particularities of distributed software development. For this, different elements of the model need to be adjusted, and several simulation runs need to be performed to tune the model. For each simulation run, only a single parameter varies (e.g., average effort of each user story, project size, and team size). Finally, the total time required and throughput values are examined to understand whether the variations are continuous or non-linear. For this, the following metrics are used: `throughput` and `total_time`, for a chosen value of one parameter and for fixed values of the other inputs, the simulator is run once until it stops (the end of the simulation) and the variation of the throughput (and total time) is examined.
RQ_2	*To what extent can the simulation model reproduce the data obtained in the real project?* Having the calibrated simulation model available, the second research question aims to study whether the simulation model can be used to reproduce the real project. In particular, results (i.e., `throughput` and `total_time`) are collected feeding the simulation model with artificial and real project data. Results are used to improve the simulation model and, eventually, a comparison is carried out using the metric `distance` (between curves) of released user stories.
RQ_3	*How reliable is the simulation model?* The third research question studies the reliability of the simulation model. In particular, if many simulation runs are performed using the same inputs: Does the model behave as expected? For this, several runs of the simulation model are performed using a list of artificial user stories. As metric, the `variation` (of average effort) is used to compare the variation in the calculated effort with the real effort from the project data.
RQ_4	*Can a comparison of Scrumban, Kanban, and Scrum processes performance support decision-making?* The fourth research question studies the adaptability of the simulation model for reproducing Scrum and Kanban processes in order to compare them. For this, several runs of the simulation model are performed using input data collected from the *Software Factory* project. As metrics, the `average`, `median`, `min`, `max`, and the standard deviation of `cycle_time` are used to compare the performance of the three development processes.

3.2. Simulation Goals and Requirements

The overall goal of this study is to better understand distributed software development in a Cloud context. For this, an existing simulation model (Anderson et al., 2012) is modified to better support decision-making processes concerning planning a distributed development process. The aim of this simulation model is thus to analyze the tradeoffs regarding throughput and total time on varying project size, team size, WIP limits and average effort. Furthermore, the modified simulation model aims to help project managers selecting the most suitable planning alternative. The overall simulation goals setting the scene for the simulation are therefore in Table 2 described using the GQM goal template according to Solingen and Berghout (1999):

The simulation model is purposed to answer the detailed questions collected in Table 3. For this, we define the input and output parameters/variables as summarized in Table 4. The simulation is performed instrumenting five scenarios, which are defined in Table 5.

Table 2: Summary of the simulation goals and context using the GQM goal template.

Object	Simulation model of a distributed development process
Purpose	Support decisions for planning
Quality Focus	Throughput, total time, cycle time, size of the project, and size of the team
View Point	Project Manager
Context	Software Factory Network

Table 3: Simulation-specific questions.

	Question
Q_1	If the throughput is fixed, how can the other parameters be adjusted?
Q_2	If the project size varies, but other parameters remain fixed, what is the effect on the throughput and on the total time required?
Q_3	If the team size varies, but other parameters remain fixed, what is the effect on the throughput and on the total time required?
Q_4	If the work-in-progress limit (i.e., the maximum number of user stories that can be handled at any given time) varies for different activities, how does the throughput change?
Q_5	What is the relationship between the average effort for the user stories in the project and the total time required?
Q_6	Which parameters can be used to best compare process performance?

Table 4: Simulation input and output parameters and variables.

Input		**Output**	
I_1	Project size (total number of user stories) at time t, it is denoted by $N_F(t)$	O_1	Throughput
I_2	Team size (number of developers), it is denoted by N_D	O_2	Total time
I_3	Average effort	O_3	Duration of simulation T
I_4	Number of activities, it is denoted by N_A	O_4	Cycle time for a user story[a]
I_5	WIP Limits in each activity (the maximum number of user stories that can be handled at any given time), it is denoted by M_k for the k^{th} activity		

[a] Time required to complete a user story is collected and computed using actual time, mean, median, and standard deviation.

3.3. *Specification of the Simulation Model*

In this section, we briefly introduce the *Software Factory* process, which serves as a blueprint for distributed development projects, and we analyze and explain the modifications required to use this process as input for the simulation model.

3.3.1. *The Software Factory Process Model*

In the *Software Factory* (Fagerholm et al., 2013), Scrumban (Kniberg and Skarin, 2010) is used to run the distributed software development projects. In general, a coach combined an agile process (Scrum) with a Kanban board, which visualizes the user story assignment in each process step.

The Scrumban model as shown in Figure 1 comprises the four steps *Pregame*, *Sprint Planning*, *Sprint*, and *Review Meeting*. In the reported setting, a single sprint takes two weeks. Apart from this, most of the well-known Scrum practices are applied, e.g., the *product owner* selects *user stories*, *developers* estimate the given stories, and *daily stand-up meetings* are performed. To set up the simulation, we provide a for-

Table 5: Simulation scenarios.

Scenario	Description
S_1	For a chosen value of the throughput or total time and for fixed values of the other inputs (project and team size), the simulator is run once until it stops and the total time required is examined.
S_2	For a chosen value of the size of the project and for fixed values of the other inputs, the simulator is run once until it stops (the size of the project is reached) and throughput and total time are examined.
S_3	The simulator is run for a chosen value of size of the team, and for the fixed values of the other inputs, the values of the throughput and total time are examined.
S_4	For a chosen value of the WIP limits in each activity and for the fixed values of the other inputs, the values of the throughput and total time are examined.
S_5	For a chosen number of simulation runs, and all parameters remain fixed, and the relation among average effort and total time is examined.
S_6	For a chosen number of simulation runs, and all parameters remain fixed, the comparison of cycle time statistics of three different processes are examined.

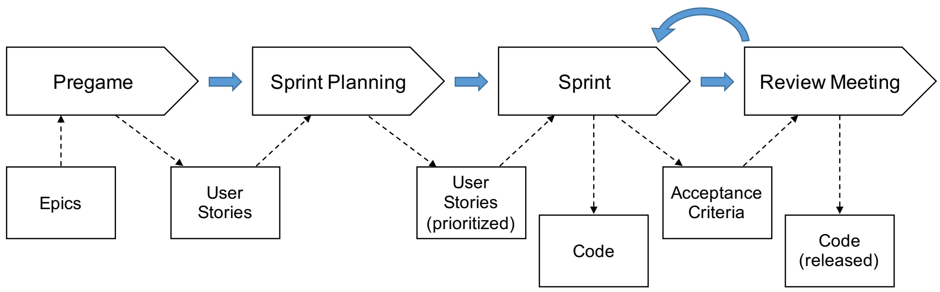

Figure 1: Overview of the Scrumban process as used in the Software Factory. This overview illustrates the main steps in the process and the incoming/outgoing artifacts per process step. The thick opaque arrows show the control flow, and the dotted arrows show the product flow.

malization of the process model from Figure 1. Therefore, we need a detailed understanding of the process model and how specific practices are implemented. Table 6 provides a detailed description of the process steps and assigns inputs and outputs.

According to the general Scrum guideline (Schwaber and Beedle, 2002; Kniberg and Skarin, 2010), the three roles *Scrum master*, *product owner*, and *team* are present in a software project. In the *Software Factory*, these roles are generally present and implemented. However, due to the distributed project setup, the team is spread across three project sites (one team per site). That is, the project is operated as a distributed project and, thus, the team faces several challenges of distributed projects (Lous et al., 2017), such as time loss due to long meetings caused by an inefficient Internet connection, due to the problems with communication tools, due to the dependencies among different user stories, and allocation of work among different sites at which the team members are located.

The *Software Factory* was used for on-site observations to collect information for modeling the project context of our simulation model appropriately. After each iteration, interviews have been conducted with the development team members. Furthermore, we were involved in the daily meetings and the sprint review meetings to collect extra data for improving the simulation model. For instance, the different teams were composed of practitioners and graduate students each with different skills and work experience. Information about the team members has been used to calibrate the simulation model.

Table 6: Detailed description of the different process elements considered in the process simulation. Implementation of actual practices in the Software Factory are explained.

Process Activity	**Input**	**Output**
In the *Pregame*, *Epics* as input are divided into *User Stories*. The outcome of this meeting is the (initial) Backlog containing all *User Stories* to be prioritized in the *Sprint Planning* activity. In the Software Factory, the *Pregame* usually takes two days.	Epics	User Stories
Based on the Backlog, in the *Sprint Planning*, each *User Story* or task (in which some user stories are divided) is prioritized.	User Stories	User Stories (prioritized)
In the *Sprint*, the actual development activities (including analysis and coding tasks performed by the developers) are carried out. During the *Sprint*, daily meetings (10-15 minutes) are held in which the four basic Daily Scrum questions are asked and answered. Eventually, this activity produces the actual systems, i.e., the *Code* of the system, and a set of *Acceptance Criteria* (according to a "Definition of Done"; DoD), which are used in later analyses of the goal achievement.	User Stories (prioritized)	Code, Acceptance Criteria
In the *Review Meeting*, the team and the Product Owner verify the fulfillment of the *Acceptance Criteria* defined in the analysis steps of the Sprint. The product-centered *Review Meeting* is complemented by a more process-oriented retrospective. Depending on the review outcomes, some tasks might be subject to rework, i.e., certain tasks might be repeated, and those tasks are scheduled for the next *Sprint*. Tasks that are considered done eventually result in released *Code*. In the Software Factory, a *Review Meeting* takes about one hour.	Acceptance Criteria	Code (released)

3.3.2. General Adaptation of the Simulation Model

The presented simulation model is grounded in a previously developed model by Anderson et al. (2012) for which the *Software Factory* process served as calibration model. The underlying simulation model was used to reproduce the originally used PSP/TSP (Humphrey, 2000a,b), Scrum, and Lean-Kanban processes by describing process elements such as features, activities, and developers. We analyzed the practical application of the *Software Factory* process and compared it to the original simulation model to determine those parameters to be used for calibration. In particular, multi-site development and the resulting challenges for collaboration and communication had to be implemented in the simulation model. Specifically, the following changes have been made to the original simulation model to adequately reflect the *Software Factory* process:

- The *Pregame* activity was added to the simulation model.
- *Rework* was added to the simulation model.
- The simulation model was modified to better reflect productivity in distributed settings.

The implementation of *rework* in the simulation model allows for repeating those tasks that are not yet finished or that do not fulfill the *acceptance criteria*. Such tasks are scheduled for the next *Sprint* and continue previous activities (from *review meeting* to *Sprint*). The productivity-related modification was performed to better reflect the productivity in terms of the number of hours worked (per developer) and changes of the team size in different phases of the project. For instance, the modification covers changing team setups, such as on-boarding a team, e.g., the core team consists of six developers (begin, end) and in selected phases, another six developers joint the team.

3.4. Implementation of the Simulation Model

Figure 2 shows the final implementation of the simulation model as a UML class diagram, which shows the entities of the system and the relationships between the different actors. The classes `KanbanSimulator` and `KanbanSystem` represent the simulator's core system comprising all methods to create the simulation environment. The remaining classes, e.g., `user story`, `Activity`, and `Developer`, reflect the process model entities to be simulated. The entity classes are complemented with some utility classes, e.g., `ActivityRecorder`, that help recording data for the simulation analysis. This way of implementing the simulation models follows a hybrid approach in which discrete-event and agent-based simulation approaches are combined. The discrete-even simulation part is used to simulate the high-level tasks and the accumulation of value, whereas the agent-based simulation part is used to model the workflow at a lower level, i.e., working teams, Kanban boards, work items, and activities. A more detailed explanation of the (original and unadjusted version of the) simulation model can be found in Anderson et al. (2011).

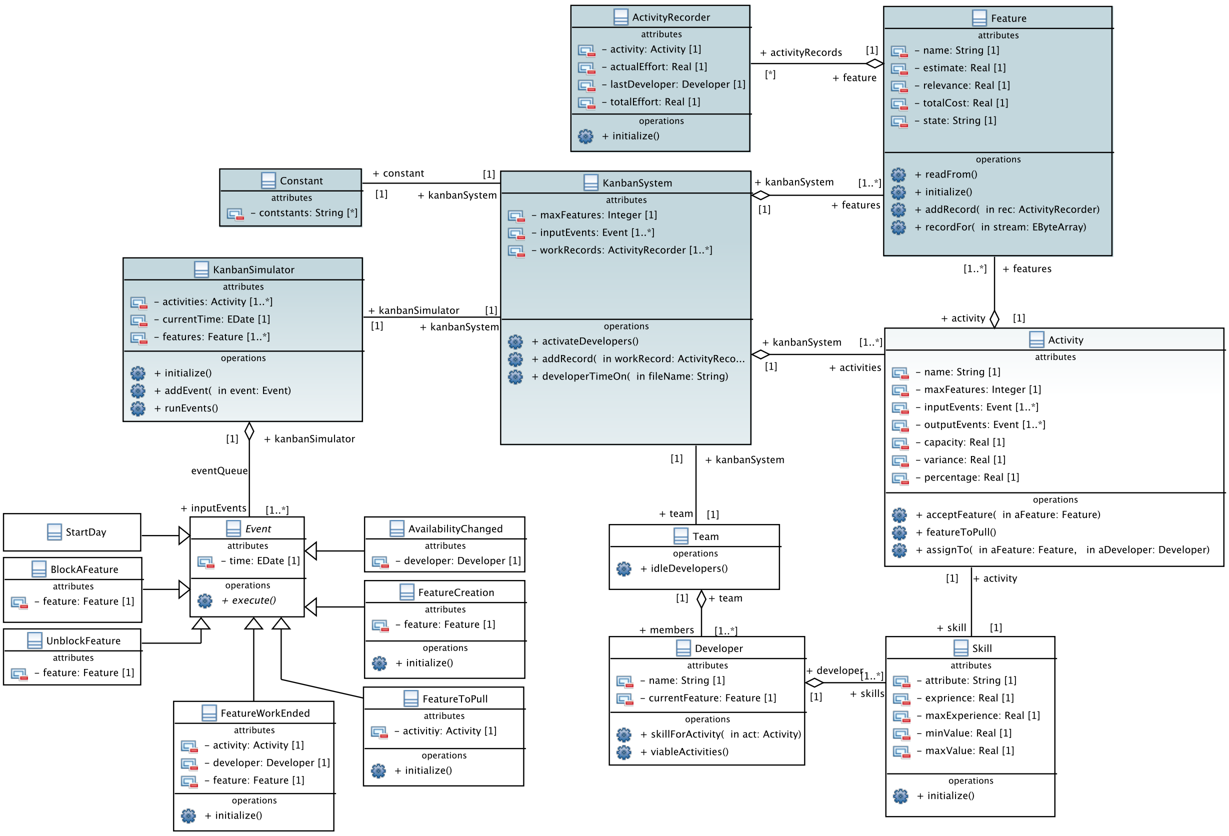

Figure 2: UML class diagram of the simulation model.

In the simulation presented in the paper at hand, the main actors are the developers of a distributed team working according to the process as shown in Figure 1, whereas each activity requires a certain set of skills. The most important events in the simulation model are: `FeatureCreation`, `FeatureToPull`, `StartDay`, and `FeatureWorkEnded`. These are used to set the scene for a simulation and to analyze the (potential) need for rework.

To run a simulation using the presented model, the following input is required: The main input is a list of user stories of which each is characterized by an identifier, a report date, an effort characterizing the amount of work required to complete a user story (in days), and a priority (as a numerical value; the higher the value the higher the user story's priority). Furthermore, a set of parameters related to the real process data, such as number of developers, developer skills, probability of rework, and work-in-progress (WIP) limits is required. Finally, a script initializes the process (the process variables), e.g., duration of meetings or sprint length. The script also runs the simulation, collects, and stores data to CSV files. The actual technical implementation of the project environment and, accordingly, the infrastructure used to realize the simulation model, which is implemented in Smalltalk, follows the infrastructure setup described in detail by Fagerholm et al. (2013).

3.4.1. Modification of the Simulation Model for Scrum and Kanban

In addition to the *Software Factory* process above, we included two more processes in our study: Kanban and Scrum. Both adaptations of the simulation are explained in the following:

Modification for Simulating Scrum. Scrum is characterized by iterations (so-called *sprints*) of a maximum 30 work days. Each sprint starts with an iteration planning meeting and ends with a *retrospective*. The length of the two meetings should not exceed one day. In a sprint, a *daily sprint* meeting is held every day. If one or more user stories from the *sprint backlog* are not finished in a sprint, they are moved to the next sprint. The completed user stories are released at the end of the sprint (so-called *potentially shippable product*). In the context of our simulation, we considered the similarities of Scrum and Scrumban. Yet, we ignored the *pregame* phase and we assumed an already completed sprint backlog containing estimated user stories. Likewise, we considered the implementation of *rework* in the simulation model that allows for repeating those tasks that are not yet finished or that do not fulfill the *acceptance criteria*. Such tasks are scheduled for the next *sprint* and continue previous activities (from *review meeting* to *sprint*). The productivity-related modification was performed to better reflect the productivity in terms of the number of hours worked (per developer) and changes of the team size in different phases of the project. For instance, the modification covers changing team setups, such as on-boarding a team, e.g., the core team consists of six developers (begin, end) and in selected phases, another six developers joint the team. Also the duration of the sprint has been adapted, since Scrum does not define a *pregame* phase and WIP-limits as used for a Kanban board.

Modification for Simulating Kanban. For simulating Kanban, we also assumed a completed backlog. The Kanban workflow has been modeled for the simulation as follows: in the first activity (*analysis*), estimated and prioritized activities are analyzed and pulled from the second activity (*implementation*), which happens respecting the WIP-limits set and the skills of available developers. Once the implementation is done, user stories are pulled from the third activity (*test*) to evaluate the quality according to the acceptance criteria set. Finally, completed user stories are either pulled from the *deployment* activity or sent back to the analysis phase in case rework is necessary. Same as in the Scrum model, we also implemented *rework* for Kanban thus allowing for repeating those tasks that have not been finished or failed the testing phase, and we provide WIP limits concerning the size of the team and activities such as: analysis, implementation and testing and deployment. The productivity-related modification was performed to better reflect the productivity in terms of the number of hours worked (per developer) and changes of the team size in different phases of the project (see adaptation of the simulation model for Scrum above).

4. Simulation Results

In this section, we present the simulation results. In Section 4.1, we describe the actual simulation setup. In Section 4.2, we present the outcomes of the simulation runs and a discussion. Finally, in Section 4.3, we critically discuss our findings regarding the threats to validity.

4.1. Simulation Setup

We observed a project[2] from April 23, 2012 till July 6, 2012 in which a team of six developers (divided into two groups) started working for 3 h/d in Spain. From May 14, 2012 until June 29, 2012, another team of six developers located in Helsinki joined the project and worked for 6 h/d. In these periods, we monitored the processes implemented and collected the raw project data, which has been analyzed and used to create the input for the simulation model. To better reproduce rework on interconnected tasks, we have reduced the 64 user stories (considering user stories and tasks, in which some user stories have been divided,) to 25 user stories. The throughput in the real project was almost three user stories per week with an average effort of 1.3 to 1.5 person days. Eventually, for the initial setup, we considered 25 user stories and tasks stored in the backlog, whereas we expect new user stories coming in after the last review meeting of an iteration, or at the beginning of a new iteration. Furthermore, we assume developers always available to work on and release upcoming user stories.

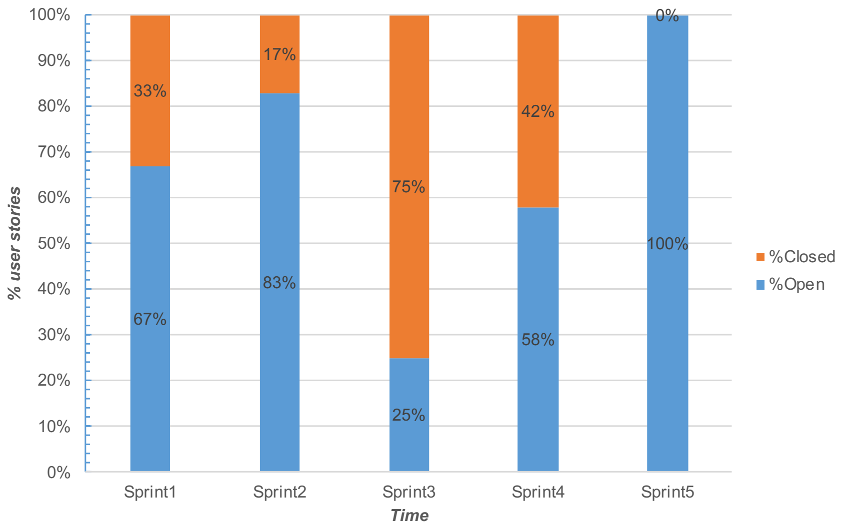

Figure 3: Percentage of open and closed user stories as used for the different simulated sprints.

In this simulation we analyzed five sprints, and we performed simulation runs using real and artificial data. Real data has been collected directly from the aforementioned project, and artificial data has been collected by using an algorithm of the simulation model that takes the real data as input. After the data analysis, we calculated the average effort and standard deviation to identify the data distribution[3] and to obtain statistical values for incoming user stories. Having the data required, we built the list of user stories that serve as input for the simulator (Figure 3 shows the resulting user story setup used for the simulation).

[2]In this project in the context of a smart grid environment, the system to be implemented had to process and analyze a substantial quantity of data concerned with measurement of data consumption. Data was collected hourly, daily, and monthly. The teams were appointed to implement the different modules that compose the system for the processing data.

[3]For the data distribution, we assume a log-Normal distribution. For incoming user stories, however, the distribution is unknown. Therefore and in order to allow for replicating input data in different simulation runs, we use a linear interpolation method.

4.2. Simulation Runs

In this section, we provide insights into the simulations and present and discuss the results. The different simulations address the questions (Table 3) and scenarios (Table 5) as introduced in Section 3.2. The full mapping of simulations, questions, and scenarios is shown in Table 7. For example, question Q_1 is studied using the first simulation for scenario S_1, i.e., for a given throughput, what parameters can be varied. Similarly, simulation two for scenario S_2 helps answering Q_2 and Q_5, i.e., how total time and throughput vary in relation to varying project size. In the following, we first describe the individual simulations before integrating the different outcomes for answering the research questions in Section 4.2.8.

Table 7: Mapping of simulations to questions (Q) and scenarios (S).

Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Simulation	S_1	S_2	S_3	S_4	S_5	S_6
✗						1	✗					
	✗			✗		2		✗				
		✗				3			✗			
			✗			4				✗		
				✗		5					✗	
						6					✗	
					✗	7						✗

4.2.1. Simulation 1

The first simulation addresses Q_1 and studies variables that can be modified—and how they can be modified—if the variables `team_size` or `project_size` are fixed. In particular, the variables `throughput` and `total_time` are of interest, and how they can be modified. For S_1, the throughput is set and time required to complete the simulation is examined.

In the chosen team setup (12 developers, *Group 1* has six developers working 6 h/d and *Group 2* has six developers working 3 h/d), a project of 25 user stories, each with an effort of 1.3–1.5 person days, was chosen. The overall performance was five to six user stories per day and, eventually, the team could work on about 500 user stories without inflationary growth of the backlog. However, communication issues and dependencies among user stories and/or tasks limited the performance, such that S_1 yielded in an average throughput of only three user stories per day.

4.2.2. Simulation 2

The second simulation studies Q_2, i.e., studying what effect a varying project size has (i.e., keeping the other parameters fixed) on the throughput and the total time required for a project. For a chosen value of `project_size` and fixed values of other inputs, the simulator is run once until it stops (the size of the project is reached) and the values of the variables `throughput` and `total_time` are evaluated. We assumed that a linear relation among `project_size`, and `throughput` and `total_time` required exists. Therefore, we re-ran the simulation with a stepwise increasing `project_size`, but kept the other parameters fixed. The number of user stories (with an average effort of 1.3 person days) increases and we checked the differences in `throughput` and `total_time` as shown in Figure 4.

For a doubled project size the throughput increases in linearly, yet shows a little steep when the size of the project increases from 200 to 400 user stories; and then continues with a linear trend until 500 user stories[4]. Regarding the total time required, the trend is almost linear with slow rise and a steep when the

[4]Which is almost the maximum number of user stories the team can work on without an exceeding growth of the backlog.

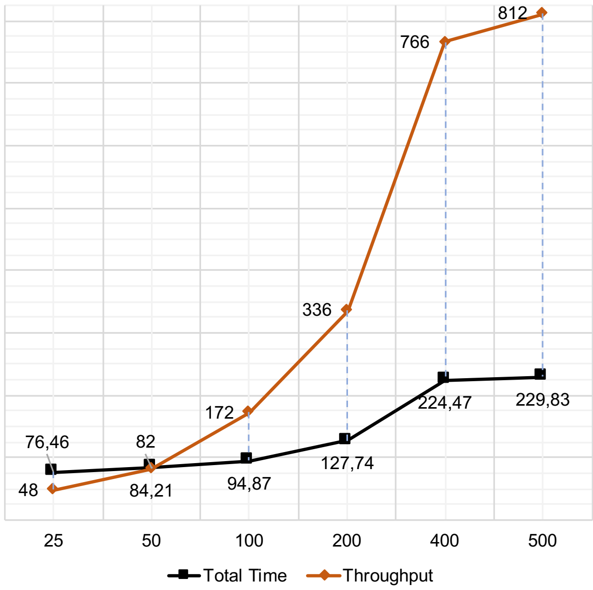

Figure 4: Relation of total time (in days) and throughput (user stories released) of the simulated projects with different amounts of user stories (see Table 8).

projects size increases from 200 to 400 user stories—and then continues with a linear trend until 500 user stories. In Table 8 we tabulated the obtained throughput as the total user stories released during the project. We consider the `throughput` as the number of user stories released at the end of the project. Table 8 shows that for a project size of 25 user stories, the throughput is approx. three to four user stories per week (48 user stories, including assumed 20% of rework) and a `total_time` of 76.46. For a project size of 50 user stories, the `throughput` is seven to eight user stories per week, i.e., the `throughput` equals 82 closed user stories and a `total_time` of 84.21, and so forth. Hence, doubling the project size also doubles the throughput.

Table 8: Variation in the size of the project and impact on throughput and total time required.

Size	Total Time	Throughput
25	76.46	48
50	84.21	82
100	94.87	172
200	127.74	336
400	224.47	766
500	229.83	812

4.2.3. Simulation 3

In S_3, we study the relationship between `team_size`, `throughput` and `total_time` to answer Q_3, i.e., what the effect on the throughput and the total time required is if the team size varies, but other parameters are fixed. We assume that no linear relation exists among `team_size`, `throughput`, and `total_time`. That is, if the number of developers skilled in testing goes to zero, throughput is blocked. If the number of blocked user stories grows, adding new tester does not increase the throughput, due to the bottleneck from the previous phase. The summary of the simulation results is shown in Table 9 in which the `throughput` is again represented by the number of user stories released by the end of the project.

Table 9: Team size and throughput (project performances in relation to different team size and skill profiles; skills for activities: 1=analysis, 2=development, 3=testing, and 4=deployment).

Team Size	Skills 1	2	3	4	Total Time in days	Backlog	Pregame	Sprint Planning	Sprint	Review Meeting	# of Released User Stories
All		–			83.923	25	25	25	33	30	42
6		✗	✗	✗	85.0125	25	25	25	32	28	34
6		✗	✗	✗	98.0263	25	25	25	30	29	37
6		✗	✗	✗	98.0263	25	25	25	30	29	37
6		✗	✗	✗	84.3339	25	25	25	39	34	52
6	✗	✗	✗		76.7853	25	25	25	26	20	0
6		✗	✗		77.4417	25	25	25	30	19	0
6		✗	✗		76.7853	25	25	25	26	20	0
6		✗	✗		96.651	25	25	25	41	31	45
6	✗	✗	✗	✗	96.0761	25	25	25	33	28	38
6		✗	✗	✗	96.0761	25	25	25	33	28	38
6		✗	✗	✗	99.2204	25	25	25	33	27	31
6	✗	✗	✗	✗	86.5942	25	25	25	37	32	46
6		✗	✗	✗	89.6905	25	25	25	36	32	46
3	✗	✗	✗	✗	98.5971	25	25	25	39	34	52
3		✗	✗	✗	98.5971	25	25	25	39	34	52
3		✗	✗	✗	98.5971	25	25	25	39	34	52
3		✗	✗		125.745	25	25	25	38	32	46
3		✗	✗	✗	79.3474	25	25	25	26	22	0
3		✗	✗	✗	105.672	25	25	25	39	31	43
3		✗	✗	✗	104.795	25	25	25	35	30	40
3			✗	✗	77.4448	25	25	25	33	22	0
3		✗	✗	✗	78.2208	25	25	25	32	22	0
3		✗	✗	✗	84.6146	25	25	25	37	29	39
3		✗	✗	✗	95.5282	25	25	25	34	29	39
3			✗	✗	105.672	25	25	25	39	31	43
3	✗	✗	✗	✗	84.7019	25	25	25	32	28	34
3	✗	✗	✗	✗	84.7019	25	25	25	32	28	34
3		✗	✗	✗	103.79	25	25	25	36	32	16

The team size was chosen, in order to study the impact on the throughput when other project parameters remained fixed. For this, we use the number of hours that each developer works. We assume that variations on `throughput` and `total_time` depend on the developers' skills, on the number of hours they work, and on the strategy used to assign them to the activities rather than the size of the team. We selected the cases of the whole team, and teams with six and three developers respectively. The results demonstrated that variations in `throughput` and `total_time` mostly depend on the skills of the developers and their assignment to the different activities. Table 9 shows that if three developers or six developers, that are skilled in all activities, work on the same number of user stories, they may obtain the same `throughput` and the same `total_time`. Instead, when the number of developers is not high enough to satisfy the effort required for an activity, `throughput` decreases and the `total_time` increases.

4.2.4. *Simulation 4*

In S_4, we study the relation between the use of WIP limits, `throughput` and `total_time` to answer Q_4, i.e., what the impact on the throughput is if the WIP limit for activities varies. For a given WIP limit, it is possible to examine the resulting throughput and total_time, if other parameters remain fixed. It is required to perform many simulation runs to obtain WIP-limit values, which can yield optimal throughput in the minimum time required. For a team setup of 12 developers, we performed several simulation runs with different values for the WIP limit, and without limits. For example, at first one may consider a WIP limit of *10–12*, i.e., 10 in the first and last activity, and 12 in the second and third activity. WIP limits tested were also *6–8* and *3–4*. We observed that for lower WIP-limit values, throughput decreases and the total_time increases—a bottleneck may exist. However, if we consider WIP limits of six to eight or higher, results are the same as if there were no limits at all. This could be a result from the small number of user stories or the big team size and, thus, WIP limits are not useful (this also hampers generalizability). In a nutshell, for low WIP limits, we obtained a low throughput and a longer total_time, yet, higher WIP limits have not shown any effect on the throughput.

4.2.5. Simulation 5

In simulation S_5, we study the relation between average_effort, throughput, and total_time to answer Q_5, i.e., whether there is a relation between effort for user stories and the total time required for the project. For a given number of simulation runs, all simulation parameters remain fixed. At the end of each simulation, values for average_effort, throughput, and total_time are examined to understand if variations, as expected, are continuous. Accordingly, two experiments have been conducted: one with real data, and another using artificial data. The results obtained show that variations in the effort cause variations in throughput and total_time. Furthermore, the relation is almost continuous without major gaps. Performing many simulation runs, we found a low correlation between variations in average_effort and total_time.

Table 10: Correlation of effort and total time.

Variable	**Average**	**Standard Deviation**
Total Time	77.26	2.62
Effort	1.297	0.203
Corr (effort/time)	0.0888	

Table 11: Correlation of effort and throughput.

Variable	**Average**	**Standard Deviation**
Throughput	44.19	6.02
Effort	1.297	0.203
Corr (effort/throughput)	-0.119	

Simulations performed using the real project data, did not show any variation for neither variable. Yet, simulations using the artificial data showed variations. In total, we performed 100 simulation runs and found a low correlation 0.0888 between the variation in average_effort and total_time (Table 10). Furthermore, we found a correlation of -0.119 between average_effort and throughput (Table 11). Hence, there is no direct relation between average_effort and throughput.

4.2.6. Simulation 6

In S_6, we study the relations between average_effort, throughput, and total_time with a particular focus on the question to what extent the simulation model can reproduce data from the real project.

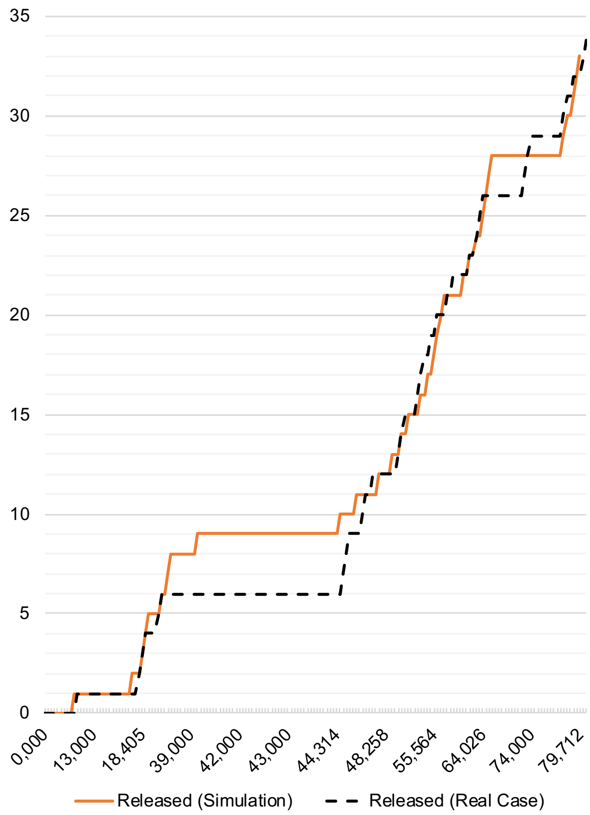

Figure 5: Comparison between simulated and real case with the number of user stories released (y-axis) and the time required to release the user stories (x-axis).

The implemented software development model presented in Section 3.3.1 is used to allow for comparing the results (i.e., `throughput` and `total_time` required to finish the project) obtained from the simulations performed on real and artificial data. In particular, simulations were run using the list of user stories, parametrized with values for the effort taken from the real project. The analysis was carried out on the number of released user stories, in particular by comparing the two performance curves shown in Figure 5. The curves represent the cumulative number of user stories released in the project and, in an optimal case, both curves should overlap. As Figure 5 shows, our experimental results suggest that the presented simulation model produces data that well match, which demonstrates the feasibility of the approach presented.

4.2.7. Simulation 7

In last simulation S_7, we compare the `cycle_time` of the three different processes Scrumban (the original *Software Factory* process), Scrum, and Kanban to improve our ability to choose the right process for the respective context and to adapt other processes in similar cases. Again, we study in how far our simulation model can also reproduce data from the real project.

The *Software Factory's* Scrumban model (see Section 3.3.1) and the adaptations of our simulation model for Scrum and Kanban (see Section 3.4.1) are to compare the performance of the different processes, specifically the `cycle_time`. Furthermore, data is used to compare the simulation outcomes with results obtained in the real projects. In this simulation, the different runs used the list of estimated and parametrized user stories from the real case project. Analyses have been performed on the real case data as well as on the

Table 12: Summary of cycle time statistics of the Scrumban, Kanban, and Scrum processes in the real cases compared with the simulation results.

Process	Real Case					Simulated Case				
	Average	Median	Min	Max	St.Dev.	Average	Median	Min	Max	St.Dev.
Scrumban	7.58	6.82	1.06	17.46	5.46	7.34	4.93	1.80	19.45	4.79
Kanban	6.65	5.70	1.58	18.48	4.80	6.28	4.33	1.17	20.56	4.93
Scrum	8.42	6.21	1.23	25.70	6.99	7.36	5.15	2.44	26.21	5.45

simulation results collected from 100 runs for each case. Table 12 shows the results, which suggest that our simulation model produces data that well match the real cases. Hence, we conclude that our simulation model satisfactorily reproduces the real case.

Comparing the three different processes, we see that the results related to each process are very close, in particular Scrum and Scrumban. Yet, our data suggests that—in the current distributed context—Kanban seems to be more efficient. A reason can be the more "sequential" nature of Kanban and its strong focus to limit work-in-progress, i.e., an attempt to improve the effective work assignment. This effect can be observed in the real case and the simulated case alike.

4.2.8. Summary of the Simulation Results

In this section, we briefly summarize our simulation results and answer the research questions (see Section 3.1). To support answering the research questions, in Table 7, we relate the different simulation scenarios shown in Table 5 with the detailed simulation questions shown in Table 3.

Research Question 1. To answer the first research question, we use the simulations S_{1-5}. The different outcomes presented in the previous paragraphs show the relationships between the three variables `throughput`, `total_time`, `average_effort`. The findings further show how the original simulation model by Anderson et al. (2012) can be calibrated in order to reproduce distributed software development (processes). In particular, the simulation for scenario S_1 showed that for a given throughput, `total_time` is the only parameter that can change (gives all other variables are immutable). The simulation for scenario S_2 showed a linear relationship between `throughput` and `total_time` for a varying `project_size`, whereas the simulation for scenario S_3 found no linear relationship if `team_size` is the subject of study. The simulation for scenario S_4 studied WIP limits and the impact on `throughput`, finding no effect on the throughput for higher WIP limits.

Research Question 2. The second research question aims at comparing simulation results with real project data (and experience). For this, the simulation for scenario S_6 is used. The results are shown in Figure 5, which shows the distance of the two curves as a measure of accuracy. In summary, the adapted simulation model was found feasible to reproduce a real project.

Research Question 3. The third research question aims to study the reliability of the simulation model. For this, the simulation for scenario S_5 was used, and the simulation was run several 100 times. The outcomes show the simulation model reliably reproducing results with acceptable variations for `throughput`, `total_time`, and `average_effort` regardless of the input data, i.e., real or artificial data.

Research Question 4. The fourth research question aims at comparing simulation results from three different processes to support project managers in selecting the project-specific development approach. For this, the simulation seven was used, and the simulation was run several 100 times. The outcomes show the simulation model reliably reproducing results (in our case for the `cycle_time`) from a real case and, thus, providing a means to ground decisions in the simulation results.

4.3. Threats to Validity

In the following, we discuss the threats to validity to be considered when applying the method presented in the paper at hand.

Internal Validity. According to Shadish et al. (2001), an experiment may have unknown and hidden factors that could affect the results. In the presented case, information regarding teams and organization of work originated from the projects. Data used in the simulation model was extracted from systems used by the teams and personal observations, which might influence result quality. Although the model properly simulates skilled developers performing task sequences, still, the simulation model does not fully cover interactions among the developers thus introducing a threat regarding the inclusion of human factors in the simulation.

Construct Validity. Construct validity concerns the degree to which inferences are warranted from the observed phenomena to the constructs that these instances might represent (Wohlin et al., 2012). A first threat to construct validity is that, although in this study we have carefully analyzed and preprocessed the *Software Factory* data, our results could be affected by the data quality (such as possible noisy data). Another threat related to construct validity is the fact that our work is centered on the study of how the process determines the efficiency of the development activity. However, there are many other human-related factors that could affect the efficiency and productivity of the team, e.g., considering (co-)workers, keeping the team motivated and satisfied, and so on. Just limiting the work-in-progress will not be effective if a team is troubled and dissatisfied. A simulation model can simulate a process, but it is very difficult to explicitly include human factors. To mitigate this threat, data about the Software Factory process (e.g., user stories, effort, and WIP limits) was collected daily by external researchers. Furthermore, at the end of the Software Factory projects, one researcher extracted data from the different tools used in projects, e.g., documentation and code, and interviews with the team members have been performed.

External Validity. If a study possesses external validity, its results will generalize to a larger population not considered in the experiment (Shadish et al., 2001; Wohlin et al., 2012). In this study, we only ran the simulation model on one development project. This project is small, and the number of subjects used in this study is small. This is a clear threat to external validity of our results. However, the simulation methods we proposed are evaluated on large software systems that experienced a long evolution. Furthermore, we extended our simulation in terms of modifying the simulation model to represent further development processes for a comparative study. Since these extra simulations confirmed the study of the Software Factory process, we assume a generalizability of the general simulation model. However, further studies need to be conducted to also confirm the project-related findings and whether these findings can be generalized.

Reliability. The main threat to the reliability of the simulation model and the input data is that only one researcher performed the observation, data collection and initial data analysis. To mitigate this threat, researcher triangulation was implemented for quality assurance of the different procedures applied and the data collected. To improve the data basis for developing the simulation model, in a first step, the data collected from the Software Factory projects was pre-processed by one researcher. During the data collection and the pre-processing phase, researchers and project team members established a continuous communication and result analysis to reduce the risk of misinterpreting (tentative) results. In a second step, using a linear regression algorithm, an artificial list of user stories was created from the actual project data, which allows for testing the reliability of the dataset in the simulation model.

5. Conclusion

In this paper, we presented a simulation process model able to reproduce the process followed in the *Software Factory* project. We demonstrated the calibration of the simulation model and its implementation. An existing simulation model was modified to reflect the Scrumban process as used in the *Software Factory*. We described the customization of the relevant parameters and aspects to implement the *Software Factory* process. Eventually, we performed a case study with (real-life) data gathered from *Software Factory* project.

Summary of Findings. The simulation results in the following major findings: Project teams face problems regarding communication and organization of distributed projects affecting the teams' productivity and/or increasing the time required to achieve the project goals. The results obtained from our simulation show the influence of decisions in the project planning activities, e.g., in assigning work, when a distributed development is considered for a project. Therefore, our simulation model can be used to model project setups of interest, to elaborate potential pitfalls, and to work out solutions to address those problems. This opportunity was especially shown by a comparative analysis of a simulated case and a real case. We could successfully model and reproduce the Scrumban process as used in the *Software Factory*, and our simulation generated results comparable to the real project data. Hence, the simulation model allows for modeling a distributed project, analyzing and predicting trends, and eventually selecting the most promising (according to the respective project goals) project configuration.

The key advantage of using a simulation is that various project parameters can be evaluated quickly and relatively easy to support the project management in selecting the most promising process alternative to positively influence the project performance. In our previous work, we could also show that project managers could improve their knowledge about the issues critical to the project and, thus, adapt the process for next iteration or for future projects. Hence, project managers get a tool to early analyze project configurations, to better understand the development process and variations thereof and, in future, to apply the most suitable planning alternatives for the respective context. Beyond the analysis of the *Software Factory* process, we also analyzed the general adaptability of our simulation and therefore evaluated the suitability of the simulation model for further process models. For this, we tailored the simulation model to support "pure" Scrum and Kanban and conducted a comparative analysis of the processes' `cycle_time`. Again, we could see that the simulation model adequately reproduces the real case data.

Companies doing this kind of simulation projects can use the presented simulation model for identifying and better understanding factors (e.g., communication, work assignments) that could have an impact on the planning and operation of projects. These factors might need a specific consideration. The simulation models might also help to better understand the mechanics and dynamic relationships inside such projects or lead to important questions to be posed before starting a project. However, the models are not aimed at generating precise point estimates or supporting decision making at a micro level. This would require a very careful customization of the models to a company's context and a respective calibration.

Limitations. The model presented only partially addresses the (quantitative) relationship of different actions, which introduces some conceptual issues (e.g., human factors) in the model. Hence, the simulation capabilities of the model are limited to only those project aspects that can be sufficiently measured. Therefore, the results obtained in the presented simulation are limited for specific cases and can only serve as indication, but do not yet allow for generalization. However, such a generalization would be very helpful to have "standard" process customizations at disposal, which could be used to calibrate an organization- or project-specific simulation model.

Future Work. Future work thus comprises gathering data from further *Software Factory* projects and from other industrial projects from different contexts. These steps will enhance the data bases and they will support the model's validation to improve its reliability. Furthermore, the present model is expected to be extended to allow for simulating and reproducing further processes, i.e., to be generalized and then customized for application to further domains. We demonstrated this by providing an initial simulation and comparison of the *Software Factory's* Scrumban process and the "pure" Scrum and Kanban processes. Yet, a transfer to other processes and process combinations in different application domains remains subject to future work. Another aspect that is worth consideration is the improvement of the presented simulation model towards a prediction tool. So far, we could increase understanding of the relationships, e.g., project size and work-in-progress, and we could reproduce real project data, i.e., the model is primarily used as analysis tool. Therefore, given a sufficient dataset as a basis and a sufficiently validated model, the approach presented in this paper could also serve as prediction tool to proactively improve the decision-making process of project managers. In this regard, an updated work-in-progress version of the simulator could directly access issue tracking systems such as Jira or Redmine. This extended simulation tool would collect data about the project such as, e.g., number and list of issues, estimated time and time spent for resolving issues, priority of issues, team size, and the process followed as a workflow (number of steps and connection among the steps). Furthermore, this extended simulator could be quickly adapted for a particular project to reproduce and/or simulate the project providing the total time needed to finish the project and some statistics, e.g., concerning the number of issues per day, developer productivity, and so forth. Using Montecarlo simulations and variations of project parameters such as developer availability or error in effort estimation, such and updated simulator would also allow for risk analyses.

References

Ahmad, M.O., Kuvaja, P., Oivo, M., Markkula, J., 2016. Transition of software maintenance teams from scrum to kanban, in: Hawaii International Conference on System Sciences, IEEE. pp. 5427–5436.

Ahmad, M.O., Markkula, J., Oivo, M., 2013. Kanban in software development: A systematic literature review, in: Euromicro Conference on Software Engineering and Advanced Applications, IEEE. pp. 9–16.

Alajrami, S., Gallina, B., Romanovsky, A., 2016. Enabling global software development via cloud-based software process enactment. Technical Report TR-1494. Newcastle University.

Ali, N.B., Petersen, K., de França, B.B.N., 2015. Evaluation of simulation-assisted value stream mapping for software product development: Two industrial cases. Information and software technology 68, 45–61.

Alzoubi, Y.I., Gill, A.Q., Al-Ani, A., 2016. Empirical studies of geographically distributed agile development communication challenges: A systematic review. Information & Management 53, 22–37.

Anderson, D., Concas, G., Lunesu, M.I., Marchesi, M., 2011. Agile Processes in Software Engineering and Extreme Programming: 12th International Conference, XP 2011, Madrid, Spain, May 10-13, 2011. Proceedings. Springer Berlin Heidelberg, Berlin, Heidelberg. volume 77 of *Lecture Notes in Business Information Processing*. chapter Studying Lean-Kanban Approach Using Software Process Simulation. pp. 12–26.

Anderson, D.J., Concas, G., Lunesu, M.I., Marchesi, M., Zhang, H., 2012. Agile Processes in Software Engineering and Extreme Programming: 13th International Conference, XP 2012, Malmö, Sweden, May 21-25, 2012. Proceedings. Springer Berlin Heidelberg, Berlin, Heidelberg. volume 111 of *Lecture Notes in Business Information Processing*. chapter A Comparative Study of Scrum and Kanban Approaches on a Real Case Study Using Simulation. pp. 123–137.

Armbrust, O., Berlage, T., Hanne, T., Lang, P., Münch, J., Neu, H., Nickel, S., Rus, I., Sarishvili, A., Stockum, S.V., Wirsen, A., 2005. Handbook of Software Engineering and Knowledge Engineering. World Scientific. volume 3. chapter Simulation-based Software Process Modeling and Evaluation. pp. 333–364.

Bird, C., Nagappan, N., Devanbu, P., Gall, H., Murphy, B., 2009. Does distributed development affect software quality? an empirical case study of windows vista, in: International Conference on Software Engineering, IEEE Computer Society, Washington, DC, USA. pp. 518–528.

Cao, L., Ramesh, B., Abdel-Hamid, T., 2010. Modeling dynamics in agile software development. ACM Transactions on Management Information Systems (TMIS) 1, 5.

Cocco, L., Mannaro, K., Concas, G., Marchesi, M., 2011. Simulating kanban and scrum vs. waterfall with system dynamics, in: International Conference on Agile Software Development, Springer. pp. 117–131.

Concas, G., Lunesu, M.I., Marchesi, M., Zhang, H., 2013. Simulation of software maintenance process, with and without a work-in-process limit. Journal of Software: Evolution and Process 25, 1225–1248.

Ebert, C., Kuhrmann, M., Prikladnicki, R., 2016. Global software engineering: An industry perspective. Software, IEEE 33, 105–108.

Fagerholm, F., Kuhrmann, M., Münch, J., 2017. Guidelines for using empirical studies in software engineering education. PeerJ Computer Science 3.

Fagerholm, F., Oza, N., Münch, J., 2013. A platform for teaching applied distributed software development: The ongoing journey of the helsinki software factory, in: 2013 3rd International Workshop on Collaborative Teaching of Globally Distributed Software Development (CTGDSD), pp. 1–5.

Femmer, H., Kuhrmann, M., Stimmer, J., Junge, J., 2014. Experiences from the design of an artifact model for distributed agile project management, in: International Conference on Global Software Engineering, IEEE Computer Society, Washington, DC, USA. pp. 1–5.

Hashmi, S.I., Clerc, V., Razavian, M., Manteli, C., Tamburri, D.A., Lago, P., Nitto, E.D., Richardson, I., 2011. Using the cloud to facilitate global software development challenges, in: International Conference on Global Software Engineering Workshops, IEEE Computer Society, Washington, DC, USA. pp. 70–77.

Herbsleb, J., Mockus, A., 2003. An empirical study of speed and communication in globally distributed software development. Software Engineering, IEEE Transactions on 29, 481–494.

Houston, D., 2012. Research and practice reciprocity in software process simulation, in: Proceedings of the International Conference on Software and System Process, IEEE Press, Piscataway, NJ, USA. pp. 219–220.

Humphrey, W., 2000a. The Personal Software Process (PSP). Technical Report CMU/SEI-2000-TR-022. Software Engineering Institute, Carnegie Mellon University. Pittsburgh, PA.

Humphrey, W., 2000b. The Team Software Process (TSP). Technical Report CMU/SEI-2000-TR-023. Software Engineering Institute, Carnegie Mellon University. Pittsburgh, PA.

Ikonen, M., Pirinen, E., Fagerholm, F., Kettunen, P., Abrahamsson, P., 2011. On the impact of kanban on software project work: An empirical case study investigation, in: IEEE International Conference on Engineering of Complex Computer Systems, IEEE. pp. 305–314.

Kellner, M.I., Madachy, R.J., Raffo, D.M., 1999. Software process simulation modeling: why? what? how? Journal of Systems and Software 46, 91–105.

Kniberg, H., Skarin, M., 2010. Kanban and Scrum-making the most of both. Enterprise Software Development, lulu.com.

Kuhrmann, M., Diebold, P., Münch, J., Tell, P., Garousi, V., Felderer, M., Trektere, K., McCaffery, F., Prause, C.R., Hanser, E., Linssen, O., 2017. Hybrid software and system development in practice: Waterfall, scrum, and beyond, in: Proceedings of the International Confernce on Software System Process, ACM, New York, NY, USA. pp. 30–39.

Lamersdorf, A., Münch, J., del Viso Torre, A.F., Sánchez, C.R., Heinz, M., Rombach, D., 2012. A rule-based model for customized risk identification and evaluation of task assignment alternatives in distributed software development projects. Journal of Software: Evolution and Process 24, 661–675.

Liu, D., Wang, Q., Xiao, J., 2009. The role of software process simulation modeling in software risk management: A systematic review, in: Empirical Software Engineering and Measurement, 2009. ESEM 2009. 3rd International Symposium on, IEEE. pp. 302–311.

Lous, P., Kuhrmann, M., Tell, P., 2017. Is scrum fit for global software engineering?, in: International Conference on Global Software Engineering, IEEE Press, Piscataway, NJ, USA. pp. 1–10.

Lunesu, I., Münch, J., Marchesi, M., Kuhrmann, M., 2017. Using measurement and simulation for understanding distributed development processes in the cloud, in: International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement, ACM, New York, NY, USA. pp. 1–11.

Lunesu, M.I., 2013. Process Software Simulation Model of Lean-Kanban Approach. Ph.D. thesis. University of Cagliari.

Martin, R., Raffo, D., 2001. Application of a hybrid process simulation model to a software development project. Journal of Systems and Software 59, 237 – 246. Software Process Simulation Modeling.

Melis, M., Turnu, I., Cau, A., Concas, G., 2006. Evaluating the impact of test-first programming and pair programming through software process simulation. Software Process: Improvement and Practice 11, 345–360.

Mujtaba, S., Feldt, R., Petersen, K., 2010. Waste and lead time reduction in a software product customization process with value stream maps, in: Australian Software Engineering Conference, IEEE. pp. 139–148.

Paasivaara, M., Durasiewicz, S., Lassenius, C., 2009. Using scrum in distributed agile development: A multiple case study, in: Global Software Engineering, 2009. ICGSE 2009. Fourth IEEE International Conference on, IEEE. pp. 195–204.

Phalnikar, R., Deshpande, V., Joshi, S., 2009. Applying agile principles for distributed software development, in: Advanced Computer Control, 2009. ICACC'09. International Conference on, IEEE. pp. 535–539.

Portillo-Rodríguez, J., Vizcaíno, A., Piattini, M., Beecham, S., 2012. Tools used in global software engineering: A systematic mapping review. Inf. Softw. Technol. 54, 663–685.

Ramesh, B., Cao, L., Mohan, K., Xu, P., 2006. Can distributed software development be agile? Communications of the ACM 49, 41–46.

Rus, I., Neu, H., Münch, J., 2003. A systematic methodology for developing discrete event simulation models of software development processes, in: In Proceedings of the 4th International Workshop on Software Process Simulation and Modeling, IEEE.

Schwaber, K., Beedle, M., 2002. Agile software development with Scrum. Prentice Hall.

Sengupta, B., Chandra, S., Sinha, V., 2006. A research agenda for distributed software development, in: International Conference on Software Engineering, ACM, New York, NY, USA. pp. 731–740.

Shadish, W.R., Cook, T.D., Campbell, D.T., 2001. Experimental and Quasi-Experimental Designs for Generalized Causal Inference. Cengage Learning, Boston, New York. 2 edition.

Solingen, R.V., Berghout, E., 1999. Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development. McGraw-Hill Inc.. 1 edition.

Sutherland, J., Viktorov, A., Blount, J., Puntikov, N., 2007. Distributed scrum: Agile project management with outsourced development teams, in: System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on, IEEE. pp. 274a–274a.

Tanner, M., Dauane, M.E., 2017. The use of kanban to alleviate collaboration and communication challenges of global software development. Issues in Informing Science and Information Technology 14, 177–197.

Theocharis, G., Kuhrmann, M., Münch, J., Diebold, P., 2015. Is Water-Scrum-Fall reality? on the use of agile and traditional development practices, in: International Conference on Product Focused Software Development and Process Improvement, Springer, Cham. pp. 149–166.

Tregubov, A., Lane, J.A., 2015. Simulation of kanban-based scheduling for systems of systems: initial results. Procedia Computer Science 44, 224–233.

Turner, R., Madachy, R., Ingold, D., Lane, J.A., 2012. Modeling kanban processes in systems engineering, in: Proceedings of the International Conference on Software and System Process, IEEE Press, Piscataway, NJ, USA. pp. 23–27.

Turnu, I., Melis, M., Cau, A., Setzu, A., Concas, G., Mannaro, K., 2006. Modeling and simulation of open source development using an agile practice. Journal of Systems Architecture 52, 610–618.

Vallon, R., da Silva Estácio, B.J., Prikladnicki, R., Grechenig, T., 2017. Systematic literature review on agile practices in global software development. Information and Software Technology .

Vijayasarathy, L.R., Butler, C.W., 2016. Choice of software development methodologies: Do organizational, project, and team characteristics matter? IEEE Software 33, 86–94.

Šmite, D., Wohlin, C., Gorschek, T., Feldt, R., 2010. Empirical evidence in global software engineering: A systematic review. Empirical Softw. Engg. 15, 91–118.

Wakeland, W.W., Martin, R.H., Raffo, D., 2004. Using design of experiments, sensitivity analysis, and hybrid simulation to evaluate changes to a software development process: a case study. Software Process: Improvement and Practice 9, 107–119.

Wang, X., Conboy, K., Cawley, O., 2012. "leagile" software development: An experience report analysis of the application of lean approaches in agile software development. Journal of Systems and Software 85, 1287–1299. Special Issue: Agile Development.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A., 2012. Experimentation in Software Engineering. Springer-Verlag, Berlin Heidelberg.

Yara, P., Ramachandran, R., Balasubramanian, G., Muthuswamy, K., Chandrasekar, D., 2009. Global Software Development with Cloud Platforms. Springer Berlin Heidelberg, Berlin, Heidelberg. volume 35 of *Lecture Notes in Business Information Processing*. pp. 81–95.

Zhang, H., Kitchenham, B., Pfahl, D., 2008. Reflections on 10 years of software process simulation modeling: a systematic review, in: International Conference on Software Process, Springer. pp. 345–356.