Quantifying the Effect of Using Kanban versus Scrum: A Case Study

Dag I.K. Sjøberg, University of Oslo

Anders Johnsen and Jørgen Solberg, Software Innovation

// Proponents of various processes and methods in the agile and lean communities have made many bold claims about usefulness, but those claims are rarely supported by empirical investigations. Data gathered from more than 12,000 work items collected over three years sheds light on Kanban versus Scrum. //

SUPPOSE THAT YOU'RE a manager in a software development company that has used Scrum for some period of time, but primarily because of its timeboxing, you've decided that Scrum is too rigid for your company. Would Kanban be a better approach? Unfortunately, the literature doesn't yet report the sort of evidence you would want to make this type of shift, such as that by introducing Kanban your company will show improved performance with respect to lead time, quality, and productivity.

Most people would agree that systematically collected empirical evidence should be the basis for important decisions. Data that can back up claims is considered necessary in most scientific or engineering disciplines. However, in the IT industry, little solid evidence generally supports the utility of a method in a given setting. Collecting relevant data is perceived to be too difficult and resource-demanding for most software organizations, and academia seems not to have prioritized this area.

Empirical studies have been conducted on certain agile practices,[1] in particular, pair programming,[2] but we haven't found any scientific study with industry data that compares the effects of using various agile or lean methods on the most interesting variables, such as lead time, quality, and productivity. In response, we report such a study here.

Setting

Software Innovation (SI) is a Scandinavian company that has developed and sold document management products for 28 years. These products are built on the Microsoft SharePoint platform and tightly integrated into the Microsoft Office environment. Currently, approximately 100 developers and specialists work in 10 teams on the products. In total, SI has 330 employees distributed over five countries. The developers and testers are primarily located in Oslo, Norway, and Bangalore, India. SI has partners in 12 countries and has 400 customers.

From 2001 to 2006, SI followed a waterfall process, with an annual cycle of design, implementation, testing, and deployment for each new release. In early 2007, the company carefully examined its development process, which resulted in the decision to introduce Scrum. SI implemented Scrum with the standard elements: cross-functional teams, sprint planning meetings

TABLE 1

Measuring process quality.

	Name	Values
Independent variables	Process	Scrum or Kanban
	Type of work item	Bug or project backlog item (new features, adaptive maintenance tasks, and support tasks —that is, all tasks that aren't bug fixing)
Control variables	Year.quarter	Each quarter from 2009.1 to 2011.4
	Churn	Number of lines added, deleted, or modified
Dependent variables	Lead time	Number of days from "next" state to "ready for release" state on the board
	Production	Number of work items developed per quarter (often called "throughput"[4])
	Productivity	Production per developer
	Productivity 2	Total churn per developer per quarter
	Quality	Number of weighted bugs in the severity levels: blocking (weight 8), critical (4), moderate (2), and minimal (1)

that included estimation of work items using planning poker, daily standup meetings, sprints of three weeks with shippable increments of code (fully tested) at the end of each sprint, and demos in the review meetings. Work status was made visible through automated reports and task boards for all teams.

After a couple of years, the second and third authors (Johnsen, SI's R&D operations manager; Solberg, SI's CTO) felt that Scrum was too rigid, didn't scale, and was unsuitable for maintenance tasks. They also feared that the combination of inaccurate estimates and timeboxes gave longer lead times and what they perceived as "waste," such as Scrum planning meetings, reduced productivity and quality. Consequently, in 2010, the company switched from Scrum to Kanban.

SI has implemented Kanban in the following manner. When work starts on an item, the company attempts to let the item flow through all stages until it's ready for release at a satisfactory quality and as quickly as possible (fast delivery)—that is, without using timeboxes. Furthermore, only a limited number of work items are in progress simultaneously (WIP limit). If the WIP limit has been reached, work doesn't start on a new item until the previous work finishes ("just-in-time" [JIT]). In particular, one item at a time is pulled out from the backlog and designed, as opposed to in the Scrum period, where all the items thought to be finished within a sprint were pulled out simultaneously from the backlog and estimated and initially designed.

Another change from the Scrum period is that SI no longer needs cross-functional teams. It has abandoned start-up meetings focused on estimations about work items. SI still has daily standup meetings under Kanban, but instead of demo meetings at the end of each sprint, status meetings with demos are held once or twice a week, regardless of the progress of the work items being discussed. There's no difference in the quality gates between Scrum and Kanban; all code is equally shippable.

In 2011, the second and third authors realized that "being rigorous with agility just because it is written in theoretical books showing toy examples is of no business value,"[3] so they contacted the University of Oslo to help quantify their hypothesis that the company had benefitted from switching to Kanban with respect to lead time, quality, and productivity. To investigate this hypothesis, we (primarily the first author) analyzed data collected from more than 12,000 work items over three years (2009–2011) by using Microsoft's Team Foundation Server (TFS). Although the company started using Scrum in 2007, we analyzed data only from 2009 onward because SI had overcome the start-up difficulties with agile development by this point. Table 1 shows the independent, control, and dependent variables used in this study.[4]

Lead Time

Merriam-Webster's defines lead time as "the time between the beginning of a process or project and the appearance of its results." *Collins* provides two definitions: "Manufacturing: the time between the design of a product and its production" and "Business: the time from the placing of an order to the delivery of the goods." For a consultancy company contracted with a customer who requests tailored software solutions, this last definition is a useful starting point—in other words, lead time can be defined as the amount of time between the proposal of a new feature or another request and its deployment in the customer's environment.

However, for an in-house development company such as SI that provides two or three releases of its products a year to 400 customers, this definition is unsuitable for two reasons. First, the amount of time a work item remains in the backlog queue before it's put on the board is a function of priority, not whether the company uses Scrum, Kanban, or other development methods. Furthermore, companies that develop and sell products to many customers might propose new features themselves and put them on the backlog before any customers request them. Second, given a policy of two or three releases a year, the result of a work item isn't delivered to the customer immediately after it's finished.

Consequently, the *Merriam-Webster's* definition is more appropriate—that we define lead time as the amount of time that passes from the moment that the development team receives a request to the moment that it completes the work item. This definition is consistent with the one given elsewhere:[5] "The time for an item to move all the way across the board."

Figure 1 shows the average lead time for bugs and project backlog items (PBIs) for each quarter within the periods in which SI used Scrum or Kanban. In the third quarter of 2010, both Scrum and Kanban appear in the data because during that period, some teams in SI still used Scrum, while other teams had switched to Kanban. (To prevent outliers from having a large effect, we removed the work items within the top 10 percent of the longest lead times in each quarter for each type of work item. Consequently, the analysis set was composed of 10,804 work items.)

The figure doesn't show the large variation in lead times. The standard deviation in the Scrum period (17 days for bugs and 20 for PBIs) was much greater than the standard deviation in the Kanban period (five days for bugs and nine for PBIs). As the figure shows, the long lead times for Scrum occurred in 2009. In 2010, the lead time of the Scrum period was at the same level as the lead time in the Kanban period (from 2010: 4.7 days for Scrum bugs versus 5.4 for Kanban bugs, and 8.2 days for Scrum PBIs versus 7.4 for Kanban PBIs).

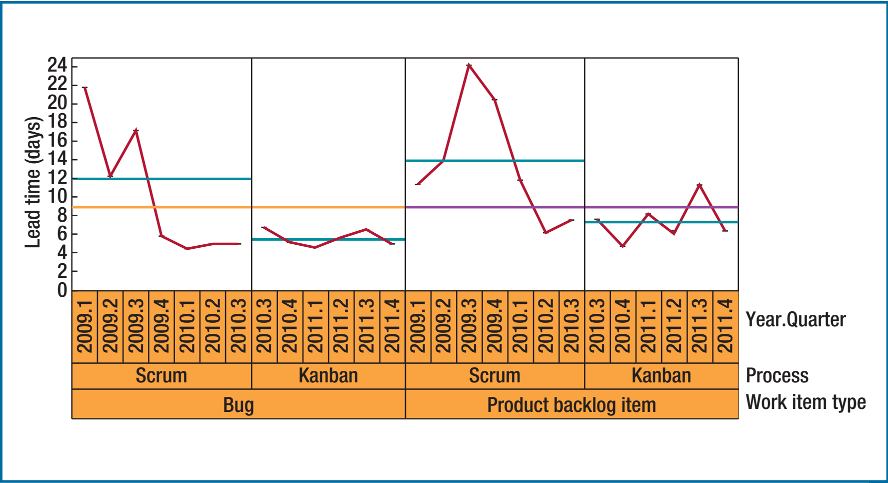

FIGURE 1. Average lead time measured in days by work item type, process, and quarter. The average lead time declined by approximately 50 percent from the Scrum period to the Kanban period. For bugs, the average lead time fell from 12 days for Scrum to five for Kanban. For the project backlog items (PBIs), the lead time declined from 14 to seven days. The orange and purple lines indicate that the bugs and PBIs had the same (weighted) average lead time (nine days) over the whole period. The local top on each third quarter is due to less activity during the summer holiday.

Churn

A change in development performance could be due to aspects other than a change in the formal process—for example, it could be due to changes in the products or the technological environment. However, in this case, products and environment were both very stable. Furthermore, this work assumes that the average amount of work per work item is stable over time. We don't have timesheets that show how many hours each developer or tester spent on each work item. Instead, we use churn as a surrogate measure of effort. Churn is defined as the sum of the number of lines added, deleted, and modified in the source code. A study with exact measures of effort found a correlation of 0.6 between effort and churn for the modification of existing files and a correlation of 0.7 for the development of new files.[6]

Figure 2 shows the average churn for bugs and PBIs. We removed the work items within the top 10 percent of the largest churns within each quarter for each work item type before conducting the analysis to reduce outlier effects. This analysis pertains to the work items that involved changing code, which comprised approximately half of all work items (that is, those with churn > 0). This finding indicates that the size of the work needed to finish a work item might change over time, although the changes aren't dramatic. Only a small correlation exists between churn and lead time at the individual file level (for bugs, Spearman's $\rho = 0.13$, $p < 0.01$; for PBIs, Spearman's $\rho = 0.17$, $p < 0.01$). However, at the quarterly level, a medium, insignificant correlation exists between average churn and average lead time for bugs ($\rho = 0.45$, $p = 0.15$), whereas a large, significant correlation exists for PBIs ($\rho = 0.71$, $p = 0.01$).

Consequently, even if we account for the possible changes over time in the effort needed to finish a work item, as measured by change in churn, the average lead time still declines by

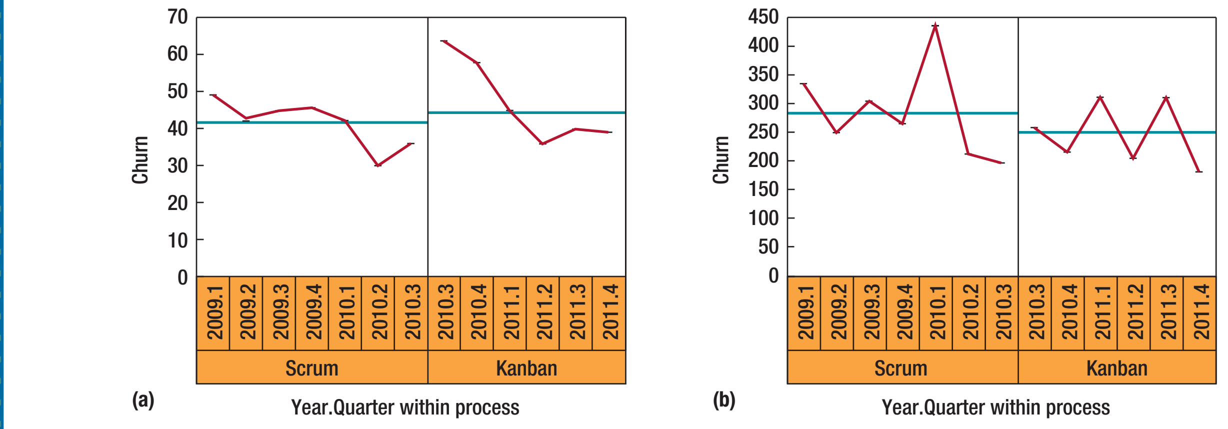

FIGURE 2. Average churn of (a) bugs and (b) PBIs. The average churn for bugs is 6 percent higher in the Kanban period than in the Scrum period, while for PBIs, the average churn is 12 percent lower in the Kanban period than in the Scrum period.

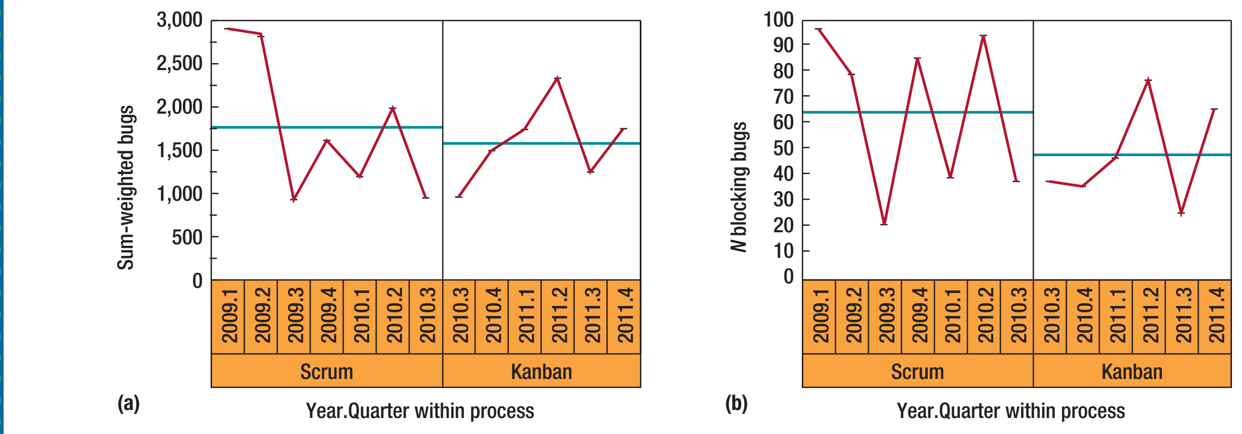

FIGURE 3. Bugs: (a) weighted and (b) blocking. The average number of weighted bugs per quarter fell from 1,774 in the Scrum period to 1,591 in the Kanban period. The most critical bugs, blocking bugs, declined in number even more between the two process periods (from 65 to 48).

approximately 50 percent from the Scrum to the Kanban periods (58 percent for bugs and 40 percent for PBIs).

Quality

According to the ISO/IEC standard 9126, a software system has six major dimensions that pertain to quality: functionality, reliability, usability, efficiency, maintainability, and portability. In this study, we focus on reliability, which is important because bugs in an operational system could lead to undesirable outcomes, such as system crashes or data corruption. To measure reliability, we used the number of bugs, which we classified into four levels of severity, as indicated in the Orthogonal Defect Classification.[7] We gave each bug a weight corresponding to its level of severity; see the last row of Table 1. In SI, bugs are detected both internally (70 percent) and externally by SI's customers (30 percent). Most of the internal bugs are detected the last three weeks before a release because of intense manual and automatic testing in that period.

Figure 3a shows that the average number of weighted bugs per quarter fell from 1,774 in the Scrum period to 1,591 in the Kanban period (that is, by 10 percent). The variability declined much more—the standard deviation

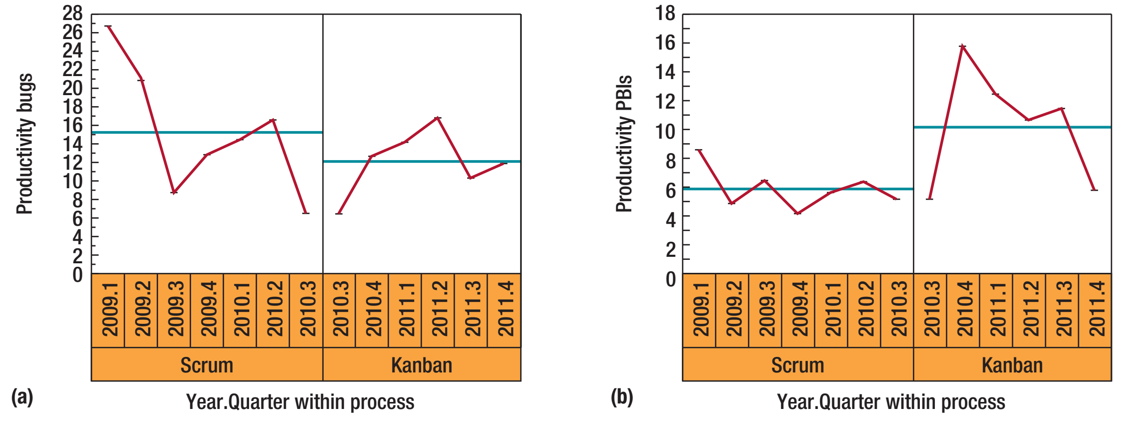

FIGURE 4. Productivity: (a) bugs per developer and (b) PBIs per developer. The number of work items per person, or productivity, decreased from 15.3 to 12.1 for bugs but increased from 5.9 to 10.2 for PBIs.

was 832 for Scrum and 476 for Kanban. The most critical bugs, blocking bugs, declined in number even more between the two process periods (from 65 to 48 [by 26 percent; see Figure 3b]). The standard deviation fell from 31 to 19. The dip in the third quarters is mainly due to less bug fixing during the summer holiday.

More weighted bugs were found in the Scrum period during the two first quarters of 2009—afterward, Scrum was no worse than Kanban. Hence, the reduction in the number of bugs might be independent of whether the process was Scrum or Kanban. In any case, these numbers must be interpreted with caution: an increase in the number of bugs could be due to better bug detection or larger products. Since 2009, SI has employed more and presumably better testers, and the code base of its three products is continually extended.

Production and Productivity

We measure production in terms of the numbers of bugs fixed and PBIs finished. The number of bugs fixed is almost the same over the Scrum (mean per quarter 595, stddev 271) and Kanban periods (mean 580, stddev 164), whereas the production of PBIs more than tripled from the Scrum period (mean 190, stddev 50) to the Kanban period (mean 601, stddev 227).

However, you can usually increase production by employing more people. In the long run, productivity might be more important to a company than its total production. In SI, the number of developers and testers who fixed bugs increased from an average of 40 in the Scrum period to 48 in the Kanban period. The number of people who worked with PBIs increased from 34 to 59. Productivity (the number of work items per person) decreased from 15.3 to 12.1 (by 21 percent) for bugs (see Figure 4a) but increased from 5.9 to 10.2 (by 73 percent) for PBIs (see Figure 4b).

By using churn as an indicator of work item size (see Figure 2), we propose an alternative measure of productivity to validate our results. Specifically, we define Productivity 2 as the total churn divided by the number of developers in each quarter. Figure 5a shows that, for bugs, productivity decreased from an average of 0.46 KLOC (stddev 0.22) for Scrum to 0.41 KLOC (stddev 0.12) for Kanban—a reduction of 11 percent. Figure 5b shows that, for PBIs, productivity increased from an average of 1.28 KLOC (stddev 0.39) for Scrum to 1.55 KLOC (stddev 0.61) for Kanban—an increase of 21 percent. Consequently, if we adjust for work item size measured by churn, we get a reduction in productivity for bugs, but productivity still increases considerably overall from the Scrum period to the Kanban period.

The productivity gain in the Kanban period should also be viewed in light of the growth in the number of employees and the reduction in the number of project managers. In a period during which the number of employees increases, you would expect productivity per employee to decline slightly because of organizational and communication overhead.[8] Furthermore, despite almost doubling the number of developers and testers, SI managed to reduce the number of (costly) project managers from four to three by transitioning from Scrum to Kanban.

Qualitative Evaluation

To complement the quantitative data presented so far, we sought the opinions of the R&D Operations Manager

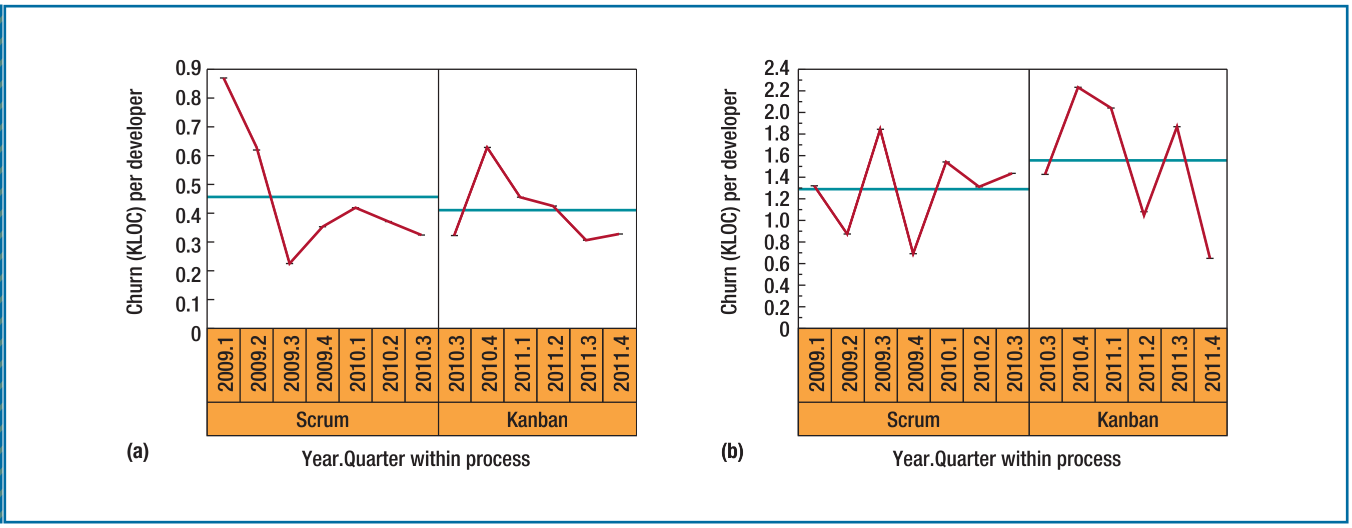

FIGURE 5. Productivity 2: (a) bugs per developer and (b) PBIs per developer. For bugs, productivity decreased from an average of 0.46 KLOC for Scrum to 0.41 KLOC for Kanban; for PBIs, productivity increased from an average of 1.28 KLOC for Scrum to 1.55 KLOC for Kanban.

(second author), CTO (third author), a team leader, and a developer, all of whom have been with SI for more than 10 years. The first author interviewed the team leader and developer for one hour each. All these people clearly favored Kanban over Scrum.

All four perceived the fixed timeboxes in Scrum to be artificial. Work items were frequently underestimated, and developers also had to deal with ad hoc bug fixing, support, and maintenance tasks while working on the items. Regardless, they were supposed to finish the items within the given timebox. In practice, this timeline led to work items that were finished before quality was satisfactory, that were deferred to the next iteration (which required new planning activities), or that weren't finished at all. In the Kanban period, at least all of the items that had been started were finished because developers could focus on one item at a time until it was finished.

In the Scrum period, it was difficult to allocate resources optimally within the sprints—for example, testers tended to have little to do in the beginning of a sprint and too much to do at the end. Much of the sprint start-up meetings were perceived as "waste." In fact, SI had already reduced sprint-planning activities (and abandoned cross-functional teams) by the end of 2009. Two employees mentioned this relaxation of the Scrum rules as an explanation for why the lead time reduced from 2009 to 2010 (see Figure 1).

Did the lack of timeboxes in Kanban lead to insufficient pressure to finish items? The consensus was that the combination of daily stand-up meetings and weekly status meetings, the visibility of the items' status on the board, and the personal ambitions to complete the job constituted sufficient pressure.

After replacing Scrum with Kanban, SI almost halved its lead time, reduced the number of weighted bugs by 10 percent, and improved productivity. Consequently, SI appears to benefit more from using Kanban than from using Scrum. We strongly recommend software companies that face difficulties with effort estimation and interruptions caused by ad hoc bug fixing, support, and maintenance tasks to consider using the lean practice of Kanban.

Nevertheless, as with any kind of study in software engineering, generalizing the results of case studies is challenging. Even though SI had been using Scrum for almost two years before the data analyzed in this study was collected, much of Kanban's indicated advantage might have simply been due to the fact that Kanban was used after Scrum. SI was familiar with agile methods (Scrum) for more than three years before Kanban was introduced, and other aspects, such as SI's technological environment and products, were basically the same in both the Scrum and Kanban periods. Readers should judge for themselves whether they're in a situation similar enough to this company to apply the results of this case study to their own environment.

To provide the agile and lean software community with more evidence on how various processes, particularly Scrum and Kanban, work for different organizations or teams in different contexts, we encourage other companies to collect and analyze data similar to our dataset. Keep in mind that collecting high-quality data might be a challenge. Obtaining reliable information about the performance of a particular process or method requires reliable raw data. In a hectic environment, companies might find it difficult to motivate developers and testers to record information continually about the states of the work items on which they're working. Fortunately, our experience suggests that people become motivated if they observe that the data that they record leads to useful feedback. In addition to feedback on the overall effects of various processes, SI also displays information about the number of bugs detected in the past week and month on monitors in its building's common areas. When visiting the company, partners and customers can then observe the number and trends of bugs in the various products.

Our study compared Scrum with Kanban, but different implementations of these processes might have given different results, which is another reason why our study should be replicated in other environments. For example, a particular characteristic of Kanban is that the WIP should be limited, but Kanban doesn't specify the WIP limit. To test the effects of various WIP limits, we plan to conduct a controlled experiment in which some teams will have lower WIP limits than other teams. We'll then measure the team performance based on the same success criteria described in this article.

Acknowledgments

This work was partly funded by the Research Council of Norway through the project TeamIT, grant 193236/I40. We thank the interviewees at Software Innovation for participating in this study.

References

1. T. Dybå and T. Dingsøyr, "Empirical Studies of Agile Software Development: A Systematic Review," *Information & Software Tech.*, vol. 50, nos. 9–10, 2008, pp. 833–859.
2. J.E. Hannay et al., "The Effectiveness of Pair-Programming: A Meta-Analysis," *Information and Software Tech.*, vol. 55, no. 7, 2008, pp. 1110-1122.
3. C. Ebert and R. Dumke, *Software Measurement*, Springer, 2007.
4. D. Anderson, *Kanban: Successful Evolutionary Change for Your Technology Business*, Blue Hole Press, 2010.
5. H. Kniberg and M. Skarin, *Kanban and Scrum: Making the Most of Both*, InfoQ, 2009.
6. D.I.K. Sjøberg et al., "Quantifying the Effect of Code Smells on Maintenance Effort," IFI Research Report 417, University of Oslo (ISBN 82-7368-381-8), 2012.
7. R. Chillarege et al., "Orthogonal Defect Classification—A Concept for In-Process Measurement," *IEEE Trans. Software Eng.*, vol. 18, no. 11, 1992, pp. 943–956.
8. T.K. Abdel-Hamid and S.E. Madnick, *Software Project Dynamics: An Integrated Approach*, Prentice Hall, 1991.

ABOUT THE AUTHORS

DAG I.K. SJØBERG is a professor of software engineering at the University of Oslo. His main interests are the software life cycle, including agile and lean development processes, and empirical research methods in software engineering. Sjøberg has a PhD in computing science from the University of Glasgow. He's a member of IEEE and ACM. Contact him at dagsj@ifi.uio.no.

ANDERS JOHNSEN is head of product operations at Software Innovation. His main interests are agile and lean development processes, offshoring strategies, and the software life cycle. Johnsen has 11 years of industry experience as a software developer, project manager, and department manager. He's a certified ScrumMaster. Contact him at anders.johnsen@software-innovation.com.

JØRGEN SOLBERG is the CTO of Software Innovation. He's responsible for the technical product road map, strategy, and architecture, as well as managing SI's development centers. Solberg has an MS in engineering from the University of Trondheim. Contact him at jorgen.solberg@software-innovation.com.