Computer Engineering and Intelligent Systems
ISSN 2222-1719 (Paper) ISSN 2222-2863 (Online)
Vol.6, No.5, 2015

Enhancing Software Team Planning and Delivery Performance using Agile Driven Quality Measures: Approach, Results, and Recommendations

1. Hanadi Salameh, Middle East University, Amman, Jordan, hanadis@hotmail.com

2. Haitham Ali Hijazi, Middle East University, Amman, Jordan, info@hijazi-km.com

Abstract

Implementing a performance-management program in an agile software-development organization is important to retain sight of software team performance as a whole as well as linking performance-management metrics to business value. The evaluation criteria for software engineers' performance have been traditionally driven by metrics that don't fit into agile-development principles. This research implements a measurement metric that aligns with agile-development core values and principles to evaluate engineers in a software engineering firm using the Scrum development method. The research focused on measuring and evaluating the productivity, development efficiency, social skills, team collaboration and breadth of knowledge. Observing the Productivity Index (PI), Actual Productivity for software Engineers (APSE), and Productivity Variance (PV) for individual software engineers and the development team at the end of each sprint helped the development team have a better understanding of its productivity levels in a matter that facilitated planning future sprints. In addition, this productivity measure helped early identification of productivity challenges encountered by several software engineers and productivity identification of productivity challenges encountered by several software engineers.

Keywords: Agile development, software teams, process measurement, scrum, productivity, efficiency, software metrics, software project measurement

1. Introduction

In an agile development environment there is an equal emphasis on technical and non technical skills such as creative thinking, and self-organization; hence, software engineering teams are obliged to be great communicators, moderators, analyzers, planners, and have sufficient business knowledge. Nevertheless, in most agile-software organizations, the criteria for team performance evaluation are still traditional and only put the emphasis on technical skills and the ability to follow orders. Consequently, evaluation metrics of agile development teams frequently does not mirror the accurate and proper abilities of team members. The use of traditional performance measures in agile software firms hinders the implementation of agile practices and processes as these traditional metrics don't enforce or appreciate agile practices.

This study implements and presents the results and recommendation of a performance-evaluation framework proposed by (Alnaji & Salameh, 2015) for software engineers functioning in an agile-development environment. The performance measure used focuses on evaluating the performance of software engineers in light of agile-development core values, principles, and metrics such as team velocity, escaped-defects rate, defect-cycle time, defect spill-over rate, and individual communication and social skills.

2. IMPORTANCE OF THE STUDY

It is a prominent challenge for software organizations to adopt performance-management programs that align project and team metrics with business goals and objectives. Collecting software development metrics is essential to provide insights to project and teams performance which in turn leads to being able to make better decisions and deliver higher quality software to customers in a timely matter. Conboy et al. (2011) reported that across all the 17 software organizations they studied, not having an agile-compliant performance evaluation criteria for software engineers and developers was one of managers' main concerns as well as a reason for low morale among engineers. This study reports on the results of applying a performance measurement metrics for agile software development teams and projects. It implements the proposed framework by (Alnaji & Salameh, 2015) and illustrates the benefits gained by applying this agile metric framework. This study's importance stems from the fact that it demonstrates to software managers and practitioners the steps and procedures needed to apply such measurement system in a matter that facilitates automatically collecting needed input and information from software-development tools to ensure timely decision making and continuous process improvement. By adopting a performance-management program that support metrics collection, reporting, and analysis, software-development organizations will gain better insights about their projects, engineers, and processes, consequently having the ability to make better decisions and deliver higher quality software to customers in a timely matter. This study also demonstrates how performance measures can be reported easily and fairly fast on the team as well as the individual levels facilitating quick productivity enhancements and right-on-time feedback. This is important as 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 (Conboy et al., 2011).

AGILE AND SCRUM DEVELOPMENT

Agile methods are a type of iterative and incremental-development methods (Larman & Basili, 2003; Larman, 2004). Agile development is driven by iterative enhancement on software product and development processes. 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). 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 to 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 processes support and accept change for customer's competitive advantage; in addition, they emphasize continuous attention to technical excellence and process improvement. Agile development is driven by the concept of time-box development, which implies that the length of each iteration is predetermined (Georgsson, 2010). 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 1 and 4 weeks. It intentionally does not exceed 30 days as a risk and complexity mitigation and control approach (Larman, 2004).

The Scrum methodology is one of the mostly used agile development methods (AgileMethodology, 2015). Scrum process is driven by managing iterations that are called sprints. Scrum development is carried out by software-development teams that are self-directed and self-organizing (Georgsson, 2010). The team is given the authority, responsibility, and autonomy to decide how best to meet the goal of a sprint. Before each sprint, the team plans the sprint and chooses the backlog items to be developed and tested in the sprint (Highsmith, 2008). 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). A very important contributor of team success and development progress during a sprint is the scrum master. The scrum master is the person responsible for managing the Scrum project. At the end of a 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.

SOFTWARE AND AGILE PERFORMANCE MEASUREMENTS

Measurement is a very important part of any software organization. In the software engineering and development industry, metrics are a measure of the characteristics of a piece of software or its specifications. In the review of the literature in relation to software metrics, it is apparent that there are various types of metrics. There are some metrics that are focused on measuring business value of the developed features and functionality such as Net Present Value (NPV), Internal Rate of Return (IRR), and Return on Investment (ROI) (Hartmann and Dymond, 2006). Other type of metrics focuses on strategic objectives of the software making; these strategic metrics include net present value, earned business value, running tested features, and real option analysis (Rawsthorne, 2006). Code related metrics include LOC size or LOC to FP, cyclomatic complexity, escaped bugs, code duplication, code coverage, dead code, running tested features, or build creation success rate (Cohn, 2006). Jyothi et al. (2012) stated that object oriented code measurement metrics evaluate object oriented concepts related to methods, classes, complexity and inheritance. These metrics focus on internal object structure in terms of complexity as well as the interaction among objects. Automation metrics focuses on code coverage, number of builds per day, time taken per build, number of failed/successful builds (Oza and Korkala, 2012). Finally, according to (Downey and Sutherland, 2013), there are three types of software metrics: 1. Code level such as running tested features, and code and design quality metrics, 2. Productivity/effort level such as burn-down charts, and project size units and 3. Economic metrics such as earned business value and break-even point processes

There is an increased interest in agile metrics as the metrics used for traditional software development methods are mostly unusable for agile methodology due to each method's different view on developing software. Traditional software development is plan driven compared with agile software development that is result driven with iterative processes. Agile development is team centric by nature and is a big driver for self managed teams, hence having performance metrics that allow the team to have a holistic view of what is happening during project is very important. Especially to account for the many unforeseen tasks assigned to team members that lead to reduction in total velocity of teams and increase in technical debt. Having a dashboard that highlights all major collected metrics to the development team is very important to provide visibility into what is going on in terms of performance and plan (Downey and Sutherland, 2013).

In terms of agile metrics it is a challenge to find quantifiable measures for softer aspects such as motivation, commitment, and satisfaction. Those traits and characteristics are usually measured using opinion polls or surveys to insure regular team feedback (Klein and Canditt, 2008). Furthermore, designing agile metrics requires an evolutionary approach through iteration. Metrics should be designed, tested through practice, redesigned and re-tested (Dingsoyr et al., 2008). Some examples of the agile metrics used and discussed in the literature include velocity, story points, ideal engineering hours, number of user stories done per iteration, backlog size, and number of user stories removed or added from scope per iteration (Benefield, 2008; Cohn, 2010). Velocity although one of the mostly used agile metrics, yet can be very misleading; when using velocity it is very important to pay attention to several factors that might have an impact on team's velocity such as changing team members, difficulty of features, learning curve, toolsets used, etc. Over time velocity tends to stabilize and improve with a stable team working on the same project with the required skilled resources (Hartmann and Dymond, 2006). Furthermore, it is very important to note that velocity is not comparable between teams, and it should only be investigated over time for each team and a whole product or project.

As an agile-value driven metric, Hartmann and Dymond (2006) recommended the use of Earned Business Value (EBV) due to the needed macro level insight it provides into business value of developed functionality. Though according to (Oza and Korkala, 2012), the use of this type of metrics requires the availability of financial data which might not be the case in some software organizations, in such case, using estimated financial values is an option. Sulaiman, et al. (2006) proposed an Agile Earned Value management approach based on the Earned Value Management (EVM) method to manage and track costs to evaluate actual ROI against planned. In addition, there are agile-test related metrics such as unit tests per user story, acceptance tests per user story, functional tests per user story, defect counts per user story, manual tests per story, defects carried to next iteration, or automation percentage (Mahnic and Zabkar, 2008). Some of the agile metrics on the engineer-level include average user stories per day per developer, scope burn up, and cost per iteration (Georgsson, 2011). Oza and Korkala (2012) categorized agile software metrics into seven categories of quantitative metrics. These metrics include engineer specific, strategic measures, test specific, automation maturity, iteration related measures, code specific, and project management specific. Finally, Downey and Sutherland (2013) proposed nine Scrum related metrics that can lead to hyper productive development teams. These metrics include Velocity, Work Capacity, Focus Factor, Percentage of Adopted Work, Percentage of Found Work, Accuracy of Estimation, Accuracy of Forecast, Targeted Value Increase, Success at Scale, and the Win/Loss Record of the Team.

Going through the literature, it is clear that there are many software related metrics that were proposed, adopted, collected and investigated in the software industry. This is justified with the fact that engineering and development metrics are essential as they allow software teams to measure their own performance and adjust their course of action based on facts. With all these software metrics, there is not a one size fits all as these quantitative metrics are not always applicable across all software development methodologies specifically agile. It is worth noting when adopting metrics to only have one metric that is tuned in for a specific period of time. This approach helps building measurement culture as well as building team capability in the specific area of the specific applied metric. Moreover, it is recommended in an agile development environment to use a small set of metrics and diagnostics. Too much collected data and information might hide important trends and distract from intended objectives and valuable process improvements (Hartmann and Dymond, 2006). In terms of best practices when using metrics, it is recommended to follow trends rather than numbers, focus on the team level measure rather than individual, and never report at a level more granular than team and iteration when communicating with upper management. One of the drawbacks noticed when defining and adopting metrics in software organizations is that over time the attention shifts from being organization performance improvement to becoming the interest of individuals (Oza and Korkala, 2012). As a result, when adopting a specific metric, software organizations should understand and not lose sight of the rational, significance, and objective of the collected metric.

OVERVIEW OF THE APPLIED AGILE MEASURE TO EVALUATE SOFTWARE ENGINEERS' PERFORMANCE

This study applies the proposed agile-performance measurement framework proposed by (Alnaji & Salameh, 2015). This research 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 (Alnaji & Salameh, 2015). 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. A summary of the used performance measurements collected and analyzed in this study per (Alnaji & Salameh, 2015) is next.

Productivity Measure for Software Engineers

The development/coding productivity of a software engineer during a sprint is measured using Planned Productivity, Actual Productivity, Productivity Variance, and Productivity Index. Planned productivity for a software engineer (PPSE) is the ratio of assigned and planned story points to the divided by the team velocity (TV). Actual productivity for a software engineer per sprint (APSE) is the ratio of the number of story points completed by the engineer to the number of team's completed story points in the sprint. Productivity variance for a software engineer per sprint (PV) 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 planned. When positive, it means the engineers' productivity is better or higher than planned. Productivity index (PI) provides an indication of the productivity level in a sprint. PI of 1.0 implies that the actual productivity matches the estimated one. PI greater than 1.0 indicates work was accomplished by the developer at a rate faster than planned. PI less than 1.0 indicates a deficiency in software-engineer productivity.

Development-Efficiency Measure for Software Engineers

According to (Alnaji & Salameh, 2015), there are four metrics used to evaluate the efficiency of software teams individually and as a whole. These metrics include defects escaped rate, defect cycle-time, user stories cycle-time, and spill-over rate. Defect escaped rate is the average number of defects that have been reported by the customer after a release; defects cycle time is the average amount of time it takes to fix a defect in all completed sprints; user stories cycle time is the average amount of time it takes to complete user stories among all completed sprints; spill-over rate is the average number of story points that have spilled over future sprints.

Development Team's Soft Skills Measure

Agile development principles evolve around activities that harness social interaction and communication. Agile practices such as co-location, on-site customer presence, daily stand-up meetings, sprint retrospectives, and pair programming all are examples of agile practices that emphasize the importance of social interaction, communication, and presentation skills (Conboy et al., 2011). As a result, per (Alnaji & Salameh, 2015), evaluating the performance of software development team and individuals should take social skills practices in consideration. Software engineers that demonstrate good communication and social skills with peers and customers should have this reflected positively in their performance evaluation.

THE STUDY METHOD AND RESULTS

The proposed performance measurement framework has been introduced and used by a software development team using the Scrum software development method. The monitored team consists of eight software engineers. Individual and team performance were measured and observed over the first four sprints of a release. Conducting individual as well as team-based performance evaluation fosters team collaboration and encourages agile practices. The software engineers had various experience level ranging from Junior, Mid Level, to Senior. 30% of the engineers were Senior with over 10 year's software engineering experience; 40% were Mid Level with 5 to 7 years of software engineering experience; while 30% were Juniors with 1 to 3 years of experience. Each sprint was four weeks long. The initial team velocity (TV) was estimated at 450 story points based on previous experience with similar projects as well as guess-estimating the number of story points to be completed by the team during the initial sprint planning meeting. At the end of each sprint, the team velocity was revised based on the number of actually completed story points by the entire team. Table 1 demonstrates the team velocity of the team and how it was recalculated and adjusted after each sprint based on the number of completed story points in each sprint.

Table 1: Team velocity for at the beginning (planned TV) and end of each sprint (Adjusted TV)

Iterations	**Planned TV**	**Completed Story Points**	**Adjusted TV**
Sprint 1	450	420	435
Sprint 2	435	410	422.5
Sprint 3	423	400	411.5
Sprint 4	412	405	408.5

Results for the Development Productivity Measure

The development productivity metrics were used to measure the productivity level for the individual software engineers as well as the entire team through the four observed sprints. The Actual Productivity of Software Engineer (APSE), Productivity Variance (PV) along with the Productivity Index (PI) were all calculated at the end of each sprint. Table 2 illustrates the measures planned, collected, and calculated within sprint 1. Software engineers with PV and PI less than one were considered low in terms of productivity. This information provided the scum master or project manager with insights on how to enhance the productivity of the team by digging deeper on why such engineers were struggling, consequently, increasing their individual productivity and the team's as well.

Table 2: Productivity Metrics Calculations for the Observed team during Sprint 1

Software Eng. (SE)	Eng. Planned Story Points **(EPSP)**	Planned Productivity for Software Eng. **(PPSE) = EPSP/TV *100**	Eng. Completed Story Points **(ECSP)**	Actual Productivity for Software Eng. **APSE = ECSP/TCSP * 100**	Productivity Variance **(PV) = APSE - PPSE**	Productivity Index **PI = APSE/PPSE**
Team Velocity	**450**					
SE 1	50	11.11	30	7.14	**-3.97**	**0.64**
SE 2	60	13.33	50	11.90	**-1.43**	**0.89**
SE 3	30	6.67	30	7.14	0.48	1.07
SE 4	60	13.33	70	16.67	3.33	1.25
SE 5	40	8.89	20	4.76	**-4.13**	**0.54**
SE 6	50	11.11	50	11.90	0.79	1.07
SE 7	80	17.78	80	19.05	1.27	1.07
SE 8	80	17.78	90	21.43	3.65	1.21
TOTAL	**450**	100	**420**	100	**0.63**	**0.97**

Similar calculations were done and monitored for each of the four sprints. Figure 1 demonstrates the Actual Productivity of the development team during the observed four sprints. Looking at Figure 1, the productivity among the different engineers varied widely in the first two sprints, this can be a result of having the team velocity for sprint 1 guess estimated based on previous projects and team members' experience. As the number of completed sprints increased, the team velocity was recalculated and adjusted to the actual team productivity level. In addition, the use of productivity level measure helped the development team adjust the number of planned story points assigned for each engineer for future sprints based on the individual and team productivity level attained for each completed sprints as well as the complexity and knowledge of the system. This adjustment has lead to stabilizing the productivity level of the team and minimizing the sharp dips and spikes in the productivity achieved in sprints 3 and 4 compared with the one observed in the first two sprints.

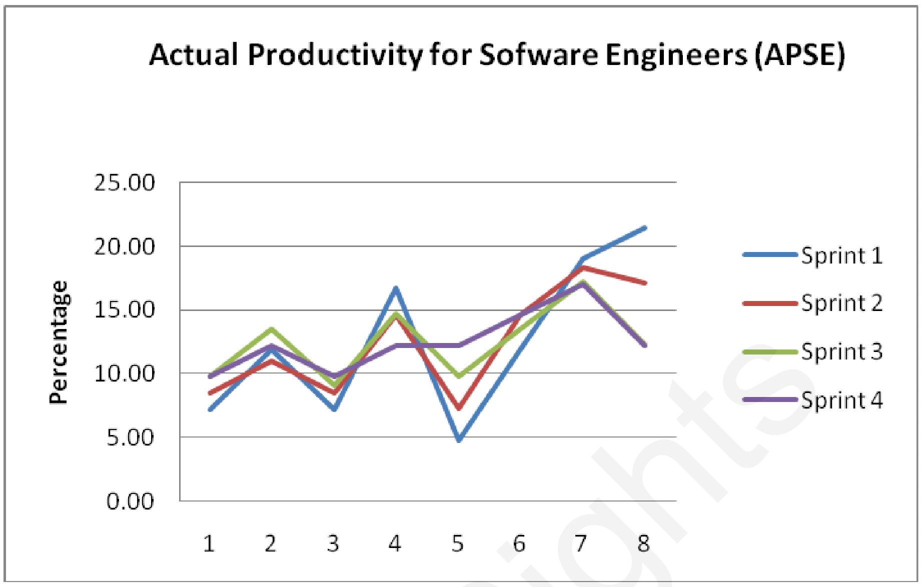

Figure 1: Actual Productivity for Software Engineers (APSE) in 4Sprints

Figure 2 and 3 illustrate the Productivity Index (PI) and Productivity Variance (PV) for the software development team and its 8 software engineers. Similar to the APSE measure, both PI and PV measures for the team had a great level of inconsistency and very sharp peeks and dips among engineers during the first sprints compared with the latest ones. The PV for the team was negative during sprints 1 and 2 but improved to a positive value toward sprints 3 and 4 due to the adjustment of the team and engineers planned story points according to the actual team velocity, number of completed story points, APSE, PI, and PV during the completed sprints.

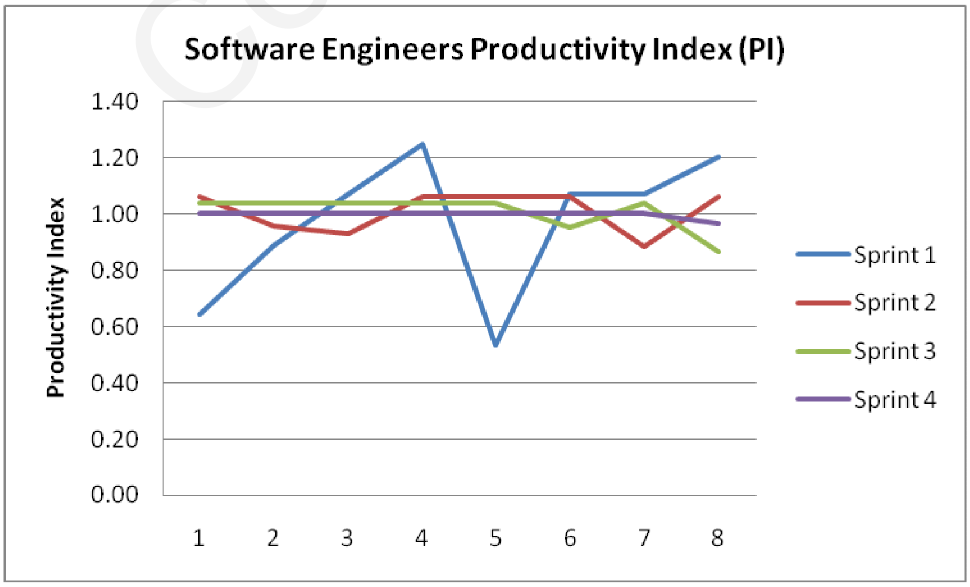

Figure 2: Software Engineers Productivity Index (PI) in 4 Sprints

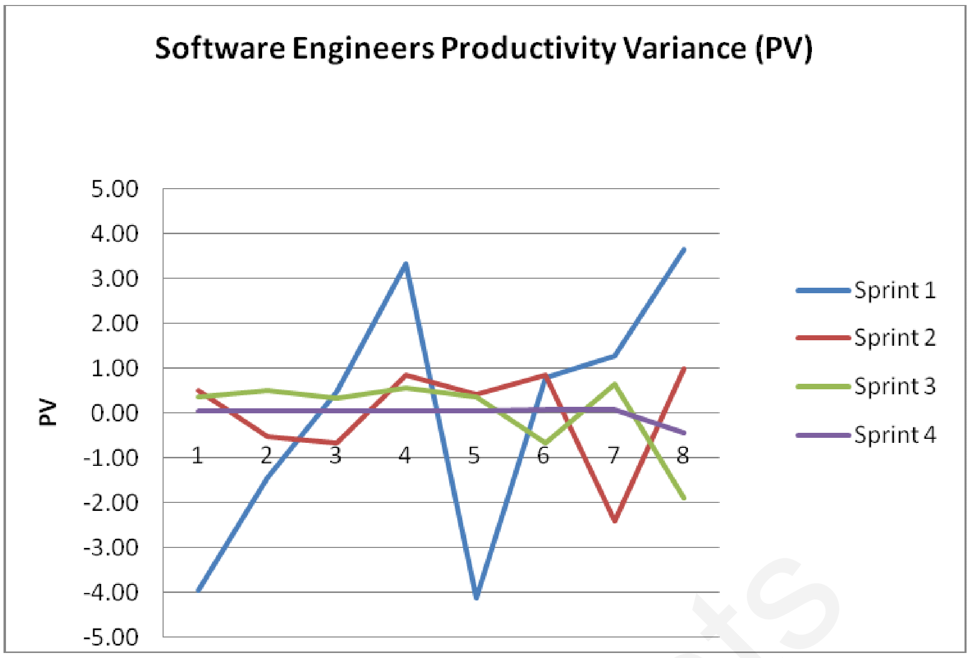

Figure 3: Software Engineers Productivity Variance (PV) in 4 Sprints

Moreover, Figure 4 and 5 demonstrate individual engineers' productivity and how the planned productivity is compared with the actual one in the four sprints. This analysis facilitates identifying early productivity issues with individual engineers to provide the needed support, mentoring, or planning adjustment as needed.

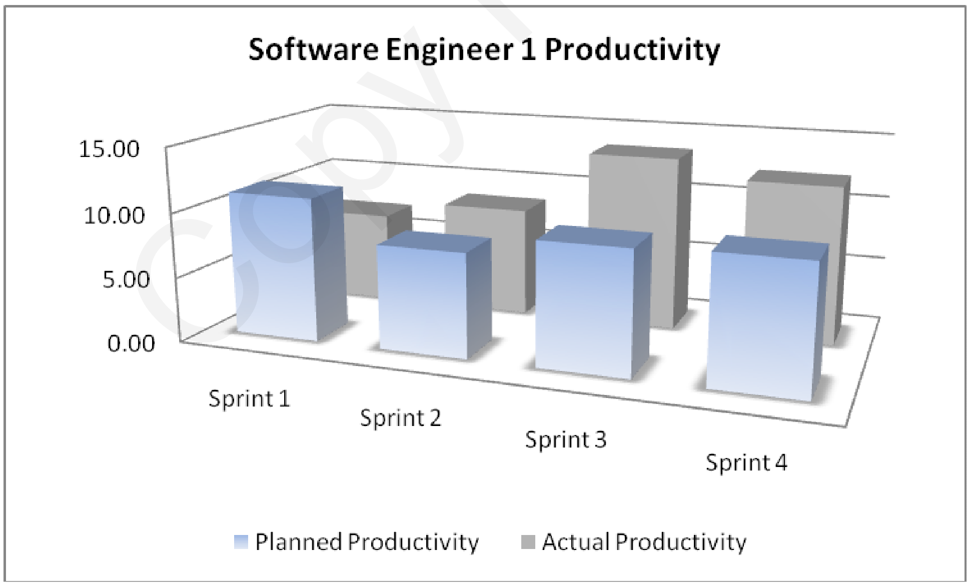

Figure 4: Software Engineers (1) Actual Productivity vs. Planned Productivity in 4 Sprints

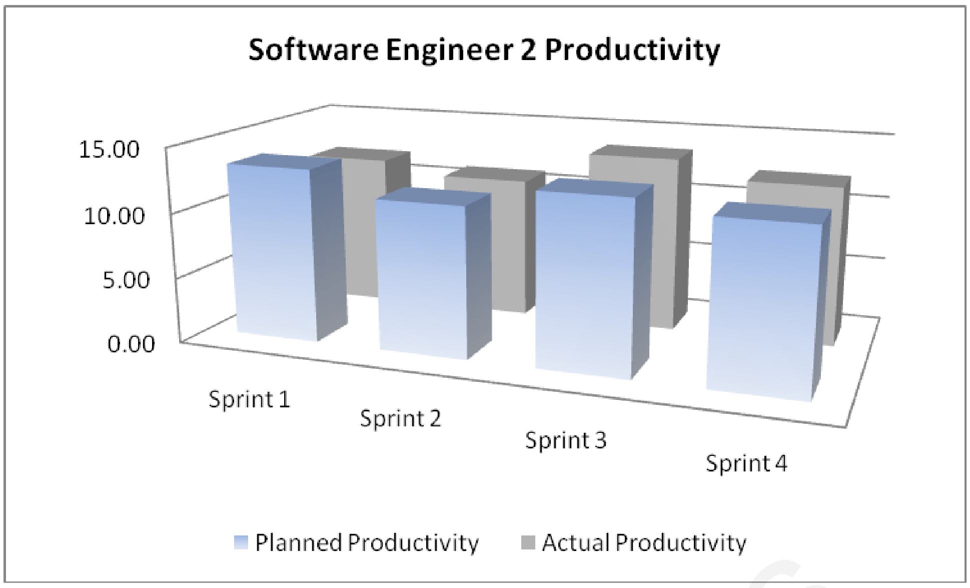

Figure 5: Software Engineers (2) Actual Productivity vs. Planned Productivity in 4 Sprints

In summary, observing the PI, APSE, and PV for individual software engineers and the development team at the end of each sprint helped the development team have a better understanding of its productivity levels in a matter that facilitated planning future sprints. Planning future sprints was reflective and in alignment with the actual team productivity. In addition, this productivity measure helped early identification of productivity challenges encountered by several software engineers. This in turn helped the development team provide the needed training, coaching, and support for those engineers whose productivity was compromised. All this eventually lead to increase in individual as well as overall team productivity which was reflected on better planning, delivery, and team morale.

Results for the Development Efficiency Measure

In measuring the efficiency of the development team and individual software engineers, the metrics of user stories cycle-time, spill-over rate, and defects cycle-time were used. Table 3 lists the metrics observed and collected for the team during the four sprints.

Table 2: Efficiency Metrics Calculations for the Observed team for 4 Sprints

	Total Number of Planned Story Points	*Actual Completed Story Points*	*Customer Accepted Story Points*	*Number of Escaped Defects*	*Defect cycle time (hrs)*	*User Stories Cycle-time (hrs)*	*Spill over Rate*
Sprint 1	450	420	300	30	0	60	30
Sprint 2	435	410	320	25	25	100	25
Sprint 3	423	400	390	10	15	80	23
Sprint 4	412	405	390	5	10	70	7

The user-stories cycle-time is observed to be lower in early sprints, increasing in subsequent sprints. This is justified by the fact that user stories in early sprints tend to be simpler, while complexity of user stories start to increase in later ones. As the team knowledge and experience in the application domain increased, the team's cycle-time rate was observed to improve in spite of increased complexity of user stories. Similarly, for the defect escaped rate, the rate appeared to go down with later sprints as the team's system knowledge and experience seemed to increase enhancing team and quality practices. Figure 6, 7, and 8 demonstrate the team's user stories cycle time, spill-over rate, and defects escaped rate, respectively. Monitoring and measuring the escaped rate of both story points and defects along with the spill-over rate, helped the agile team understand its performance, look into, and analyze the reasons behind escaped defects and story points. In this instance, enhancing team interaction with the customer, having better detailed test cases and quality check practices help reduce the number of escaped defects and escaped story points. In addition, better planning and better team collaboration helped enhance the performance of team members that had higher rate of escaped defects or story points. The team discussed among itself the reasons leading to the obtained rates as part of the team retrospective meeting after each sprint and identified actions needed to mitigate the rate in subsequent sprints. Consequently, this lead to better and more accurate delivery to customer that in turn lead to more customer confidence with the team ability in planning and delivery.

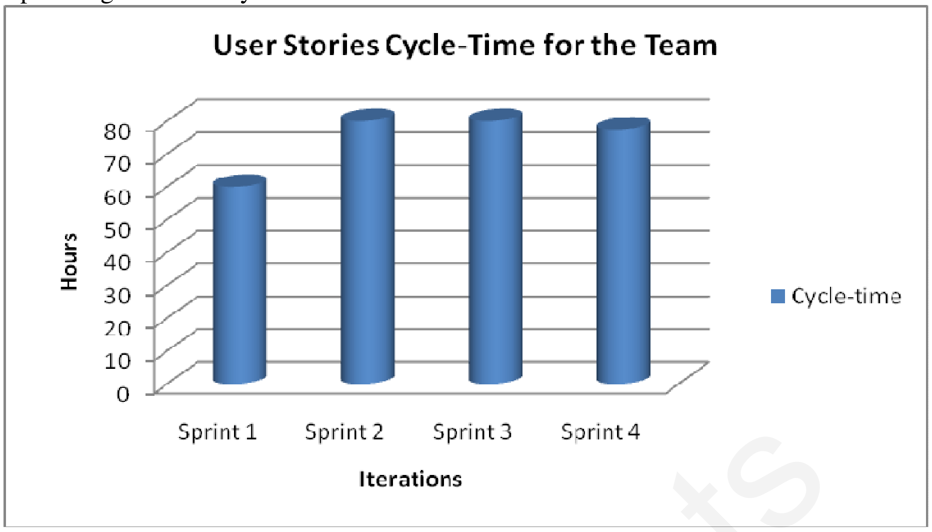

Figure 6: User Stories Cycle-Time for the Development Team over four Sprints

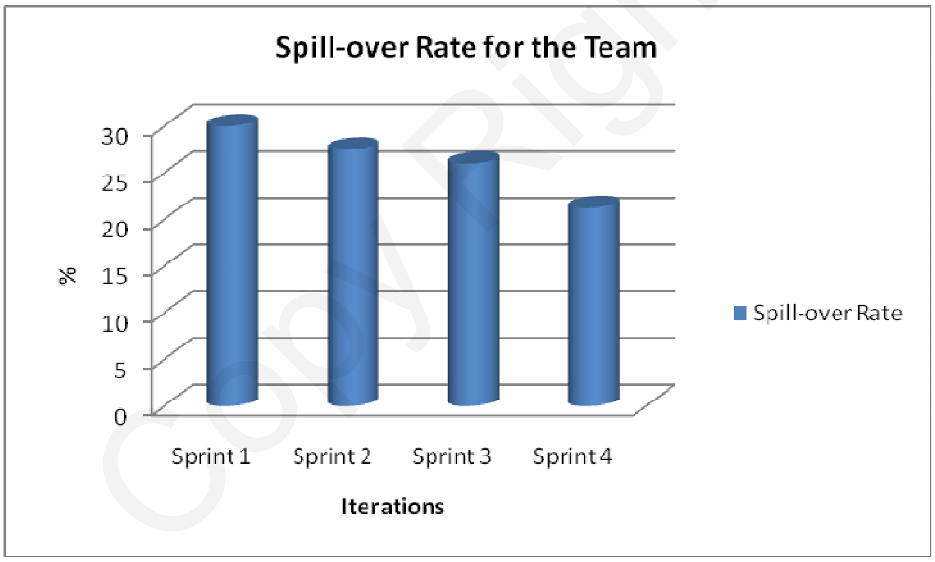

Figure 7: Spill-over Rate for the Development Team over four Sprints

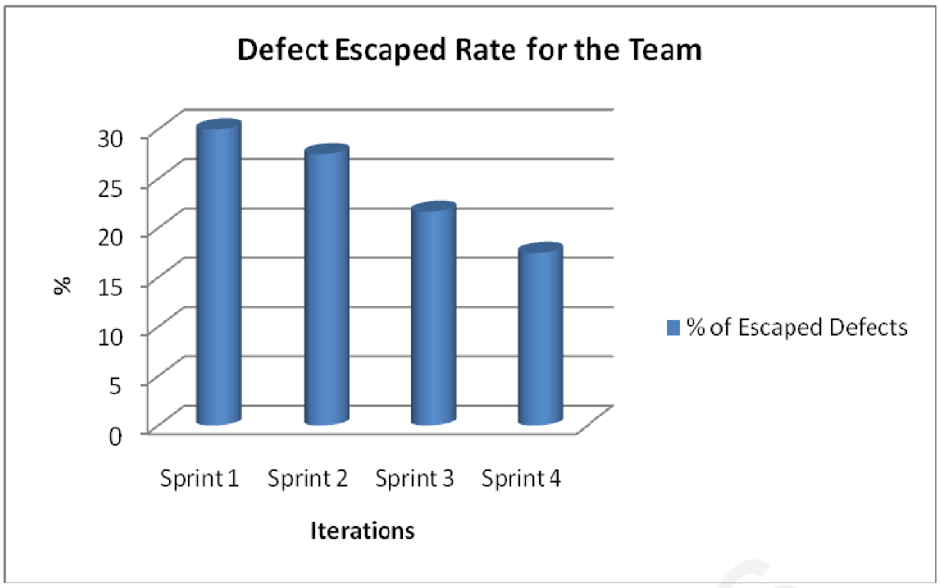

Figure 8: Defect Escaped Rate over four Sprints

Results in terms of Team and Engineers' Soft Skills

Soft skills demonstrated through social-skill, mentorship, team-collaboration, and breadth-of-knowledge, due to their qualitative nature, they were observed by the Scrum master and leveraged when calculating the qualitative measures. In most cases, the plunge in the productivity and efficiency levels on both the individual and team levels was a result of deficiency in one of these areas. Paying attention to those engineers that were relatively low in terms of social skills, experience, and breadth of knowledge was one of the major contributors to enhancing individual and team level performance in subsequent sprints. Furthermore, supporting and encouraging team mentorship and collaboration led to increase in team and individual performance. A mentor for each junior and mid level engineer was appointed for each sprint during the sprint planning meeting. In addition, practices such as code reviewing, extreme programming and dual programmers' assignment were used for complex stories or with less experienced engineers. All these practices helped increase the synergy among the team, consequently increasing the performance of the team as a whole and individuals. As mentioned, it is very important to conduct evaluation and performance measures at the team level to maintain individual performance in perspective to the team's. All the results presented were collected and evaluated for the individual engineers in perspective of team level performance.

CONCLUSION

Providing performance appraisal and evaluation for software teams in an agile development environment is not easy. Agile environments require teams to demonstrate various types of skills including technical, business, communication, collaboration, presentation, testing, and designing skills. Traditional software-engineers-performance-evaluation techniques don't reflect agile development principles and metrics. This study applied and demonstrated the results of an agile driven evaluation framework to evaluate the performance of software teams as proposed by (Alnaji & Salameh, 2015). The framework focuses on a set of five metrics or measures: productivity, efficiency, social skills, mentorship and team collaboration, and breadth of knowledge. The results of the study revealed that the metrics are easy to collect, calculate and analyze as they were driven by agile specific metrics, tools, and method. The input parameters of the framework were easily retrieved from software development and project planning and tracking tools. The results of the study emphasized the ability of monitoring and measuring the performance on the individual and team level, hence, facilitating performance observation and planning and execution adjustment as the software project and development are underway. Also, one of the intangible benefits observed was the collective code ownership among the team due to emphasizing team level performance rather than individual alone. Achieving collective-code ownership prevents bottlenecks that might hinder the project and team progress when a specialized developer is not available to make a necessary change (Szalvay, 2004).

The framework brings so much emphasis on acknowledging soft skills related to social, mentorship, team collaboration as well as breadth of knowledge. However, the framework did not provide solid guidelines on how to monitor, track, and measure such traits and left it to the scrum master or project manager to do so subjectively. Hence, it is recommended to expand and clarify on these areas in future research. Using these sets of metrics as shown in the study results helped understanding and enhancing team performance as a whole as well as on the individual level reflecting on better planning, delivery, and consequently better team moral and customer satisfaction. This aligns with the findings of Conboy et al. (2011) in relation to how software companies can experience advancement on team level performance in terms of voluntary contributions and mentoring as a result of implementing team-based performance evaluation.

REFERENCES

[1] AgileMethodology.org. (2015). Agile Methodology Retrieved April 29, 2015, from http://AgileMethodology.org
[2] Alnaji, L. and Salameh, H. (2015). Performance-Measurement Framework to Evaluate Software Engineers for Agile Software-Development Methodology. European Journal of Business and Management, 7(2).
[3] Beck, K., Beedle, M., Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M. Thomas, D. (2001). The agile manifesto. Retrieved January 20, 2015, from http://www.agileAlliance.org
[4] Benefield, G. (2008). Rolling Out Agile at a Large Enterprise in HICSS'41, Proceedings of the Hawaii International Conference on Software Systems, Big Island, Hawaii.
[5] Cohn, M. (2006). Agile estimating and planning. Upper Saddle River, NJ: Pearson Education.
[6] Cohn, M. (2010). Succeeding with agile. Boston, MA: Pearson Education.
[7] Conboy K., Coyle S., Wang X., Pikkaraine M. (2011). People Over Process:Key People Challenges in Agile Development Software, IEEE, 28(4).
[8] Dingsoyr, T., Dyba, T. and Abrahamsson, P. (2008). A Preliminary Roadmap For Research On Agile Software Development Research. In Proceedings of Agile Conference, pp. 83-96
[9] Downey, S. and Sutherland, J. (2013). Scrum Metrics for Hyperproductive Teams: How They Fly like Fighter Aircraft. Proceedings of the 46th Hawaii International Conference on System Sciences.
[10] Georgsson, A. (2010). Agile methods - a survey of theory and practice. Umea University: Department of Computing Science.
[11] 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.
[12] Hartmann, D. and Dymond, R. (2006). Appropriate Agile Measurement: Using Metrics and Diagnostics to Deliver Business Value. In Proceedings of the Conference on AGILE 2006.
[13] Highsmith, J. (2008). Keeping the Customer in the Product Loop, The Cutter Edge, Retrieved on 15th December 2009 from: http://www.cutter.com/research/2008/edge080909.html
[14] Klein, H. and Canditt, S. (2008). Using opinion polls to help measure business impact in agile development. In proceedings of BIPI 2008. ACM. Leipizig, Germany. pp. 25 – 31.
[15] Jyothi, V. E., Srikanth, K., & Rao, K. N. (2012). Effective Implementation of Agile Practices-Object Oriented Metrics tool to Improve Software Quality. International Journal of Software Engineering & Applications, 3(4).
[16] Larman, C. (2004). Agile and iterative development: A manager's guide. Boston, MA: Addison Wesley.
[17] Larman, C., & Basili, V. (2003). A history of iterative and incremental development. Computer, 36(6), 47–56.
[18] Mahnic, V. and Zabkar, N. (2008). Using COBIT indicators for measuring Scrum-based software development. Wseas transactions on computers. 10(7).pp. 1605 - 1617
[19] Rawsthorne, D. (2006). Calculating earned business value for an agile project. Net Objectives. 3(3). Pages 10. [URL: http://www.netobjectives.com/ezines/ez0607NetObj_CalculatingEBV.pdf]
[20] Schwaber, K., & Beedle, M. (2002). Agile software development with SCRUM. Upper Saddle River, NJ: Prentice-Hall.
[21] Sulaiman, T. Barton, B. Blackburn, T. Agile EVM – Earned Value Management in Scrum Projects, In Proceedings of AGILE Conf 2006, pp. 10
[22] Szalvay, V. (2004). An introduction to Agile software developmen. Technical report. Danube Technologies.
[23] Oza N. and Korkala M. (2012). Lesson Learned in Implementing Agile Software Development Metrics; Proceedings of the UK Academy for Information Systems Conference