Performance-Measurement Framework to Evaluate Software Engineers for Agile Software-Development Methodology

Loay Alnaji[1*] Hanadi Salameh[2]
1. School of Business Administration, Al Ain University of Science and Technology, Al Ain, United Arab Emirates.
2. School of Business Administration, Middle East University, Amman, Jordan
* E-mail of the corresponding author: LOAY.ALNAJI@AAU.AC.AE

Abstract
In spite of the marked benefits agile development brings, it has several shortcomings in quantitative quality measurement, especially in evaluating the performance of individual software engineers. The evaluation criteria for software engineers' performance have been traditionally driven by metrics that don't fit into agile-development principles. This study proposes a measurement framework to evaluate the performance of software engineers. The proposed measurement framework aligns with agile-development core values and principles. This framework can be applied to various agile methods, although the research assumes the use of the Scrum methodology by the software-development team and organization. The proposed framework is simple and doesn't impose overhead on the development team or organization, as it is driven by key agile and Scrum development metrics such as team velocity, escaped-defects rate, defect-cycle time, defect spill-over rate, and individual communication and social skills.
Keywords : Agile Software Management, Software Quality Management, Software Engineers Performance Measure, Scrum

1. Introduction
Agile development brings great emphasis on traits and skills that are not traditional in the world of software development such as team collaboration, mentoring, teamwork, and transferring knowledge; yet, these values were not reflected in evaluation of engineers (Conboy, Coyle, Wang, &Pikkarainen, 2011). In most agile-software organizations, the criteria for performance evaluation still focuses on technical skills and the ability to follow direction, whereas agile development requires greater emphasis on social skills, creative thinking, and self-organization. As a result, performance evaluation often did not reflect the true abilities of team members. This study proposes a performance-evaluation framework for software engineers functioning in an agile-development environment using the Scrum development method. This framework is simple and is driven by agile metrics such as team velocity, cycle-time, and spill-over rate. The proposed framework does not impose any overhead on the development team or on the project, as the required metrics are collected and calculated automatically through software development and tracking tools during sprint planning and review phases.

2. Importance of the Study
Software projects have a history of cost overruns, schedule slips, and quality issues (Standish Group, 2008). For example, only 34% of software projects are deemed successful, costing over $300 billionannually; 49% of budgets suffer overruns, and 62% fail to meet their schedules. According to a global survey of CIOs conducted by IBM in 2008 (Ryman & Reddy, 2009), to overcome these obstacles and mitigate risk an cost overruns, business and technology leaders are directing teams to focus on return on investment and quantified business outcomes to mitigate risk and reduce costs. Software organizations should adopt performance-management programs that align project and team metrics with business goals and objectives. Project and team metrics should be automatically collected from software-development tools to ensure timely decision making and continuous process improvement. By adopting a performance-management program that support metric collection, reporting, and analysis, software-development organizations will gain better insights about their projects, engineers, and processes, and consequently the ability to make better decisions and deliver higher quality software to customers in a timely matter. According to a Businessweek Research Services (2008) study, organizations implementing performance-management practices enjoy a 2.4 times greater market return compared with typical companies. Performance management is the process of continuously measuring software-delivery progress and taking actions to achieve predetermined business goals to align individual and team objectives with organizational objectives (Ryman & Reddy, 2009). For companies to effectively manage the performance of their software-delivery projects, initiatives, and teams, they should have effective performance-measurement programs in place.

Use of agile-software-development methods in place of traditional ones is an example of the software industry shift to address challenges in meeting projects costs, scope, timeline and constant business change. One of the main reasons driving this shift was the inability of old software-development models to respond to constantly changing business requirements (Ylimannela, 2012). In spite of the great benefits agile development brings, it has several shortcomings when it comes to quality measurements. One of the shortcomings of agile development is the lack of quantitative quality measurement, process modeling, and metrics use(Williams, 2007). In addition, Conboy et al. (2011) reported that across all the 17 software organizations they studied, not having and agile-compliant performance evaluation criteria for software engineers and developers was one of managers' main concerns as well as reason for low morale among engineers. Having a software-development team with the impression that they are not evaluated fairly will lead to demotivated engineers, especially when they are passed over for promotion. Repercussions are that 29% of engineers left the organization while 82% of engineers changed their teams (Conboy et al., 2011).

3. Agile Development Methods

Agile methods are a type of iterative and incremental-development methods (Larman, 2004; Larman & Basili, 2003). Agile development is driven by iterative enhancement on software product and development processes (Curtis, 1989). One unique characteristic of agile development is that each iteration is self-contained with activities spanning requirements analysis, design, implementation, and testing(Larman, 2004).Each iteration ends with an iteration release that integrates all software components. The purpose of these incremental releases is to present the customer and receive feedback and refinement on the requirements and features of the system. By doing so, agile methods stratify the main principle for agile-software development: to satisfy the customer through early and continuous delivery of valuable software (Beck et al., 2001). Agile development focuses on value-driven delivery that provides customer with high-priority requirements early to ensure quick and timely market and business presence and penetration. In addition, agile development supports change(Beck et al., 2001). Agile processes harness change for customer's competitive advantage and emphasizes continuous attention to technical excellence and process improvement. Agile development places great importance on people over processes or tools. According to Beck et al.(2001), the quality of people on a project and their organization and management are more important to project success than tools or the technical approach they use.

Finally, agile development is driven by the concept of time-box development, which implies that the length of each iteration is predetermined. Instead of increasing the iteration length to fit the scope, the scope is reduced to fit the iteration length. One of the main differences between agile development and other iterative methods is the length of iteration. With agile methods, the length of iteration ranges between 1and 4 weeks. It intentionally does not exceed 30 days as a risk and complexity mitigation and control approach (Larman, 2004).

3.1 Scrum Software-Development Definition

The Scrum development process is driven by managing iterations that are called sprints. Scrum development is driven by software-development teams that are self-directed and self-organizing (Highsmith, 2002). The team is given the authority, responsibility, and autonomy to decide how best to meet the goal of iteration. Before each sprint, the team plans the sprint and chooses the backlog items to be developed and tested in the sprint (Highsmith, 2002).In Scrum, there are three main artifacts: the product backlog, the sprint backlog, and the sprint burn-down chart (Schwaber &Beedle, 2002). These should be openly accessible and visible to the Scrum team. The product backlog is an evolving, prioritized list of business and technical functionality that needs to be developed into a system and defects that should be fixed. A sprint backlog is a list of all business and technology features, enhancements and defects selected to be addressed in the current iteration (called sprint). For each task in the sprint backlog, the description of the task, its owner, the status, and the number of hours remaining to complete the task are recorded and tracked. The sprint backlog is updated on a daily basis to reflect the number of remaining hours to complete a task. The team has the right to increase or decrease the number of remaining hours for a task as team members realize that the work was under- or overestimated. The sprint burn-down chart illustrates the hours remaining to complete sprint tasks. The team assigns the sprint items among its members. During the sprint, the team writes the code, tests it, and documents the changes.

At the end of the sprint, the team demonstrates the features developed in a sprint-review meeting with stakeholders and the customer. During this meeting, the team might add new backlog items and assess risk, as necessary. Agile development is driven by the concept of time-box development, implying that the length of each sprint is predetermined and the scope for the sprint is chosen to fill its length. Instead of increasing the sprint length to fit the scope, the scope is reduced to fit the sprint length.

3.2 Agile and Scrum Software Sizing and Estimation

Software estimation is a difficult but important part of the software-development process. It is impossible to perfectly predict a software-development project (Cohn, 2004), but there are a number of ways to estimate them. The process is called estimation, not exactimation (Armour, 2002). Agile development focuses on value-driven delivery. As a result, it is important to start development and delivery of high-priority requirements to ensure high return on investment. It is the responsibility of stakeholders to prioritize requirements and the responsibility of developers to provide the estimation (Ambler, 2009). In Scrum development, to be able to accurately project and estimate, the team velocity should be calculated. Three ways to determine team velocity are determining the historical average, running one or more iterations to observe the velocity, or making a forecast. In agile development, software requirements are expressed as user stories. A user story is a system feature or functionality written in normal language. User stories describe problems to be solved by the system being built; they do not describe a solution or use technical language (Cohn, 2010).

In agile development, to estimate as accurately as possible, the team velocity should be calculated. Velocity is the amount of work that a team can complete in each iteration (Schuh, 2005). Usually the amount of work is measured in story points. According to Schuh (2005), story points have proven to be the best measure of software size in agile development in general and in Scrum methodology specifically. Story point is an arbitrary measure used by Scrum teams used to measure the effort required to implement a story (Schofield, Armentrout, &Trujillo, 2013). Story points are a number that tells the development team how hard or easy a requirement story is. Difficulty could be related to risk, new technology, complexity, unknown factors, and effort. An estimate is a probability; hence, it is not possible to make a commitment to a probability, only to a date (Armour, 2002). It is of utmost importance to separate estimates of size from estimates of duration (Cohn, 2006). One way to do this is to use a measure of size that is relative in a unit that people do not associate with duration. Story points are perfect for that.
A popular scale for estimating story points is the Fibonacci scale, which sums the previous two numbers to derive the next number in the sequence. The sequence looks like this: 1, 2, 3, 5, 8, 13,… . The main benefit of the Fibonacci scale is that enough separation exists between the numbers to prevent the team from debating over slight differences. For example, if the scale were 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, team members might debate whether a feature was a 7 or an 8. It is easier for team members to reach agreement if the scale jumps from 5 to 8 (Smith & Sidky, 2009) .Story points don't specify the time when a story will be completed. To estimate the time needed to implement a story, the number of story points for the feature is divided by the team's velocity. In agile, it is proven beneficial to have the entire team make the estimates because when planning the product backlog at the beginning of the project, it is not yet known who will do the work (Cohn, 2004; Schuh, 2005). If the team does not agree on any estimates, they are discussed until agreement is reached. In agile development, the entire team should agree on one estimate for each story, regardless of the team member or engineer who will do the work (Cohn, 2006). Doing so brings two main advantages: first, it emphasizes the team rather than individuals; second, it leads to reduction in the amount of work that has to be done.
Planning poker (Georgsson, 2011) is one of the most popular and reliable ways to estimate the number of story points associated with a feature (Cohn, 2006; Grenning, 2002). Planning poker also helps increase the synergy on the agile team because it requires the involvement of various types of development stakeholders such as programmers, testers, designers, and database experts. Planning poker is used to estimate the story points for each user story in the product backlog. Planning poker is reliable and recommended for several reasons (Cohn, 2006). First, in this game, people with various levels of experience and background are involved and working together to reach a reasonable estimation. Second, during the planning-poker game, the discussion among the team members to clarify any missing or unclear information ensures system and requirements understanding. Group discussions of estimates (Jørgensen & Moløkken, 2002) and averaging team members' estimates (Höst & Wohlin, 1998) lead to better results.

4. Agile-Team Velocity

The proposed evaluation criterion for software engineers in this research is driven by agile-team velocity, hence this section elaborates on the definition of Scrum/agile-team velocity and how it is calculated. Team velocity is the amount of story points a team can complete in an iteration/sprint (Schuh, 2005). Of the three ways to estimate velocity, the first approach is to use historical averages of previous projects, if there are any (Cohn, 2006). However, the difference among projects and teams in technology, domain, product, and tools used should be considered. The second approach is to observe the velocity of one or more iterations or sprints and calculate the velocity based on those. The velocity will be calculated as the average of the number of story points completed by the development team in all completed sprints. The third alternative is to forecast. This is useful when there are no historical averages and there is no possibility to wait for several iterations to be able to calculate velocity average. The second approach is recommended (Cohn, 2006) although in practice it might be challenging to postpone planning and estimation for couple of iterations to calculate team velocity. When using this approach, explain to the customer that this is done to avoid giving inaccurate estimates (Cohn, 2006).

Team velocity is considered one of the most important metrics of an agile team. Velocity provides a realistic idea of how much work (story points) can be completed in an iteration. Hence, by estimating the size of the product backlog in story points, the software team can determine with relative accuracy when project releases and milestones will take place. Without determining team velocity and estimating the backlog based on development-team real experience, project and iteration planning is difficult and less accurate.

5. Framework to Evaluate Software Engineers' Performance

This proposed framework focuses on evaluating the performance of software engineers with agile-development core values and principles. This framework can be applied to various agile methods, although in this research the focus is on the Scrum software-development methodology. This study assumes the use of the Scrum software-development methodology by the software-development team and organization. This framework is driven by key agile and Scrum development metrics such as team velocity, escaped-defects rate, defect-cycle time, defect spill-over rate, and individual communication and social skills. The framework evaluates software engineers from five different perspectives deemed necessary to the success of software engineers and the development team working in an agile-software-development environment. The proposed measurements relate to software engineers' productivity, efficiency, social skills, mentorship and team collaboration, and breadth of knowledge.

5.1 Software-Engineer Development-Productivity Measure

The proposed productivity metrics focus on evaluating productivity in an iteration/sprint. Initial values needed for the performance framework are straightforward. The values shown in Table1 provide the ability to create an initial baseline to measure individual performance.

Table 1. Input Values to the Software Engineer Productivity Measure

Name	Definition
TV	Team Velocity—The number of story points a team is planned to complete during a given iteration.
EPSP	Engineer Planned Story Points—The number of Planned Story Points to be completed by a software engineer during an iteration
ECSP	Engineer Completed Story Points—The number of completed story points by a software engineer in an iteration
TCP	Team Completed Points—Total number of story points planned for the iteration

The metrics shown in Table 1 are collected during iteration planning and at the end of a sprint. The numbers of planned story points to be completed by the development team (TV) along with the story points to be completed by an engineer (EPSP) are defined during sprint planning. The number of story points actually completed by the team (TCP) and an engineer (ECSP) are collected at the end of a sprint. When measuring actual productivity, only measure completed story points for complete used stories. The story points for incomplete user stories should not be measured or counted when measuring TCP. Completed user stories are the ones that passed unit, functional, and user-acceptance testing. Only user stories that have been accepted by the customer should be counted as complete.

The development/coding productivity of a software engineer during a sprint is measured using the metrics shown in Table 2.

Table 2. Software Engineer Productivity Measures

Name	Definition
PPSE	Planned productivity for a software engineer during a given iteration/sprint
APSE	Actual productivity for a software engineer during a given iteration/sprint
PV	Productivity variance: The difference between a software-engineer planned productivity and actual productivity
PI	Productivity index.

At the completion of each sprint, four productivity-measurement metrics are calculated. These are used to measure the development productivity of an individual software engineer

$$PPSE = EPSP/TV * 100 \qquad (1)$$

$$APSE = ECSP/TCSP * 100 \qquad (2)$$

$$PV = APSE - PPSE \qquad (3)$$

$$PI = APSE/PPSE \qquad (4)$$

Planned productivity for a software engineer (a) is the ratio of the assigned story points to the engineer (EPSP) divided by the number of story points planned to be completed by the team in the iteration (TV). Actual productivity for a software engineer per iteration (b) is the ratio of the number of story points completed by the engineer (ECSP) to the number of completed user stories by the team (TCSP). Productivity variance for a software engineer per iteration (c) is the difference between the actual productivity achieved for the engineer (APSE) and the planned productivity (PPSE). The value of variance can be either positive or negative. When negative, this implies the engineers' actual productivity is less than was planned. When positive, it means the engineers' productivity is better or higher than was planned. Productivity performance index per iteration (d) provides an indication of the productivity level per iteration. PI of 1.0 implies that the actual productivity matches the estimated productivity. PI greater than 1.0 indicates work was accomplished by the developer at a rate faster than planned. PI less than 1.0 indicates software-engineer productivity deficiency.

It is very important not to evaluate engineers' productivity in isolation from other iterations as productivity can fluctuate in a release, due to complexity, learning curve, or vacation or sick time. Hence, average productivity should be calculated among sprints. At the end of each sprint, the team average velocity should be recalculated to consider the completed story points in the completed sprint. Similarly, the average productivity level of the software engineer should be calculated, as shown below, where *m* is the number of completed sprints.

$$\text{Avg. PPSE} = \sum_{n=1}^{m} PPSE\ /m \qquad (5)$$

$$\text{Avg. APSE} = \sum_{n=1}^{m} APSE\ /m \qquad (6)$$

$$\text{PV} = \text{Avg. APSE} - \text{Avg. PPSE} \qquad (7)$$

$$\text{PI} = \text{Avg. APSE/Avg. PPSE} \qquad (8)$$

The metrics shown above are good indicators of productivity, but not necessarily a good measure of quality or efficiency; that is, delivery of quality software/code. As a result, it is quite important to use additional measures to provide a comprehensive and realistic measure of software engineers' efficiency.

5.2 Software-Engineer Efficiency Measure

Agile software development puts much emphasis on the development of high value and quality product to customers; hence, it is important to have this reflected on the software engineer-evaluation metric. Agile development uses several metrics to measure quality and efficiency of the software-development team and process, such as the average number of escaped defects, cycle-time, and story points/defect spill-over rate. Those metrics are used in the proposed framework to measure software engineers' efficiency. Using these measures in an individual engineer's evaluation emphasizes to software-development and quality-assurance teams that the functionality released should be of a good quality.

Escaped defects are those defects that have been reported by the customer after a release (AgileBOK.com, 2011b). They are called escaped defects as they have escaped all software-quality processes before a release. It is important to keep the number of escaped defects low, as the fewer they are, the more confidence and satisfied the customer is with the software product, team, and quality. Moreover, a small number of escaped defects reduce the complexity and cost of software engineering. Escaped defects may take from a week to several weeks of effort to correct, including isolation of defect, repair, integration, retest, reconfiguration, and redeployment. Fixing defects in the development iteration is always simpler and cheaper (AgileBOK.com, 2011b).

Cycle-time for software development is the number of days needed between feature or user-story definition and release to the customer, also referenced as software in process (SIP) (AgileBOK.com, 2011a). In other words, cycle-time can be referenced as the time between two successive deliveries to the customer. A shorter cycle indicates a healthier and controlled development process and project.

Story-points/defects spill-over rate defines the number of story points or defects that were committed to be completed in a sprint but did not get completed in the sprint as planned; hence they spilled over to future sprints (Georgsson, 2011). The spill-over rate is the average number of story points that got spilled over among various sprints. The lower the rate, the better the team in estimating its workload and meeting the expected and committed deliverable to the customer.

When evaluating software engineers' performance, it is important to use the productivity measure in addition to agile metrics driven by agile principles and processes. Doing so drives efficiency on all fronts including individual, team, and process. In agile, the metrics shown above are used to measure team performance to ensure continuous process improvement and detection of any problems on the team level. Similarly, these measures can be used to measure software engineers' performance. At the end of each sprint, the metrics—escaped-defects ratio, average cycle-time, and story-points/defects spill-over ratio—on the team level and individual engineer level are shown in Table 3.

Table 3. Definition of Efficiency Metrics for

Efficiency Metric	Definition
Team Defects Escaped Rate	The average number of escaped defects in all the completed sprints
Defect Cycle-Time	The average amount of time it takes to fix a defect in all completed sprints
User Stories Cycle-Time	The average amount of time it takes to complete user stories among all completed sprints
Spill-over Rate for	The average number of story points that have spilled over future sprints in all the competed sprints

The formulas used to calculate the efficiency for the team and individual software engineers are shown in Table 4. At the end of each sprint, the efficiency metrics for the team and the individual software engineer can be calculated using the formulas shown in Table 4. It is important, when evaluating the efficiency metrics of individual software engineers, to keep it in perspective with the team's efficiency.

Table 4: Formulas for Efficiency Metrics for the Software Development Team

Team efficiency metric	Formula	
Defects escaped rate for the team	$= \sum_{=m=1}^{n} NumberofEscapedDefectsperSprint/n$ Where *n* is the number of completed sprints.	(5)
Defect cycle-time for the team	$= \sum_{k=1}^{x} DefectCycleTime\ /x$ where *x* is the number of defects fixed in all completed sprints	(6)
User stories cycle-time for the team	$= \sum_{n=1}^{n} UserStoriesCycleTime\ /n$ where *n* is the number of user stories completed and accepted by the customer in all completed sprints	(7)
Spill-over rate for the team	$= \sum_{m=0}^{n} \text{Number of spilled over story points}/\text{n}$ Where *n* is the number of completed sprints	(8)

5.3 Social-Skill Measure

Agile development in general and Scrum specifically bring great emphasis and much reliance on social skills. Agile practices such as co-location, an on-site customer presence, daily stand-up meetings, sprint retrospectives, and pair programming all are examples of Scrum practices that markedly emphasize the importance of social interaction, communication, and presentation skills (Conboy et al., 2011). Consequently, such emphasis should be reflected in software engineers' performance assessment. Software engineers who demonstrate good communication and social skills with peers and customers should have this boon reflected positively in their performance evaluation.

5.4 Mentorship and Team-Collaboration Measure

Agile methods advocate people interactions, collaboration, mentoring, teamwork, and transferring knowledge among team members. Engineers who demonstrate such team spirit and play the role of mentors to their less senior colleagues deserve to have this trait and effort reflected in their evaluation as well. Implementing such a team-based performance evaluation will foster greater collaboration among team members and ensure a healthier team and development environment.

5.5 Breadth-of-Knowledge Measure

In an agile environment, a developer is required to be competent in a broad range of skills. For instance, a developer needs to be a coder, a tester, an architect, a customer, and a quality-assurance expert (Conboy etal., 2011). Consequently, developers who master a breadth of various roles should be advantaged in their performance evaluation. Breadth of knowledge and expertise along with having a specialty area should be acknowledged when evaluating software engineers.

Although it is important to evaluate software engineers on the individual level, in an agile environment, it is equally or more important to ensure team-level performance. It is quite important to keep sight of the significance of a development team that experiences synergy and is functioning as a whole. This synergy can be guaranteed by developing team-based performance evaluations with indicators tuned to agile attributes. Developing team-based performance evaluations can foster team collaboration and use of agile practices. Creating bonus or performance-acknowledgement programs that are team-based rather than individual based is guaranteed to encourage team collaboration, synergy, and consequently the success of software-development processes and projects. According to Conboy et al.(2011) several software companies have experienced advancement on the team level, such as voluntary contributions and mentoring, due to implementing team-based performance evaluation in which "360-feedback" is used.

6. Conclusion

Providing performance appraisal and evaluation for software engineers in an agile development environment is complex. Agile software environments require software engineers to be more than good coders. They are required to be good communicators, presenters, testers, designers, and have sufficient business knowledge. Traditional software-engineers-performance-evaluation techniques don't reflect agile development principles and metrics. This study presents a framework to evaluate the performance of software engineers using five metrics or measures: productivity, efficiency, social skills, mentorship and team collaboration, and breadth of knowledge.

When evaluating software engineers using the performance metrics proposed, it is important to ensure alignment with the agile development principles and manifesto. The primary objective should be to enhance the development process; the project, the team, and the individual software engineer strive to reach their best state. In addition, it is important to link these performance-management metrics to business value. A potential problem most software organizations experience when using performance-management metrics is to confuse metrics with business goals (Rayman & Reddy, 2009). For example, encouraging developers to find more bugs in the testing phase may cause them to allow more bugs to escape from the coding phase. Metrics are purely a tool to be used to attain business goals. The central business goal for any agile development organization is to deliver high-quality software to the customer frequently. It is critical that all software managers and team members understand the goal so they are driven to prioritize business goals and targets over metrics targets(Rayman & Reddy, 2009).

Moreover, when using these metrics to measure software engineers' performances, it is important to retain sight of the team performance as a whole. For instance, one of the main agile-development principles is to have collective-code ownership among the team so the code can be altered by any team member (Szalvay, 2004). This collective-code ownership provides each team member with the feeling of owning the whole code, which in turn prevents bottlenecks that might hinder the project and team progress when a specialized developer is not available to make a necessary change (Szalvay, 2004). Providing such emphasis to individual engineer performance during the development process might drive engineers a way from functioning as effective team players and contributors.

Finally, implementing a performance-management program in a software-development organization must be accomplished in a focused and incremental manner. Companies should define business goals and link them to the performance metrics to be monitored. These metrics then become the key performance indicators for the development team and its engineers. One advantage of the proposed framework is that it is derived from agile-related metrics collected automatically via software-development tools. In addition, the proposed metrics link directly to the predominant objective of any agile-development environment: the delivery of high-quality software frequently to facilitate high-value delivery and customers business success.

References

AgileBOK.com. (2011a). *Cycle time*. Retrieved January 16, 2014, from http://www.agilebok.org/index.php?title=Cycle_Time

AgileBOK.com. (2011b). *Escaped defects.* Retrieved January 16, 2014, from http://www .agilebok.org/index.php?title=Escaped_Defects

Ambler, S. W. (2009). Agile Requirements Change Management. Retrieved February 11, 2014, from Agile Modeling: http://www.agilemodeling.com/essays/changeManagement.htm

Armour, P. (2002). Ten Unmyths of Project Estimation. Communications of the ACM 45 , no 11:15-18.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., … Thomas, D. (2001). *The agile manifesto.* Retrieved from http://www.agileAlliance.org

Businessweek Research Services. (2008). *The payoff of pervasive performance management. Business Week.* Retrieved December 23, 2013, from http://download.microsoft.com /download/6/6/0/66015B73-82F0-4528-8420-B0B6DD4016AF/pervasivePM_HI_3_13 _09.pdf

Cohn, M. (2004). *User stories applied for agile software development.* Boston, MA: Pearson Education.

Cohn, M. (2006). *Agile estimating and planning.* Upper Saddle River, NJ: Pearson Education.

Cohn, M. (2010). *Succeeding with agile.* Boston, MA: Pearson Education.

Coyle, S., & Conboy, K. (2010). People over process: key people challenges in agile development.

Curtis, B. (1989). Three problems overcome with behavioral models of the software development process. Proceedings of the International Conference on Software Engineering for Real Time Systems (pp. 398–399). Pittsburgh, PA: IEEE.

Georgsson, A. (2011). Introducing story points and user stories to perform estimations in a software development organisation. A case study at Swedbank IT (Unpublished master’s thesis). Umeå University, Umeå, Sweden.

Grenning, J. (2002). Published Articles. Retrieved February 11, 2014, from Object Mentor: http://www.objectmentor.com/resources/publishedArticles.html

Highsmith, J. (2002). Agile software development ecosystems. Boston, MA: Addison-Wesley.

Höst, M., & Wohlin, C. (1998). An experimental study of individual subjective effort estimations and combinations of the estimates. Proceedings of the 20th International Conference on Software Engineering (pp. 332–339). New York, NY: Association for Computing Machinery.

Jørgensen, M., & Moløkken, K. (2002). Combination of software development effort prediction intervals: Why, when and how? Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (pp. 425–428). New York, NY: Association for Computing Machinery.

Larman, C. (2004). Agile and iterative development: A manager's guide. Boston, MA: Addison Wesley.

Larman, C., & Basili, V. (2003). A history of iterative and incremental development. Computer, 36(6), 47–56.

Ryman, A., & Reddy, A. (2009). IBM: Software development and delivery performance measurement and management: Optimizing business value in software. Retrieved January 16, 2014, from http://jazz.net/library/content/articles/insight/performance-management.pdf

Schofield, Joe, Armemtrout, Alan, and Trujillo, Regina, (2013) “Function Points, Use Case Points, Story Points — Observations From a Case Study”; CrossTalk – the Journal of Defense Software Engineering, Vol 26 No 3, pp 23 – 27

Schuh, P. (2005). Integrating Agile Development into the Real World. Hingham, Massachusetts: Charles River Media.

Schwaber, K., & Beedle, M. (2002). Agile software development with SCRUM. Upper Saddle River, NJ: Prentice-Hall.

Smith, G., & Sidky, A. (2009). Becoming agile … in an imperfect world. Greenwich, CT: Manning.

Standish Group. (2008, June 30). Comparative economic normalization technology study. Chaos Chronicles, 12.3.9.

Szalvay, V. (2004). An introduction to Agile software development. Danube Technologies, 1-9.

Williams, L. (2004). A survey of agile development methodologies. URL: http://agile. csc. ncsu. edu/SEMaterials/AgileMethods. pdf

Ylimannela, V. (2012). A model for risk management in agile software development. Retrieved February 3, 2014, from http://www.cloudsw.org/under-review/a6f468c9-4857-4206-96ee-f67df0583d41/file_initial_version