INCENTIVES AND PERFORMANCE IN LARGE-SCALE LEAN SOFTWARE DEVELOPMENT
An Agent-based Simulation Approach

Benjamin S. Blau, Tobias Hildenbrand, Matthias Armbruster, Martin G. Fassunge
SAP AG, Dietmar-Hopp-Allee 16, Walldorf, Germany

Yongchun Xu, Rico Knapper
Research Center for Information Technology, Haid-und-Neu-Straße 10-14, Karlsruhe, Germany

Keywords: Lean, Agile, Agent-based simulation, Performance, Incentive.

Abstract: The application of lean principles and agile project management techniques in the domain of large-scale software product development has gained tremendous momentum over the last decade. However, a simple transfer of good practices from the automotive industry combined with experiences from agile development on a team level is not possible due to fundamental differences stemming from the particular domain specifics – i.e. different types of products and components (material versus immaterial goods), knowledge work versus production systems as well as established business models. Especially team empowerment and the absence of a a hierarchical control on all levels impacts goal orientation and business optimization. In such settings, the design of adequate incentive schemes in order to align local optimization and opportunistic behavior with the overall strategy of the company is a crucial activity of central importance.

Following an agent-based simulation approach with reinforcement learning, we *(i)* address the question of how information regarding backlog item dependencies is shared within and in between development teams on the product level subject to different incentive schemes. We *(ii)* compare different incentive schemes ranging from individual to team-based compensation. Based on our results, we are *(iii)* able to provide recommendations on how to design such incentives, what their effect is, and how to chose an adequate development structure to *foster overall software product development flow by means of more economic decisions and thus resulting in a shorter time to market*. For calibrating our simulation, we rely on practical experience from a very large software company piloting and implementing lean and agile for about three years.

1 INTRODUCTION

The application of lean and agile principles in large-scale software product development turns out as non-trivial transition and change management endeavor in most companies (Cohn and Ford, 2003). This is partly due to the fact that a simple transfer of known practices from lean manufacturing in other industries cannot be achieved due to differences between production versus product development processes and the nature of knowledge work and immaterial goods—such as software (Poppendieck, 2004; Reinertsen, 2009). Especially breaking down bigger products to an organization requiring multiple teams and hierarchy levels, dealing with product dependencies, and re-integrating features and functions while keeping the overall market and economics of decisions in mind is yet very challenging in the relatively young software industry (Leffingwell, 2007; Larman and Vodde, 2008). As a consequence, phenomena like queued artifacts, delayed product deliveries, and long-tail risks occur (Reinertsen, 2009).

This research aims at gaining a better understanding of the information sharing and motivation mechanics of a complex socio-technical system, such as a large-scale software product development organization. Based on this increased understanding, we want to derive implications for designing the development organization, and issue incentives for the teams in order to *foster overall software product development flow by means of more informed and economic decisions, resulting in a shorter time to market*.

S. Blau B., Hildenbrand T., Armbruster M., G. Fassunge M., Xu Y. and Knapper R..
INCENTIVES AND PERFORMANCE IN LARGE-SCALE LEAN SOFTWARE DEVELOPMENT - An Agent-based Simulation Approach.
DOI: 10.5220/0003418300260037
In *Proceedings of the 6th International Conference on Evaluation of Novel Approaches to Software Engineering* (ENASE-2011), pages 26-37
ISBN: 978-989-8425-57-7

Based on our research goal and the complex, large-scale industrial setting (see section 5), we follow an *agent-based* simulation approach with reinforcement learning. Using this method we *(i)* investigate the information flow in lean large-scale software product development systems in terms of dependency resolution between requirements, user stories, and other software artifacts (cp. (Hildenbrand, 2008; Sommerville, 2010)). In this context, incentives for individuals to share such information are of central importance. Therefore, we *(ii)* furthermore tackle the question of how different types of incentive schemes impact information flow and the overall performance of empowered teams. Based on our simulation results, we *(iii)* provide recommendations on how to design such incentives and how to chose an adequate development structure within an organization. For calibrating our simulation, we rely on three years of experience from one of today's largest lean and agile adoption at SAP AG (Schnitter and Mackert, 2010).

The remainder of this paper is structured as follows: Section 2 outlines related research in the context of agile and lean software development. The agent-based simulation methodology and the corresponding field of research is analyzed in Section 3. The basic model underlaying the empirical evaluation is described in Section 4. The simulation, its parametrization and the research hypotheses are specified in detail in Section 5. Evaluation results and their practical implications are discussed in Section 6. Section 7 summarizes our contribution and outlines future work.

2 RELATED WORK

In order to model and understand a complex socio-technical system, such as a multi-level software product development organization, the underlying design principles and processes need to be investigated. In this work, we specifically address the application of lean and agile principles in large software development companies (e.g. (Schnitter and Mackert, 2010)). While there is mostly narrative literature on agile principles and Scrum in large-scale enterprise environments "driven by practitioners and consultants" (Conboy, 2009, p. 329)—examples include (Leffingwell, 2007; Schwaber, 2007; Larman and Vodde, 2008; Leffingwell, 2009; Larman and Vodde, 2010), there is only little empirical evidence and rigorous research in this field. For instance, there is only little research on the effectiveness and efficiency gains actually achieved by introducing lean and agile principles, Scrum-based project management etc.—in this small set less than 2% exhibit acceptable rigor, credibility, and relevance (Dyba and Dingsoyr, 2008, p. 851), while 75% of these studies only investigated agile projects specifically applying eXtreme Programming (XP, (Beck, 1999; Dingsoyr et al., 2010)).

2.1 Agile Team Practices

The vast majority of research on agile methods and practices focuses on XP (Beck, 1999; Beck, 2000) as team practice and applies a single or multiple case study methodology (Yin, 2007). Single practices crucial to XP have been examined separately regarding their impact on software quality, e.g. pair programming is said to consume 30% more effort than solo programming (Cao et al., 2010), resulting in 40-90% fewer defects (Williams et al., 2000; Erdogmus and Williams, 2003; Cao et al., 2010). However, with respect to the broad range of agile methods and their increasing prevalence in the software industry (West and Grant, 2010), there is only very little scientific evidence so far whether or not these models lead to more effectiveness, efficiency, or productivity, respectively, in real-world large-scale development environments (Dyba and Dingsoyr, 2008).

Among the few evidence-based behavioral science contributions (Hevner et al., 2004) on software agility, Lee and Xia (Lee and Xia, 2010) investigated the impact of two major agile characteristics (team autonomy and team diversity) on three productivity measures: (1) on-time and (2) on-budget completion as well as (3) functionality provided to customers. Among their findings, it turned out that there are conflicting goals even within the boundaries of one team. Besides these findings, the model exhibits that the dependent productivity variables could only be explained to a degree that leaves substantial room for future behavioral studies.

2.2 Large-scale Lean and Agile

Lean management or lean thinking – as underlying philosophy and common set of values – as well as lean and agile principles are either already implemented or piloted in many practical scenarios of different scales today, e.g. at Salesforce (Fry and Greene, 2007) or SAP (Schnitter and Mackert, 2010). Figure 1 visualizes how specific agile software development practices, such as XP (Beck, 1999), test-driven development (TDD, (Beck, 2003)) and agile project management methods like Scrum (Schwaber and Beedle, 2001) build upon agile principles and lean thinking values. While the basic principles and philosophy apply to many industries, some address a specific one more concretely, e.g. Scrum and XP for the software industry.

Based on general principles of lean management (Poppendieck and Poppendieck, 2006) and lean thinking as well as basic agile principles (Agile Alliance, 2001) and consulting experience, a set of guiding principles and practices for scaling Scrum to larger-scale scenarios evolved, see e.g. (Poppendieck and Poppendieck, 2003; Poppendieck and Poppendieck, 2006; Larman and Vodde, 2008; Larman and Vodde, 2010). In the same vein, similar large-scale Scrum models have been described by Schwaber (Schwaber, 2007) and Leffingwell (Leffingwell, 2007). These ideas on lean management and lean software development have been further elaborated and translated to some practical guidelines based on experience from multiple consulting projects, see e.g. (Larman and Vodde, 2010). However, lean software development in large enterprise environments requires scaling team-based approaches such as Scrum (see section 2.1. Nevertheless, first implementation concepts and pilot approaches can be found even for very large-scale software vendors (Schnitter and Mackert, 2010). Hence, empirical research and evidence for complex socio-technical system in the software industry is even more scarce than for team practices (cp. section 2.2 and (Dyba and Dingsoyr, 2008)).

Lean and agile software development is based on lean enterprise characteristics comprising focus on value, synchronization, transparency, and perfection as well as Just-in-Time (JIT) principles such as (one-piece) flow, takted development, customer pull, and zero defects (Reinertsen, 2009).

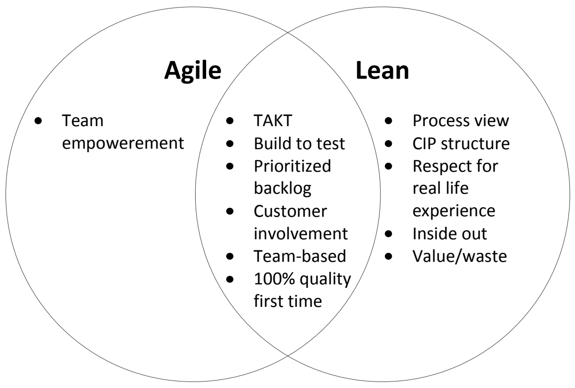

Figure 1: Comparison and interrelation of lean and agile principles.

Combining the lean enterprise perspective with an agile perspective on development teams (Agile Alliance, 2001), leads to short iterative development cycles, a uniquely prioritized backlog of requirements and work items, direct customer involvement, as well as tested and potentially shippable software increments.

As a common basis for further studies on agile practices, Conboy has developed a unified definition and formative taxonomy of agility in informations systems development or software engineering, respectively (Conboy, 2009, pp. 340). Such a common definition and/or taxonomy is required to link existing and future contributions in this very interdisciplinary field of research, e.g. from information systems, computer science, organizational science, sociology and psychology.

In context of lean and agile software development, there are to-date only very few related simulation-based contributions, e.g. using a system dynamics approach (Cao et al., 2010). Moreover, (Petersen and Wohlin, 2010) present potential performance indicators and visualizations for flow simulations (cp. also (Reinertsen, 2009)).

3 METHODOLOGY

3.1 Simulation-based Approaches

Complementary to mere behavioral and design science studies (Hevner et al., 2004), a simulation-based approach allows to analyze and better understand complex development scenarios with hundreds or even thousands of individuals and even more artifacts and process dependencies. Besides deduction and induction, experimenting with simulations is considered a “third way of doing science” (Axelrod, 1997). To analyze and optimize complex development scenarios, different analytical and simulation-based approaches can be considered: discrete event simulations, agent-based simulation (Blau et al., 2010a; Blau et al., 2010b), system dynamics etc. Simulating software development processes to answer fundamental questions about agile and lean practices is, though still scarce, rising in number (Cao et al., 2010, see).

The complexity arising from individual actions and interactions that arise in the real world can be explicitly modeled in agent-based simulations in situations discrete-event simulations or system dynamics cannot (Siebers et al., 2010). Although being relatively new, agent-based simulations gain more and more momentum in various application areas where the behavior of single individual actors constitute the fundamental issues (Macal and North, 2007). An agent-based system consists of autonomous agents following simple behavioral rules while being a direct abstraction from their real-world counterparts. Being autonomous and able to learn from their environment, they behave proactively following their own rule set (Siebers et al., 2010). Thus, the interaction among the agents directly impacts the system properties (Bonabeau, 2002a).

(Siebers et al., 2010) and (North and Macal, 2007) point out various issues where agent-based simulations are well applicable. Among other reasons, agent-based simulations can be used

- when agents are a natural representation of a system's participants,
- when the major factor is learning and adapting,
- when agents behave proactively, i.e. they make strategic decisions based on past, current as well as anticipated behavior of other agents, and
- when one important factor of a system is the relationship between the participants, i.e. agents form and dissolve relationships with each other

Evaluating certain mechanism properties or behavior of participants in settings with a multitude of variable factors, a theoretical analysis is not applicable in most of the cases due to the high complexity of the system. As a remedy, numerical simulations provide a useful tool to analyze particular properties of a mechanism by means of randomly generated problem sets, i.e. the variable factors are randomly generated for multiple simulation runs. Numerical simulations can provide insights into the general problem structure, performance aspects of the algorithm that solves the winner determination problem, mechanism properties, and strategic behavior of participants.

Focusing on more complex settings with participants that face large strategy spaces precluding theoretical solutions, the methodology of agent-based simulations has proven to be promising (Bonabeau, 2002b). In contrast to a traditional game theoretical analysis, agent-based simulations provide means for the evaluation of rare strategies which are more complex and occur in special domains. Nevertheless, it is crucial to design reasonable strategies as well as a reasonable learning behavior and incorporate them into software agents. Based on this notion, a lot of work has been done in the area of agent-based simulations, and a whole set of different strategies has been shown to work well in many settings (Phelps, 2008).

This section has shown that following an agent-based approach is an optimal choice to address the research questions. The following section will introduce, besides the system's structure and further artifacts, the actual model taken for implementation.

4 ASSUMPTIONS & MODEL

This paper addresses large-scale business software development organizations with several hundred or thousands of developers. Moreover, we take a development process based on lean management and agile principles as a basis for our assumptions. In addition, this section describes the basic model of our agent-based simulation in a mathematical notation.

4.1 Work Items and Artifacts

Iteration Backlog. This backlogs contain all the user stories (backlog items) one team has committed to for one iteration, or sprint respectively, in Scrum. The backlog items are permanently kept uniquely prioritized by the team's product owner (Schwaber and Beedle, 2001).

Iteration Backlog Item. User stories are containers for requirements and currently one of the most popular requirement modeling technique in agile methods. "User stories are the primary currency that carries the customer's requirements through the value stream into code and implementation." (Leffingwell, 2009). They briefly describe a feature from the perspective of a certain user role, letting the team freedom in implementational details. The effort of each user story is estimated in 'story points' instead of interpreted as person day effort, which are oftentimes classified into a Fibonacci-like sequence, i.e. 1,1,2,3,5,13, etc. (Cohn, 2006).

Usable Software Increments each Iteration. At the end of each iteration the team produces a new software increment. This increment must be properly tested and fulfill other criteria in order to be accepted by the responsible person with regard to prior defined "done" (non-functional and/or meta-requirements) and functional "acceptance criteria". Agile methods aim at completing potentially shippable product increments, i.e. usable software in each iteration.

4.2 Team Process and Structure

Agile methods, such as Scrum, try to attain a trade-off between pragmatism and discipline, i.e. avoiding chaos on the hand and extensive bureaucracy on the other (see figure 2).

Team Size and Skills. The team must be "fully capable of defining, developing, testing, and delivering

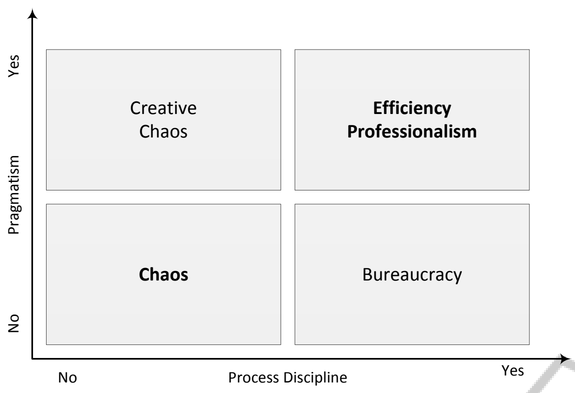

Figure 2: Pragmatism vs. discipline.

working and tested software into the system's baseline" (Leffingwell, 2009). Usually, such a "cross-functional" team consists of one product owner, a scrum master, 5-10 team members focussing on development, quality and testing as well as other functions and skills (Larman and Vodde, 2010). Teams are typically organized around particular software *components* (architectural view) or certain *features* (from a customer's perspective, see (Larman and Vodde, 2008)). In general, features and components exhibit inherently dependent requirements, i.e. inter-team dependencies. In practice, companies have a mixture of both, feature and component teams, organized in a matrix (see e.g. (Schnitter and Mackert, 2010)).

Inter-team Collaboration in Large Development Organizations. In order to be able to release complex and comprehensive software products, development organizations of several hundred or even thousands of developers in cross-functional teams need to be coordinated, for which hierarchy levels need to be introduced. In our research, we follow the large-scale lean and agile model by Larman and Vodde (Larman and Vodde, 2008; Larman and Vodde, 2010). This is also the basis for the implementation with which we calibrate our model (Schnitter and Mackert, 2010). The mix of feature and component teams (see above) is one of the reasons for occurring inter-team dependencies, which need to be resolved for product delivery. For instance, a certain set of master data requires multiple functional components of an enterprise resource planning application.

4.3 Model Parameters & Behavior

Agents & Teams. Let A^m represent the set of *agents* (e.g. developers and other cross-functional team members) in *team m* (m and n are arbitrary teams in the remainder of this article) with agents $a_1^m, \ldots, a_q^m$ such that $A^m = \{a_1^m, \ldots, a_q^m\}$. Let the agent a_1^m be a special agent ("team owner") representing the *Scrum Master* and the *Product Owner* of team *m*.[1] A team's *capacity* c^m is determined by the number of its agents n minus the team owner, i.e. a team $A^m = \{a_1^m, \ldots, a_q^m\}$ has the capacity of $c^m = q - 1$.

Team Backlog. Let furthermore B^m denote the *backlog* of team m with prioritized *backlog items* $b_1^m, \ldots, b_k^m$ such that $B^m = \{b_1^m, \ldots, b_l^m\}$ (b_x and b_y are arbitrary backlog items in the remainder of this article). The index l represents the priority or rank within the backlog – i.e. the backlog item b_{l-1}^m is the unambiguous antecessor of the backlog item b_l^m.

Backlog Processing. It is assumed that until all done criteria are satisfied, the *processing* of a backlog item consumes a well-defined[2] period of time t. The processing function $\lambda : B \rightarrow T$ maps backlog items to a processing time $t \in T$.

Backlog Dependencies. It is further assumed that *dependencies* between backlog items may exist such that the possibility to start processing a specific backlog item depends on the successful processing and finalizing of another item (all done criteria fulfilled). The dependency function $d : B \times B \rightarrow \{0, 1\}$ maps a pair of backlog items (b_x^m, b_y^m) to elements $0, 1$ with 0 representing independent backlog items and 1 denoting that backlog item b_x^m is dependent on item b_y^m.

$$d(b_x^m, b_y^n) = \begin{cases} 1 & ,\text{if} \quad b_x^m \text{ is independent of } b_y^n \\ 0 & ,\text{if} \quad b_x^m \text{ depends on } b_y^n \end{cases} \tag{1}$$

For the sake of simplicity, it is assumed that dependencies are *not directed*, i.e. if backlog items are dependent, neither of them can be processed as long as the dependency persists. More precisely, this implies that if $d(b_x^m, b_y^n) = 1 \Rightarrow d(b_y^m, b_x^n) = 1 \wedge d(b_x^m, b_y^n) = 0 \Rightarrow d(b_y^m, b_x^n) = 0 \, \forall \, x \neq y$.

From a team's perspective, it follows that there are two designated types of dependencies, i.e. *(i) intra-team dependencies* with $d(b_x^m, b_y^n) = 1 \ \wedge \ m = n$, i.e.

[1]Our model is simplified based on the assumption that both, Scrum Master or Product Owner, can take over team tasks with approximately 50% of their capacity—therefore, one full-time equivalent is accounted per team. The team owner parameter is also applied for Area Product Owners and Chief Product Owner depending on the level of hierarchy and aggregation.

[2]As an extension of the model, the processing time might be represented by a probability distribution.

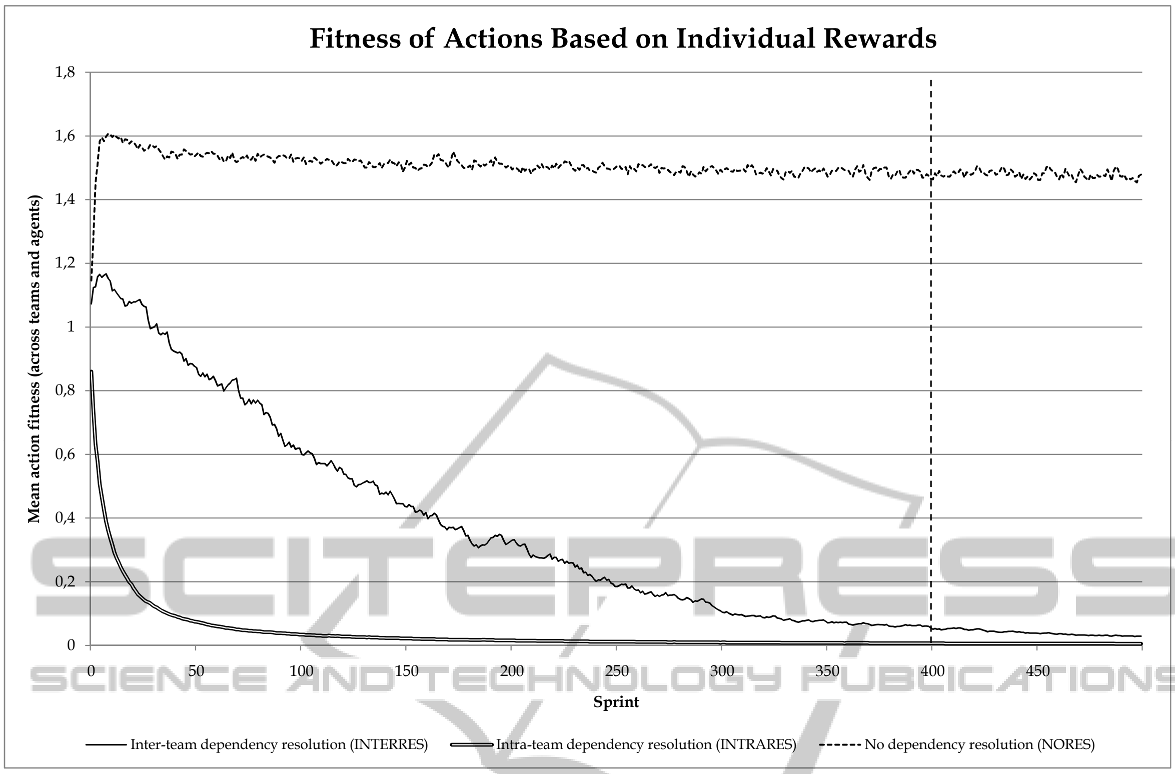

Figure 3: Mean fitness across all agents and teams with individual rewards for actions inter-team dependency resolution (INTERRES), intra-team dependency resolution (INTRARES), and no dependency resolution (NORES) (including training phase).

within the team's own backlog and *(ii) inter-team dependencies* with $d(b_x^m, b_y^n) = 1 \;\wedge\; m \neq n$, i.e. with other teams' backlog items.

Dependency Resolution. It is further assumed that dependencies between backlog items need to be resolved during the development process. Such a resolution is done by investing additional time and effort for analysis, communication, coordination, and in some cases, idle time. This means that the cost for dependency resolution depends on three factors: *(i)* The point in time during the development process the resolution is conducted and *(ii)* the complexity of the dependent backlog items which is implicitly represented by their processing time as well as *(iii)* the type of dependency (intra- or inter-team dependency). Practically this means: The earlier a dependency is detected and the lower the item complexity is, the less additional time is required to resolve it. The amount of effort, i.e. the additional time to be spent for resolving the dependency, also depends on the type of dependency (inter-team or intra-team). Thus, the resolution function $r : B \times B \times \Theta \to T$ (Equation 2) maps pairs of backlog items and the point of time within the development process to the period of time that is required for resolving their dependency (for a complete mapping, the resolution functions returns $t = 0$ in case backlog items are independent).

The resolution time at least equals the average processing time of both items, i.e. their mean complexity and is mainly determined by the constant $\bar{t}$ representing the type of dependency (intra- or inter-dependency) and the point of time θ the resolution is conducted.

5 SIMULATION

Thus, the evaluation is conducted by means of an agent-based simulation based on a simple form of a Q-Learning model (Watkins and Dayan, 1992). In contrast to more sophisticated variants of Q-learning models, the simulation model at hand considers multiple actions but only a single state. This reduction of parameter complexity is done without loss of validity and therefore simplifies the calibration of the simulation. Simplifying the simulation model reduces the number of assumptions, allowing for a better generalization of results.

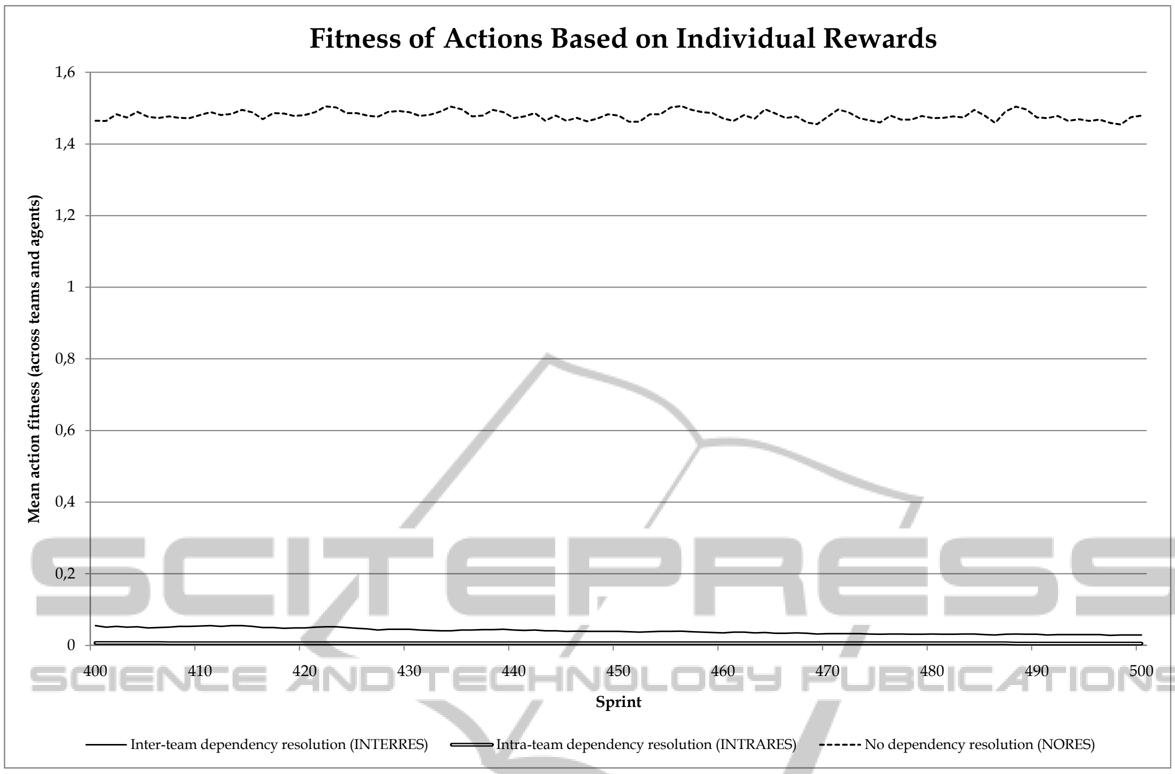

Figure 4: Mean fitness across all agents and teams with individual rewards for actions inter-team dependency resolution (INTERRES)(mean=0.040, std=0.008), intra-team dependency resolution (INTRARES)(mean=0.006, std=0.0004), and no dependency resolution (NORES)(mean=1.479, std=0.012) (convergence phase).

$$r(b_x^m, b_y^n, \theta) = \begin{cases} 0 & ,\text{if} \quad d(b_x^m, b_y^n) = 0 \\ \bar{t}_{\text{intra}} \theta^2 \frac{p(b_x)+p(b_y)}{2} & ,\text{if} \quad d(b_x^m, b_y^n) = 1 \ \wedge \ m = n \\ \bar{t}_{\text{inter}} \theta^2 \frac{p(b_x)+p(b_y)}{2} & ,\text{if} \quad d(b_x^m, b_y^n) = 1 \ \wedge \ m \neq n \end{cases} \tag{2}$$

5.1 Rounds

Reflecting the lean principles, simulation rounds Ω are mapped onto development "takts" (or "sprints" in Scrum (Schwaber and Beedle, 2001)). Each round represents a development takt that is further discretized into a fixed number of takt units ω[3].

5.2 Actions

At the beginning of each TAKT, each agent chooses an action k out of the action space K as specified in Section 5.3. The following actions are available to each agent:

Preceding Intra-team Dependency Resolution.
The agent focuses on resolving dependencies between backlog items within its team at the beginning of the development takt (preceding). If there is capacity left after this action, the agent continues with processing backlog items.

Preceding Inter-team Dependency Resolution.
The agent targets the resolution of dependencies between backlog items that are planned in different teams at the beginning of the development takt (preceding). If there is capacity left after this action, the agent continues with processing backlog items.

Development without Early Dependency Resolution.
No resolution of dependencies at the beginning of the development takt are addressed by the agent, i.e. the agent directly starts with backlog item processing. However, when processing a backlog item that is constrained by a dependency, the agent is forced to resolve this dependency at that point in time which might be time consuming due

[3]For the sake of simplicity, all time-related model values are discretized accordingly.

to the elapsed time (cp. Section 4.3).

Having chosen, the agents execute the particular action which binds their capacity according to the defined time requirements. In case of dependency resolution actions ($k = 1, 2$) the capacity is bound exactly as long as the resolution function specifies the number of TAKT units required. If this period of time is less than the total units within a TAKT, the agent's capacity is free for development activities. In case of the development action ($k = 3$), the agent is processing backlog items during the whole TAKT.

5.3 Feedback & Learning Behavior

At the end of each TAKT Ω, each agent a receives a feedback $\pi^{\Omega}_{a,k}$ as a response to the action k chosen at the beginning of the TAKT.

To analyze the effect of different incentive schemes on the exchange of information within and between teams, we examine three feedback mechanisms:

Individual incentives that reward value creation of the individual developer, i.e. the number of successfully processed backlog items by a single developer.

Team incentives that reward each individual based on the value creation of the whole team the developer belongs to, i.e. the number of successfully processed backlog items accumulated on team-level.

At the end of each takt Ω, the feedback to a chosen action k of an agent a is incorporated in the agent's private fitness function $f^{\Omega}_{a,k}$. Balancing past and present experiences, the learning parameter $\beta \in [0, 1]$ determines to which degree past and present feedback is incorporated into the fitness update. Thus, the fitness update evolves as follows:

$$f^{\Omega}_{a,k} := \beta f^{\Omega-1}_{a,k} + (1-\beta)\pi^{\Omega}_{a,k} \tag{3}$$

Thus, each agent maintains a fitness value for each possible action that represents the historical "success" of that particular action based on the cumulated feedback over time.

At the beginning of each TAKT Ω, each agent a chooses an action k out of the action space K (cp. Section 5.2) based on its particular probability $p^{\Omega}_{a,k}$ that is based on its fitness value and therefore on its historical "success":

$$p^{\Omega}_{a,k} := \frac{f^{\Omega}_{a,k}}{\sum_k {}^{\Omega}_{a,k}} \tag{4}$$

5.4 Parametrization & Hypothesis

The simulation model as described previously is parameterized as follows: According to lean development best practices, the team size is set to 10 agents per team. A learning rate $\beta = 0.5$ yields an optimal trade-off between escaping local optima and achieving a quick convergence of strategies. The first 400 rounds of 500 rounds in total are the simulation's training phase in order to achieve a converged state and are therefore not considered for the statistical evaluation. As the number of variable parameters and their interdependencies are high, heavy statistical noise is likely to be generated. To counteract the high volatility of the simulation model, a large number of 500 problem sets is evaluated and the mean results across all agents and teams are reported. The large size of analyzed problem sets for each observation assures robustness of the t-test to violations of the normality assumption (Sawilowsky and Blair, 1992).

By means of this agent-based simulation approach we intent to verify the hypotheses outlined in Table 1.

Table 1: Incentive schemes and corresponding hypotheses. NORES denotes the mean fitness value of action $k = 1$ across all agents and teams. INTRARES denotes the mean fitness value of action $k = 2$ across all agents and teams. INTERRES denotes the mean fitness value of action $k = 3$ across all agents and teams.

Incentive Scheme	**Hypothesis**
Individual Incentives	H1a: (NORES > INTRARES)
	H1b: (NORES > INTERRES)
	H1c: (INTERRES > INTRARES)
Team Incentives	H2a: (NORES < INTERRES)
	H2b: (INTERRES > INTRARES)

The set of hypotheses is derived from existing literature on the effect of incentives in lean development (Poppendieck, 2004) and practical experiences from lean projects in SAP. In settings with individual incentives that reward agents solely on the number of backlog items that are successfully processed on their own, the agents are likely to follow an opportunistic strategy, i.e. they focus on processing backlog items instead of resolving dependencies (neither within their team nor between teams) as stated in hypotheses H1a and H1b. Resolving inter-team dependencies at a later point in time is more time consuming than intra-team dependencies which is likely to incentivize agents to prefer the INTERRES strategy over the INTRARES strategy at the beginning of each sprint. This argumentation holds for either incentive scheme (cp. hypotheses H1c and H2b).

On the other hand, team incentives that reward agents based on the total number of successfully processed backlog items of the whole team are likely to implement incentives for agents to follow actions which are beneficial for the team itself. As the effort to resolve backlog item dependencies at a later point in time is exponentially higher than at the beginning of the sprint, agents are likely to follow an early dependency resolution (cp. hypotheses H2a).

The statistical significance of the stated hypothesis is tested using a one-tailed matched-pairs t-test analyzing the alternative hypothesis, that is, the mean difference of the actions' fitness values is greater than zero. For the statistical analysis, the first 400 simulation rounds/sprints are skipped as they serve as the initial learning phase of the agents until we observe a convergence of strategies and achieve a stable state.

6 EVALUATION RESULTS & IMPLICATIONS

This sections describes the main findings of the agent-based simulation for the individual and team incentive schemes. Having been analyzed by means of a sensitivity analysis, the observations are robust against the simulation parameters "number of agents per team", "number of teams", and "learning rate".

6.1 Individual Incentives

Simulation settings with the individual incentive scheme yield the following results:

- The action no dependency resolution (NORES) significantly ($p << 0.01$) yields the highest overall mean fitness across all agents and teams which supports Hypothesis H1a and H1b.
- The action inter-team dependency resolution (INTERRES) yields a significantly ($p << 0.01$) higher mean fitness across all agents and teams than the action intra-team dependency resolution (INTRARES) which supports Hypothesis H1c.

In a setting with the individual incentive scheme, Figure 3 depicts the mean fitness across all agents and teams for each action. The convergence phase that is relevant for the statistical analysis is depicts separately in Figure 4.

6.2 Team Incentives

In settings where agents are rewarded based on the total number of successfully processed backlog items of the whole team, the following results can be observed:

- The action inter-team dependency resolution (INTERRES) is strictly dominating the action no dependency resolution ($p << 0.01$) which supports hypothesis H2a.
- The action intra-team dependency resolution (INTRARES) is significantly ($p << 0.01$) outperformed by the action inter-team dependency resolution (INTERRES) which supports hypothesis H2b.

Figure 5 illustrates the actions' mean fitness across all agents and teams based on the team incentive scheme in a setting with 5 teams consisting of 10 team members. The figure shows all simulation rounds including the training phase whereas Figure 6 depicts rounds 400-500 that are relevant for the statistical analysis.

6.3 Practical Implications

In our work, we analyzed the effect of organizational settings and incentive schemes in large-scale lean software development on the information flow within and between teams as well as performance aspects.

Our analysis has shown that individual rewards foster opportunistic behavior in teams, i.e. actions that serve the team by resolving backlog item dependencies and removing impediments are not conducted by the agents. On the other hand, a team incentive scheme that rewards agents based on the total number of successfully processed backlog items of the whole team promote behavior that is beneficial for the whole team. As the effort to resolve backlog item dependencies at a later point in time is exponentially higher than at the beginning of the sprint, agents follow an early dependency resolution. More precisely, resolving inter-team dependencies at a later point in time is more time consuming than intra-team dependencies which incentivizes agents to prefer a dependency resolution across team boundaries. In general, our results underline the importance of dependency resolution— and therefore, traceability and requirements management, in large software organizations (Hildenbrand, 2008).

One of the basic principles of the lean methodology states the empowerment of the teams instead of enforcing a strictly governed process corset (Lee and Xia, 2010). As a trade-off, this implies that managerial monitoring and steering of the development process becomes cumbersome. Therefore, traditional methodologies and tools stemming from well-known project management techniques are partly not applicable in agile environments, which requires new approaches to manage a successful execution of lean

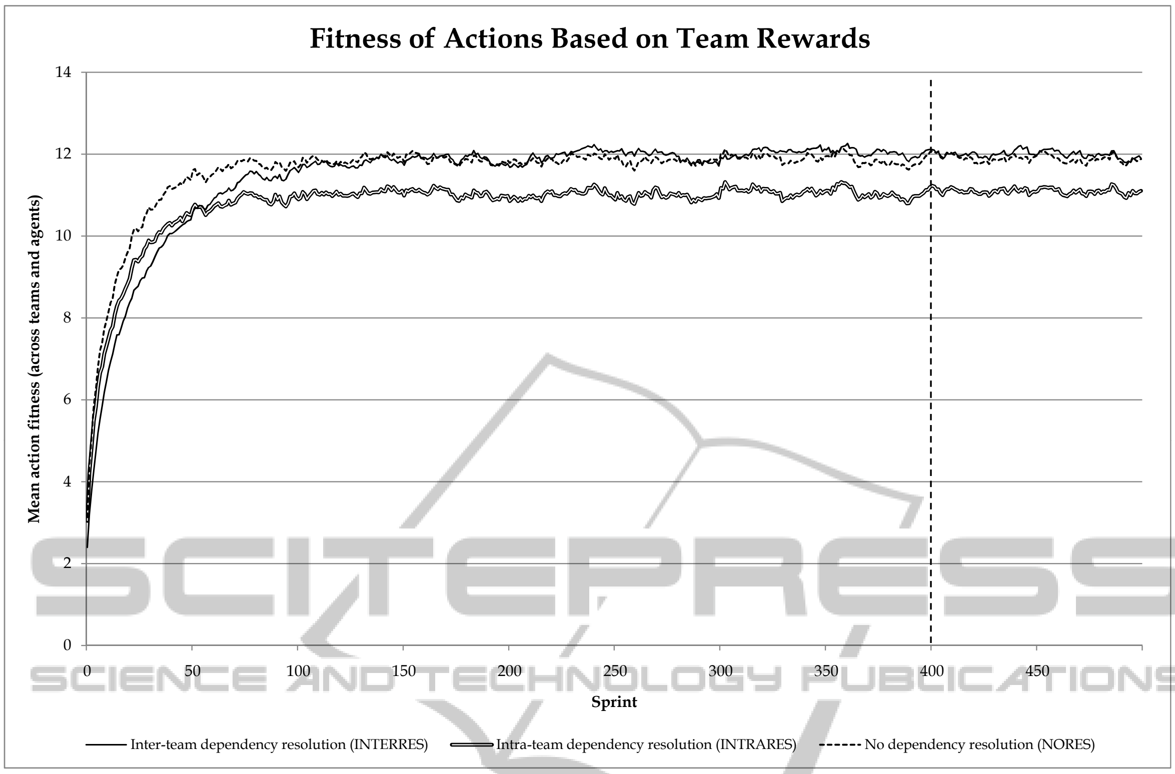

Figure 5: Mean fitness across all agents and teams with team rewards for actions inter-team dependency resolution (INTER-RES), intra-team dependency resolution (INTRARES), and no dependency resolution (NORES) (including training phase).

projects.

Moreover, our work has shown that a sensible and efficient design of incentive schemes in large-scale lean software development is a promising tool to steer individual behavior, diminish opportunism and local optimization, foster efficient communication across team boundaries, and break silos that clash with the company's overall objectives. Hence, our results indicate that team-based rewarding can prevent opportunistic behavior and silo thinking which is in line with recent literature (Poppendieck, 2004).

7 SUMMARY OF FINDINGS & CONCLUSIONS

The contribution of our work comprehends the following findings:

- Incentive schemes play a central role for steering large-scale lean software development and to align individual and company objectives.
- In such complex environments, agent-based simulations are a promising method to evaluate different incentive designs and derive practical implications.
- Rewards based on individual performance advocate selfish behavior of team members, i.e. each individual focuses on silo work instead of removing impediments and sharing information within and between teams to resolve dependencies.
- Rewards directly tied to team-based value creation help to diminish opportunistic behavior and implement incentives to foster backlog item dependency resolution through intense communication across team boundaries.

Outlook. As future work, we will validate our simulation results more systematically with real-world data from large-scale software enterprises implementing lean and agile practices. More specifically, we plan to analyze existing backlogs, log files, and other documentation of work practices as well as conduct qualitative interviews with a certain number of teams from different product areas. In doing, so we intend to (a) further elaborate the external validity of our simulation results and (b) gain more insights regarding the industrial context of our research questions.

Furthermore, we intend to investigate more sophisticated incentive schemes and their composition into hybrid patterns. We also plan to extend our

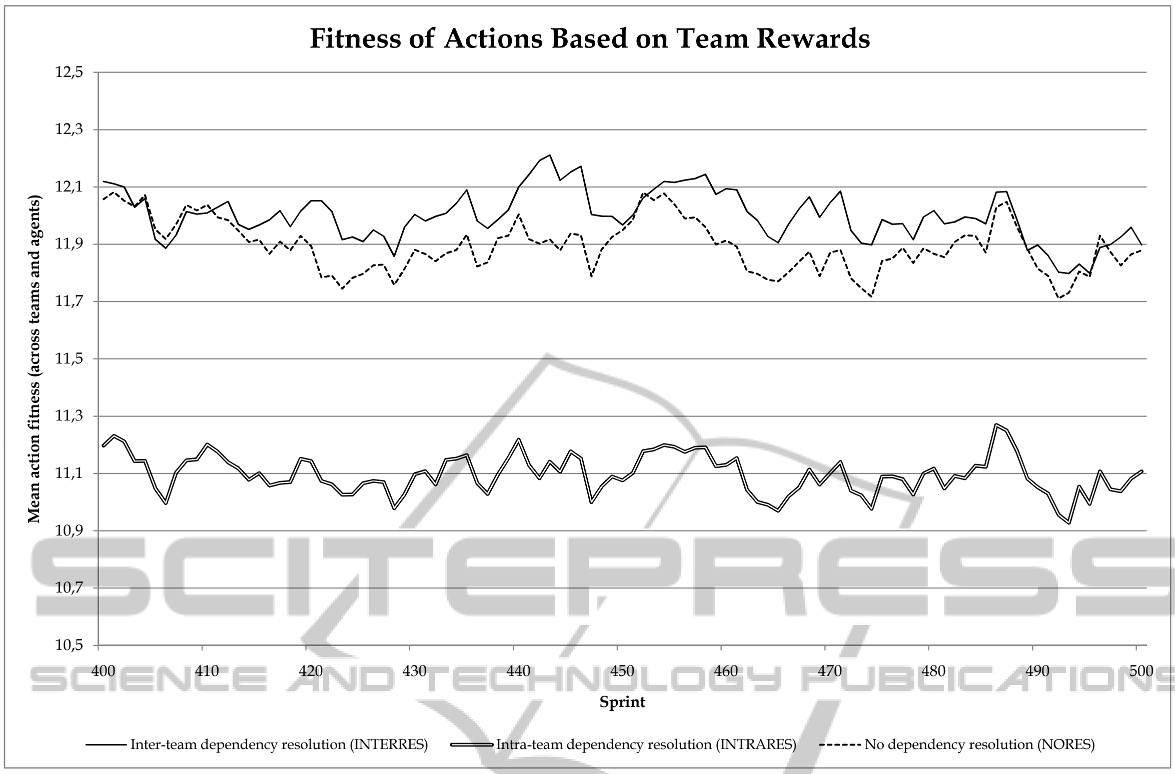

Figure 6: Mean fitness across all agents and teams with team rewards for actions inter-team dependency resolution (INTERRES)(mean=12.0, std=0.087), intra-team dependency resolution (INTRARES)(mean=11.098, std=0.068), and no dependency resolution (NORES)(mean=11.894, std=0.090) (convergence phase).

model regarding hierarchical organizational settings and implications of distributed teams with communication barriers. Questions like how different incentive schemes can be grouped and assessed regarding their applicability and suitability in different organizational settings need to be further investigated. From an economic perspective, we plan to extend the underlying model to capture partly irrational behavior and to vary the feedback quality in terms of timeliness and signal noise.

REFERENCES

Agile Alliance (2001). Agile Manifesto.

Axelrod, R. (1997). Advancing the art of simulation in the social sciences. *Complex.*, 3:16–22.

Beck, K. (1999). Embracing change with extreme programming. *IEEE Computer*, 32:pp. 70–77.

Beck, K. (2000). *Extreme Programming Explained: Embrace Change*. Addison-Wesley.

Beck, K. (2003). *Test-Driven Development: By Example*. Addison-Wesley.

Blau, B., Conte, T., van Dinther, C., and Weinhardt, C. (2010a). A Multidimensional Procurement Auction for Trading Composite Services. *Electronic Commerce Research and Applications Elsevier Journal. Special Issue on Emerging Economic, Strategic and Technical Issues in Online Auctions and Electronic Market Mechanisms*, 9(5):460 – 472. Special Section on Strategy, Economics and Electronic Commerce.

Blau, B., Conte, T., and Weinhardt, C. (2010b). Incentives in Service Value Networks – On Truthfulness, Sustainability, and Interoperability. In *Proceedings of the International Conference on Information Systems*, Saint Louis, Missouri, USA.

Bonabeau, E. (2002a). Agent-based modeling: Methods and techniques for simulating human systems. In *Proceedings of the National Academy of Science of the USA*.

Bonabeau, E. (2002b). Agent-Based Modeling: Methods And Techniques for Simulating Human Systems. In *National Academy of Sciences*, volume 99, pages 7280–7287. National Acad Sciences.

Cao, L., Ramesh, B., and Abdel-Hamid, T. (2010). Modeling dynamics in agile software development. *ACM Trans. Manage. Inf. Syst.*, 1:5:1–5:26.

Cohn, M. (2006). *Agile estimating and planning*. Prentice Hall.

Cohn, M. and Ford, D. (2003). Introducing an agile process to an organization [software development]. *Computer*, 36(6):74–78.

Conboy, K. (2009). Agility from first principles: Reconstructing the concept of agility in information systems development. *Information Systems Research*, 20(3):329–354.

Dingsoyr, T., Dybå, T., and M., B. (2010). *Agile Software Development: Current Research and Future Directions*. Springer.

Dyba, T. and Dingsoyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 50(9-10):pp. 833–859.

Erdogmus, H. and Williams, L. (2003). The economics of software development by pair programmers. *Engin. Econom*, 48:283–319.

Fry, C. and Greene, S. (2007). Large scale agile transformation in an on-demand world. In *Proceedings of the AGILE Conference 2010*, pages pp. 136 – 142.

Hevner, A., March, S., Park, J., and Ram, S. (2004). Design science information systems research. *MIS Quarterly*, 28(1):75–105.

Hildenbrand, T. (2008). *Improving Traceability in Distributed Collaborative Software Development—A Design-Science Approach*. Phd thesis, University of Mannheim, Germany, Frankfurt, Germany.

Larman, C. and Vodde, B. (2008). *Scaling Lean and Agile Development: Thinking and Organizational Tools for Large-Scale Scrum*. Addison-Wesley Longman.

Larman, C. and Vodde, B. (2010). *Practices for Scaling Lean and Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum*. Addison-Wesley Longman.

Lee, G. and Xia, W. (2010). Toward agile: An integrated analysis of quantitative and qualitative field data. *MIS Quarterly*, 34(1):pp.87–114.

Leffingwell, D. (2007). *Scaling software agility: best practices for large enterprises*. Addison-Wesley.

Leffingwell, D. (2009). The big picture of enterprise agility by dean. *Whitepaper*, pages 1–16.

Macal, C. M. and North, M. J. (2007). Agent-based modeling and simulation: desktop abms. In *Proceedings of the 39th conference on Winter simulation: 40 years! The best is yet to come*, WSC '07, pages 95–106, Piscataway, NJ, USA. IEEE Press.

North, M. J. and Macal, C. M. (2007). *Managing Business Complexity: Discovering Strategic Solutions with Agent-Based Modeling and Simulation*. Oxford University Press, Inc., New York, NY, USA.

Petersen, K. and Wohlin, C. (2010). Measuring the flow in lean software development. *Software - Practice and Experience*.

Phelps, S. (2008). *Evolutionary Mechanism Design*. PhD thesis, University of Liverpool.

Poppendieck, M. (2004). Unjust deserts. *BETTER SOFTWARE*, pages 33–47.

Poppendieck, M. and Poppendieck, T. (2003). *Lean software development: an agile toolkit*. Addison-Wesley Professional.

Poppendieck, M. and Poppendieck, T. (2006). *Implementing Lean Software Development: From Concept to Cash*. The Addison-Wesley Signature Series. Addison-Wesley Professional.

Reinertsen, D. G. (2009). *The Principles of Product Development Flow: Second Generation Lean Product Development*. Celeritas Publishing. ISBN 978-1935401001.

Sawilowsky, S. and Blair, R. (1992). A more realistic look at the robustness and type II error properties of the t test to departures from population normality. *Psychological Bulletin*, 111(2):352–360.

Schnitter, J. and Mackert, O. (2010). Introducing agile software development at sap ag - change procedures and observations in a global software company. In *Proceedings of the 5th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*.

Schwaber, K. (2007). *The Enterprise and Scrum*. Microsoft Press.

Schwaber, K. and Beedle, M. (2001). *Agile Software Development with Scrum*. Prentice Hall.

Siebers, P.-O., Macal, C. M., Garnett, J., Buxton, D., and Pidd, M. (2010). Discrete-event simulation is dead, long live agent-based simulation! *J. Simulation*, 4(3):204–210.

Sommerville, I. (2010). *Software Engineering*. Addison-Wesley Longman, 9th edition edition. ISBN-13: 978-0137053469.

Watkins, C. and Dayan, P. (1992). Q-Learning. *Machine learning*, 8(3):279–292.

West, D. and Grant, T. (2010). Agile development: Mainstream adoption has changed agility. Technical report, Forrester Research.

Williams, L., Kessler, R. R., Cunningham, W., and Jeffries, R. (2000). Strengthening the case for pair programming. *IEEE Softw.*, 17:19–25.

Yin, R. K. (2007). *Case study research: Design and methods*. Sage Publications.