Teofilo Kisanji University

TEKU Journal of Interdisciplinary Studies (TJIS)

https://www.teku.ac.tz/tjis.php

ORIGINAL ARTICLE

Received: 12th January 2023

Revised: 12th Nov. 2023

Accepted: 15th Dec. 2023

Published: 29th Dec. 2023

Available at
http://teku.ac.tz/article1.php

Measuring the effectiveness and Quality of Agile Software Design Methodologies

George Lawrence Kinyata

Faculty of Science and Technology, Teofilo Kisanji University, P.O. Box 1104, Mbeya, United Republic of Tanzania.

Correspondence: ict.teku@gmail.com

Abstract

The comparison between traditional and agile methodologies indicates that agile methodologies are a complete shift away from traditional methodologies. The paper investigates how to measure the effectiveness and quality of agile software design methodologies. The key agile metrics include Velocity, Sprint burn-down, and Release burn-up; however, other useful agile metrics include Valued delivered, on-time delivery, Software Size, Project schedule, and software productivity. Despite the various benefits of agile methodologies, various criticisms persist for example; agile is too developer-centric, inefficient in developing large-scale projects, constant user involvement, lack of documentation, and above all misinterpretation of the Agile manifesto. Such criticisms may inevitably have a negative impact on the expected quality of the final product.

Keywords: effectiveness and quality, agile software, software development, design methodologies

1. Introduction

Pinto (2007) defines a project as a collection of interrelated activities with a clearly defined start and end date, carried out in an organized manner to achieve specified objectives. However, to be more specific, software methodologies are concerned with activities that relate to the design/development of information systems. Chemuturi & Cagley (2010) define a software design methodology as an organized approach that identifies a series of steps (stages) that can be followed from the start of the project to the end when the software is finally delivered to the end users. The chosen methodology includes a set of procedures, tools, and techniques that will help the developer in designing the software that meets user requirements.

Several software design methodologies exit; common methods include the waterfall model, the V model, and the Spiral model; these are often referred to as traditional methodologies. Given the dynamic software development environment, traditional methodologies are weak in keeping up with changing customer needs; for example, the waterfall model ignores user involvement during the design stage; user involvement is often limited at the start and at the end of the project. Thus, given changing customers' requirements, especially during the design process, the use of traditional methodologies such as the waterfall model may lead to users not accepting the system once it has been delivered.

Likewise, most traditional methodologies are linear and sequential in nature, with output from the previous stage being input for the next stage, thus the developers cannot move to the next stage unless the previous stage has been completed. As a result, traditional methodologies are time-consuming and costly in terms of the human resources and required money. All in all, this leads to slow software progress, given that they are inflexible and rigid to accommodate emerging user needs. Given the weaknesses of the traditional methodologies, there is a need for dynamic methodologies that can accommodate emerging user requirements' and can cope with the rapid pace of technology. Such methodologies are commonly referred to as agile software design methodologies, common examples include Scrum, Extreme programming, clean room software development, lean software development, future-driven development but to mention a few (Harned, 2018).

2. Methodological Perspectives

The author relied on secondary data, mainly retrieved from industry and academic journals, the main databases used for searching of data included the Google Scholar, EBSCO, Emerald, Oxford Journals OUP, and Pal graves Macmillan,

but to mention a few. The literature search began by using relevant search terms such as "agile software Methodologies'', "effectiveness of agile software Methodologies", "measuring the effectiveness of agile software Methodologies", and "quality of agile software Methodologies," but to mention a few.

3. The Concept of Agile Software Development

Cohen et al. (2004) define agility as the ability to create and respond to changes to succeed in an uncertain and turbulent environment. However, Williams (2007) defines agile software development as a design methodology that encourages an iterative and evolutionary approach to software development. Thus agile methodologies offer a more dynamic approach to software project management; this is in contrast to traditional methodologies which are often linear and sequential in nature. Awa, (2005) argues that agile methodologies are reactive to user requirements, due to their iterative nature, as a result software developers can self-organize, self-prioritise and select which tasks/activities to give more effort within the project.

Gelperin (2008) argues that an agile methodology encourages close cooperation between the development team and other stakeholders to meet user requirements. Due to the iterative and evolutionary nature of agile development, testing early and testing often is highly encouraged thus viruses/bugs can easily be identified, instead of waiting for testing at a predetermined stage. It is thus vital to note that agile methodologies are a complete shift away from traditional methodologies such as waterfall, agile is more dynamic, iterative, and evolutionary in nature.

The following basic principles of Agile Methodologies have been adapted from Beck et al. (2001)

- **Iterative and incremental development,** this implies that software is not developed in a linear/sequential manner, but in a series of short iterations, often between 1 to 4 weeks. Iteration implies that a given stage can be repeated several times; however, in the context of agile iteration is considered to be complete when the given time has expired.
- **People-centric,** the focus is on the end users and developers who are responsible for planning, designing. and delivering the project tasks at each stage of the design.
- **Enabling change,** given the iterative and incremental development approach, agile methodologies embrace user-changing requirements at the end of each iteration; as a result, the end product is modified to meet user requirements
- **Business focus,** this refers to the fact that only features with maximum importance for the business are delivered thus no need to focus on irrelevant features.
- **Regular product and process inspection,** at the end of iteration, the delivered product features are inspected for errors and the team agrees on relevant process improvements.

4. Common Examples of Agile Methodologies

4.1. Scrum

Schwalbe (2012) defines "scrum" as a framework within which people can address complex problems while productively and creatively delivering quality products. The rationale behind scrum is the fact that software development is unpredictable and a complicated process that cannot be effectively planned and successfully estimated; however, scrum is flexible and the iterative approach encourages meeting user requirements. Below are the key stages of scrum, together with their respective key activities

Stage	Key activities
Pre-sprint planning	• Gathering user requirements • Analyze and prioritise requirements
The sprint	• Translate user requirements into relevant design
Post-sprint meetings	• Demonstrate working software to the client • Gather user feedback to improve design • Repeat design processes till user requirements are met

Table 1, Key stages of scrum

4.2. Extreme Programming (XP)

Lindstrom & Jeffries (2004) is based on the idea of iterative development where users are closely involved during the software development lifecycle. XP was first proposed by Beck (1999) who argued that in XP changing requirements is seen as normal and even healthy; likewise, it is acceptable to start writing code early even when the requirements analysis stage is still ongoing. Bell (2005) has listed several rules that need to be adhered to when using XP some of these include re-plan frequently, small releases, developing relevant metaphors, maintaining simple design, code refining, testing early, pair programming, collective ownership, continuous integration, avoid overwork, user involvement, follow coding standards and keep meetings informal.

4.3. Cleanroom Software Development (CSD)

The concept of cleanroom software development was first discussed by Oshana & Linger (1999), who coined the idea of developing software with zero defects, thus extreme steps are taken to avoid contamination of the software. The logic behind CSD is to design software in an extremely clean environment. All workers entering a required environment are supposed to wear special sterilized clothing, thus

ensuring that the software designed is free from any errors and defects.

Kumar Sharma (2013) stresses the need to incorporate CSD with formal methods, statistical testing and reliability growth modelling, with the key objective being error and defect avoidance in contrast to error and defect removal or fault tolerance. It is vital to note that the cost of removing errors/defects increases exponentially the longer they are not detected, thus using cleanroom software development will greatly reduce the cost of developing software.

4.4. Lean Software Development (LSD)

Wang et al. (2012) argue that the idea of lean software development is to eliminate potential waste from the software development process. The concept of lean software development has been extensively discussed by Pernstål et al. (2013), who identified seven lean principles that can be applied to agile software development. These principles include;

- **Optimize the whole**—the focus is on the entire project, not just on specific stages of the project
- **Eliminate waste**—focus on activities that create.
- **Build quality in**, quality is emphasized at every stage, thus the final product should not have defects.
- **Learn first**, it is vital to learn about the process or expected sub processes before work can begin
- **Deliver fast**, quick delivery of sub products will enhance customer value thus ensuring that user requirements do not become out-of-date
- **Engage everyone,** it is vital to ensure that all members of the development team are fully engaged, the team leader has daily feedback, and above ensure that all stakeholders are fully aware about the responsibilities and challenges at hand.
- **Keep getting better,** with each stage the design process should be improved given lessons learnt from previous stage, thus the focus should be to continuously improve in order to gain full control of the project.

4.5. Feature Driven Development (FDD)

The concept of feature-driven development was initially discussed by Palmer & Felsing (2002), the main focus is on the design stages, rather than the entire software development lifecycle. Key stages of FDD include;

- Develop an overall model; this includes developing a logical model of the required system.
- Build a features list; the main activity at this stage is to produce a complete list of system features to support the user requirements
- Plan by feature, the key focus is to plan the order in which the features will be developed, this takes into account task dependencies, risk, complexity, workload balancing and client-required milestone.
- Design by feature, this is an iterative stage that focuses on designing object models based on required features as prioritized by the chief programmer.
- Build by feature, the key focus is to transform the feature design into relevant and appropriate code, code is inspected and unit tested, after successful iteration the completed features are endorsed to be built.

It is vital to note that all the stages of FDD are iterative and adaptive to user needs; as a result, FDD methodology can easily accommodate emerging user requirements and changes; thus giving the developers the capacity to address extremely complex problems that may arise during the course of the software design process.

4.6. Dynamic System Development Methodology (DSDM)

In the early 1990 the United Kingdom government formed a task group to investigate a management approach for harmonizing the design of information systems across government agencies and departments. As a result, the DSDM was agreed upon; DSDM was built on the concept of rapid application development and iteration. According to Kasperek & Maurer (2013), DSDM philosophy is that any project must be aligned to clearly defined strategic goals and focus upon early delivery of real benefits to the end users. The key stages in DSDM include;

- Feasibility study, the viability of the project is determined, this takes into account the economical, technical, legal, schedule and operational feasibility.
- Business study, key activities at this stage include organizing workshops to understand the project and business domain.
- Functional model iteration, this stage involves analysis, coding, and prototyping, prototype results are used to improve the logical models.
- Design and build iteration, key activities include system design; users review functional design and improve it via several iterations.
- Implementation, this is the final stage, where the operational system is handed over to the end users.

Given the iterative and incremental nature of DSDM, it encourages the developers to accommodate emerging customer requirements, instead of building the project in one lifecycle, thus the system is built through several iteration cycles.

Comparison between Traditional and Agile Software Methodologies

Traditional	Agile
The fundamental logic, they are based on meticulous and extensive planning before project commencement	The planning is done in parallel with the actual work.
Often follow linear/sequential stages that need to be strictly followed, thus departure from the original plan is not acceptable.	Allow for iteration, the developer can revisit previous stages to meet emerging user needs.
System testing is often done at a predetermined stage often towards the end of the project	Encourage testing early and testing often; testing can be carried out at any stage.
Teams are typically tightly controlled by the project manager, to ensure agreed schedule and costs are kept in control.	Teams are self-directed and are free to accomplish deliverable as they see fit, as long as they follow agreed rules.
Teams work on a final product/project that should be delivered at a specific time, thus the project doesn't vary from previous clearly defined goals set at the beginning.	Teams constantly assess the scope and direction of the product/project; thus the project path will often deviate from initial project requirements in order to accommodate emerging user needs.

Table 2, Comparison between tradition and agile methodologies *Adapted from* Sureshchandra & Shrinivasavadhani (2008), & Dybå & Dingsøyr (2008)

5. Agile Metrics and Their Effectiveness

The IEEE Standard 610.12-1990 (IEEE, 1990) defines software metric as a quantitative measure of the degree to which a system, component, or process possesses a given attribute. Therefore, the use of agile metrics can be used as a measure of efficiency in order to improve effectiveness and quality of a given software product. The key metrics that can be used in measuring their effectiveness include;

5.1. Velocity

Hayes *et al*. (2014) velocity is the measure of the volume of work or how much working software is delivered in each sprint or in the agreed time. The velocity is often measured as story points completed in each sprint (iteration). It is thus vital to note that the velocity will often be unique given the specific members of the team; this implies that the team should always establish its velocity for the task at hand. Below is an example of a bar chart showing the velocity for a given team.

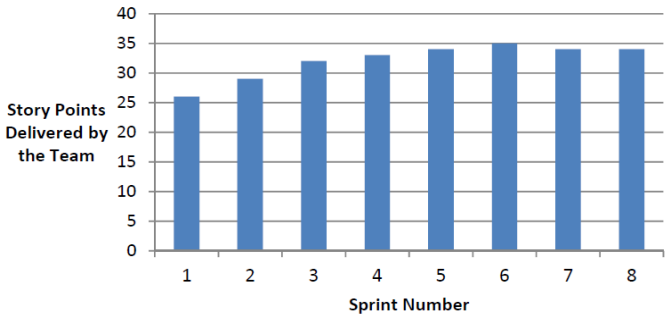

Fig 1*:* Velocity chart

In figure 1, the height of each bar corresponds to the respective sprint, for example sprint number 6 had 35 story delivered by the team, which also happens to be the highest velocity for the team. Thus one cannot expect the team to have a velocity of 42 story points in sprint number 9 (not shown on the chart, it is vital to note that this would be unrealistic given the previous trend of sprints.

5.2. Sprint Burn-down

The sprint burn-down (Deiner, 2012) is a measure of the team's progress in completing their workload; this is often shown on a day-by-day basis given the sprint being carried out. Thus the burn-down rate is the amount of progress depending on the number of items completed in the backlog. The burn-down rate can be graphically shown on the chart during the project development; the chart provides a powerful technique because it provides the means of displaying progress for the team during the sprint. Below is an example of line a graph showing the sprint burn-down rate for a given team.

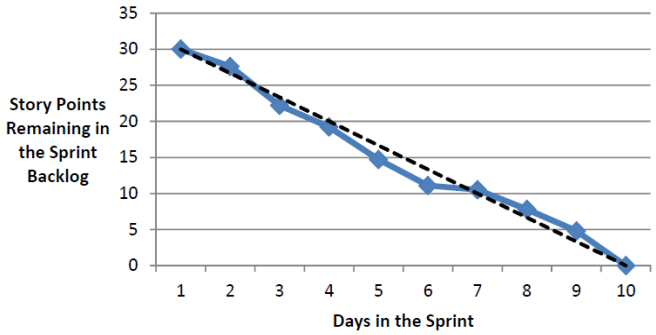

Fig 2: Sprint burn-down chart

The vertical axis of the chart (figure 2) shows the chosen workload for the sprint, while the horizontal axis shows the number of days in the sprint. Overall it is clear that there is a gradual decline in the remaining backlog items that need to be completed (i.e. the story points remaining in the sprint backlog). This is because the thick line running down from left to right with time shown in days on the horizontal axis shows the pace of work being completed; however, the dotted line shows the ideal line against which the thicker line can be compared. Take for instance there is a visible variation in the rate for the achievement of day 6 because the story points remaining on day 7 do not appear to have declined as in the previous days, this is highlighted by the dotted line. Thus the use of the sprint burn-down charts makes it easy to show the completed tasks against remaining task, given the story points to be delivered for the sprint.

5.3. Release burn-up

The release burn-up is the accumulation of the finished work, thus with each completed sprint the delivered functionality grows, and inevitably the release burn-up increases. The chart below (figure 3) shows the release burn-up for a given project.

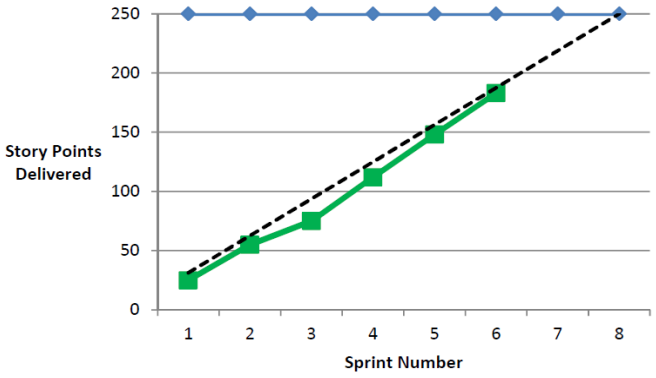

Fig 3: Release burn-up chart

The vertical axis of the chart (figure 3) shows the buildup of delivered value i.e. the story points while the horizontal axis shows the sprint number thus covered in the given time. The expected progress of value delivered is shown by the dotted line running from the bottom left to the upper right of the graph, while the thick line running from the bottom left to the upper right of the graph shows the actual progress of value delivered. Thus, the actual progress can easily be compared to the expected progress; however, the line at the top of the graph is used to show the total number of story points planned for release.

Other Useful Metrics include the following:

Valued delivered

At the end of each sprint, we need to assess the project value delivered; Ramesh et al., (2010) argue that users will often prioritize high-value tasks at the start of the project, thus if you are working on a predetermined project with a definite end in sight, the initial sprints will often have high value and impact, however value and impact will gradually decline as one gets towards the end of the project. Thus an indicator of the value delivered could be the number of story points completed in each sprint.

On-time delivery

Having an agreed date when the software project will be delivered is vital for the success of any organization; otherwise, the majority of clients are not happy with long delivery dates due to several iterative development cycles which in theory and practice will often delay the project. Thus, it is vital to note that the team's velocity should be reasonably steady otherwise wild swings from sprint to sprint will make project planning difficult thus making it difficult to deliver the project on time.

Software Size

The more the story points, the bigger the size of the software, thus it is vital to ensure that the system functionality and capabilities match the user requirements and the total story points accumulated. Thus one can measure the effectiveness of agile methodology by referring to the total story points achieved concerning the functionality and capabilities of the system by tracing these to user requirements.

Project schedule

While using agile methodology each sprint is time-boxed, which implies that the project schedule can be easily estimated, thus the development team can work to maximize performance during each sprint. Thus it is vital that the users effectively communicate the project requirements by doing this will increase the effectiveness and quality of the project.

Customer satisfaction

Due to their iterative and incremental design approach, Agile methodologies are excellent at catering to customer needs, likewise by utilizing acceptance criteria within each user story, the team can effectively understand what the customer needs. Thus this gives the developer the ability to

deliver code more frequently; thus improving the effectiveness and quality of the software Projects.

Software productivity

DeMarco & Boehm (2002) argue that the use of productivity is a more appropriate measure because it compels the team to build software that can effectively contribute to the team's success rather than focusing on metric scores. Thus one cannot underestimate software productivity as a measure of effectiveness and quality; this is because software productivity can be measured by the number of working software that meets user requirements which have been designed by the team.

6. Criticisms of Agile Methodologies

ANSI/IEEE Standard 729-1983 (IEEE, 1990) defines software quality as the totality of features, and characteristics of a product that affect its ability to satisfy given needs, for example, conformance to requirements, free from errors/defects and fit for purpose. Despite the various benefits of agile methodologies; there are some criticisms regarding their use. Inevitably, such criticisms will have a negative effect on the expected quality of the final product.

- ***Agile methodologies are more developer-centric rather than process-centric***

Mohammad (2013) explains that Agile methodologies put more emphasis on having highly qualified developers who have technical and creative skills to work in a team; however, such emphasis has the potential to create a situation whereby technical skills are more valued than customer-centric skills. In the long run, by being too developer-centric, they will compromise the software quality because developers have inadequate time with users to effectively extract user requirements. Likewise, Conboy et al. (2011) points out that agile methodology puts more emphasis on people rather than processes; thus a lack of structured processes during software design and development could lead to costly architectural mistake thus eventually leading to poor quality software.

- ***Agile methodologies are inefficient in developing large-scale projects***

Boehm & Turner (2005) argue that one of the main challenges of Agile methodologies is that they are inefficient for large-scale projects because of the large number of tasks and developers involved, thus it becomes more difficult to manage the expected story points required in each sprint, and above all to coordinate time activities. Livermore (2008) & Lagerberg et al. (2013) also argue that the use of agile methodology in a large-scale project often leads to communication breakdown among the team, primarily due to the scale of tasks and the number of developers involved.

- ***Agile methodologies advocate for constant user involvement/interaction***

The Agile manifesto (Beck *et al*., 2001) advocates for constant user involvement/interaction throughout the software development process; however, too much emphasis on user involvement/interaction is not realistic because according to Mahanti (2007), the average user is not knowledgeable enough regarding technical software design, neither do they have the necessary commitment to be part of a technical software design team. Neither the less having constant user involvement could lead to many viewpoints and conflict thus leading to low-quality software.

- ***Misinterpretation of the Agile manifesto***

Some of the concepts within the Agile manifesto can easily be misinterpreted for example the emphasis on simplicity, however in the real world software development is complex at best, because it involves modelling real life scenarios; thus, in reality, "simplicity" is not enough in developing quality software.

- ***Lack of documentation***

Given the complexity of software development and higher user involvement, the lack of proper and effective documentation advocated for by Agile methodologies is bound to lead to poor- quality software. Various industry and academic experts such as Ambler (2009) and Nerur et al. (2005) all emphasize the need for effective documentation because it is easier to plan, monitor, manage and control projects; above all, effective documentation gives developers increased ability to identify problems whenever they arise.

7. Conclusion

The key agile metrics include Velocity, Sprint burn-down, and Release burn-up; however, other useful agile metrics such as Valued delivered, on-time delivery, and Software Size; however, Project schedule and software productivity are arguably more realistic and practical given their impact on customer satisfaction. It is also vital to note that the Agile manifesto actively promotes the need to put users first over and above any agreed contract and plan. With all its weaknesses discussed on this paper, if well applied, the agile methodologies will inevitably improve the effectiveness and the quality of the software project delivered.

Funding
None

Conflicts of Interest

The author declares that there are no conflicts of interest regarding the publication of this article because no funding agent was involved.

Acknowledgments
I would like to thank Prof. Peter Chonjo, Paschal Sanka, Peter Tarimo, Robert Kwene, Laurent Juma, Bernard Msuya, Andrew Mwanga, Epaphras Mgina and Haikaeli Bakuju for their support and encouragement.
Finally, I am grateful for Prof. Elia Mligo, for his reviews, comments and corrections when writing the paper.

References

Ambler, S. W. (2009). Scaling agile software development through lean governance. *Proceedings of the 2009 ICSE Workshop on Software Development Governance, SDG 2009*, 1–2.

Awad, M. A. (2005). A comparison between Agile and Traditional Software Development Methodologies. In *The University of Western Australia*. The University of Western Australia.

Beck, K. (1999). Extreme Programming Explained: Embrace Change. *XP Series*, *c*, 224.

Beck, K., Beedle, M., Bennekum, A. Van, Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., & Thomas, D. (2001). *Manifesto for Agile Software Development*. The Agile Alliance.

Bell, D. (2005). Software Engineering for Students. In *Software Engineering for Students*. Addison-Wesley.

Boehm, B., & Turner, R. (2005). Management Challenges to Implementing Agile Processes in Traditional Development Organizations. *IEEE Software Development*, *22*(5), 30–39.

Chemuturi, M., & M., Cagley, T. (2010). Mastering software project management: best practices, tools and techniques. In *Mastering software project management: best practices, tools and techniques* (Issue 1). J. Ross Publishing.

Cohen, D., Lindvall, M., & Costa, P. (2004). An Introduction to Agile Methods. In *Advances in Computers* (Vol. 62, Issue C, pp. 1–66).

Conboy, K., Coyle, S., Wang, X., & Pikkarainen, M. (2011). People over process: Key challenges in agile development. *IEEE Software Development*, *28*(4), 48–57.

Deiner, H. (2012). Test Management: Measuring Quality in the Agile Enterprise. *TechTarget Inc.*

DeMarco, T., & Boehm, B. (2002). The agile methods fray. *IEEE Software Development*, *35*(6), 90–92.

Dybå, T., & Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. In *Information and Software Technology* (Vol. 50, Issues 9–10, pp. 833–859).

Gelperin, D. (2008). Exploring agile. *Proceedings of the 2008 International Workshop on Scrutinizing Agile Practices or Shoot-out at the Agile Corral - APOS '08*, 1–3.

Harned, D. (2018). *Hands-On Agile Software Development with JIRA: Design and manage software projects using the Agile methodology*. Packt Publishing.

Hayes, W., Miller, S., Lapham, M. A., Wrubel, E., & Chick, T. (2014). *Agile Metrics : Progress Monitoring of Agile Contractors* (Issue January). Software Engineering Institute.

IEEE. (1990). IEEE Standard Glossary of Software Engineering Terminology. In *Office* (Vol. 121990, Issue 1, p. 1).

Kasperek, D., & Maurer, M. (2013). Coupling Structural Complexity Management and System Dynamics to represent the dynamic behavior of product development processes. *SysCon 2013 - 7th Annual IEEE International Systems Conference, Proceedings*, 414–419.

Kumar Sharma, H. (2013). E-COCOMO: The Extended COst Constructive MOdel for Cleanroom Software Engineering. *Database Systems Journal*, *IV*, 3–11.

Lagerberg, L., Skude, T., Emanuelsson, P., Sandahl, K., & Stahl, D. (2013). The impact of agile principles and practices on large-scale software development projects: A multiple-case study of two projects at Ericsson. *International Symposium on Empirical Software Engineering and Measurement*, 348–356.

Lindstrom, L., & Jeffries, R. (2004). Extreme Programming and Agile Software Development Methodologies. *Information Systems Management*, *21*(3), 41–52.

Livermore, J. A. (2008). Factors that significantly impact the implementation of an agile software development methodology. *Journal of Systems and Software*, *3*(4), 31–36.

Mahanti, A. (2007). Challenges in Enterprise Adoption of Agile Methods - A Survey. *Journal of Computing and Information Technology*, *14*(3), 197–206.

Mohammad, A. H. (2013). Agile Software Methodologies : Strength and Weakness. *International Journal of*

Engineering Science and Technology (IJEST), *5*(03), 455–459.

Nerur, S., Mahapatra, R., & Mangalaraj, G. (2005). Challenges of migrating to agile methodologies. *Communications of the ACM*, *48*(5), 72–78.

Oshana, R. S., & Linger, R. C. (1999). Capability Maturity Model software development using Cleanroom software engineering principles-results of an industry project. *System Sciences, 1999. HICSS-32. Proceedings of the 32nd Annual Hawaii International Conference On, Track7*, 10 pp.

Palmer, S. R., & Felsing, M. (2002). A Practical Guide to Feature Driven Development. In *… Guide to Feature Driven Development*. Prentice-Hall.

Pernstål, J., Feldt, R., & Gorschek, T. (2013). The lean gap: A review of lean approaches to large-scale software systems development. *Journal of Systems and Software*, *86*(11), 2797–2821.

Pinto, J. K. (2007). Project Management Best Practices: Achieving Global Excellence (Kerzner, H.; 2006). *IEEE Transactions on Engineering Management*, *54*(2), 391–392.

Ramesh, B., Cao, L., & Baskerville, R. (2010). Agile requirements engineering practices and challenges: an empirical study. *Information Systems Journal*, *20*(5), 449–480.

Schwalbe, K. (2012). Managing a project using an agile approach and the PMBOK® Guide. *Proceedings of the Information Systems Educators Conference ISSN*, 1435.

Sureshchandra, K., & Shrinivasavadhani, J. (2008). Moving from waterfall to agile. *Proceedings - Agile 2008 Conference*, 97–101.

Wang, X., Conboy, K., & Cawley, O. (2012). "Leagile" software development: An experience report analysis of the application of lean approaches in agile software development. *Journal of Systems and Software*, *85*(6), 1287–1299.

Williams, L. (2007). A Survey of Agile Development Methodologies. In *Univercsty of the West England*.

Author's Biography

George Lawrence Kinyata, is Assistant Lecturer of Computer Science/IT at the Faculty of Science and Technology, Teofilo Kisanji University in Tanzania; he is an expert in Computer Accounting Packages (QuickBooks & Tally), Database Design/Administration, Artificial Intelligence, Operating Systems and Systems Analysis/Design. Kinyata has a Master of Science Degree in Information Systems Management and a BSc (Hons) Degree in Business Information Technology all from London South Bank University (UK).