DOI: https://dx.doi.org/10.4314/gjpas.v25i1.10

GLOBAL JOURNAL OF PURE AND APPLIED SCIENCES VOL. 25, 2019: 71-79
COPYRIGHT© BACHUDO SCIENCE CO. LTD PRINTED IN NIGERIA ISSN 1118-0579
www.globaljournalseries.com, Email: info@globaljournalseries.com

AN EXPLORATIVE SURVEY OF FORMAL AND AGILE SOFTWARE DEVELOPMENT METHODS

OYONG, S. B. AND EKONG, V. E.

(Received 17 September 2018; Revision Accepted 12 December 2018)

ABSTRACT

This research work explores the trends in formal and agile software development methods. Software development has evolved, from the era of "code and fix" to methods categorized as "heavyweight" and "lightweight". The heavyweight methods are championed by the waterfall method, while agile methods are considered the lead in lightweight methods of software development. Both methods have proven records of successes and failures. Bridging the divide between them and harmonizing their symbiotic properties has the synergy of creating beneficial and more robust methodology with complementary advantage which is termed ambidexterity.
Ambidexterity allows for a high level approach of selecting a methodology on the basis of the problem requirements, and coordinates their independent processes complementarily without conflict.

KEYWORDS: Global Software Development, Agile Software Development, off Shoring, Near-shoring, Distribution

1.0 INTRODUCTION

Many software development methods have been created in the past four decades, and utilized in the software industry. Each method has different features and characteristics that distinguish one from another. These methods can be classified into two groups: the heavyweight methods, also called traditional methods, which focus on comprehensive planning, heavy documentation and big design up-front (Boehm and Turner, 2003; Fruhling and De Vreede, 2006). The lightweight methods concentrate on the software development team and their interactions, rather than on the required processes and tools. It also focuses on developing working software that evolves from intense customer collaboration (Bech, Beedle, Bennekum, Cockhurn, Cunningham, et al., 2001).

The traditional methods are still widely used in the software industry because of their straightforward, methodical, and structured nature; they have proved their abilities to provide high assurance, stability, and productivity. However, they have a number of shortcomings, which include (Boehm, 2002; Boehm and Turner, 2003; Brooks, 1975; Schach, 2004):

- Slow adaptation to constantly changing business environments
- A tendency to be over budget
- They always work behind schedule
- Delivering fewer features and functions than
- specified in the requirements
- They also usually need a complete set of requirements prior to design,
- Resulting in vague user specifications.

As a remedy to the short comings of the traditional methods, Agile software development methods were created and used by practitioners. A number of Agile development methods evolved, and they include (Goldman, Kon, Silva, Yoder, 2004; Cockburn, 2006; High smith and Cockburn, 2001):

- **A**daptive software Development (ASD)
- Agile Unified Process (AUP)
- Crystal Methods
- Dynamic Systems Development Methodology (DSDM)
- Extreme Programming (XP)
- Feature Driven Development (FDD)
- Kanban
- Lean Software Development
- Scrum
- Scrumban

By their characteristics, they focus mainly on:

- Iterative and incremental development
- Customer collaboration
- Software product delivery occasioned by a light and fast development cycle

Oyong, S. B., Department of Computer Science, Faculty of Science, University of Uyo, Akwa Ibom State, Nigeria.
Ekong, V. E., Department of Computer Science, Faculty of Science, University of Uyo, Akwa Ibom State, Nigeria.

- Adaptation to changing customer and business requirements.

The potential benefits of Agile methods notwithstanding, many organizations are reluctant to throw away their traditional methods, and jump into Agile methods. Their reluctance is as a result of several issues including:

- Agile methods reduce documentation and rely on tacit knowledge
- These methods have not been sufficiently tested for mission-critical projects
- There is the belief that Agile methods are good only on small and medium scale projects, and not large scale projects.
- A concern that Agile methods can be successful only with talented individuals
- Agile methods are not adequate for high and stable projects

Although many positive aspects of Agile methods have been published, there are few empirical studies on the negative aspects of Agile methods. The negative aspects of Agile methods imply that there are issues, problems, and challenges faced in developing high quality software products using these methods. Identifying the problems, issues, and challenges of Agile methods should be more beneficial to organizations propagating them than merely showing their positive benefits.

More so, since only the positive aspects of Agile methods are receiving publicity, it is not clear whether Agile methods can provide end users with the desired quality, in a timely manner, on large-scale and mission-critical projects. Therefore, the need to identify the issues and challenges of Agile methods and assess the possible application of Agile methods on large-scale and mission-critical projects cannot be over emphasized. It is intended that through an explorative research approach we can uncover these issues.

1.1 STATEMENT OF THE PROBLEM

The few empirical field studies of the negative aspects of Agile software development methodologies have failed to identify how the methods can still be useful to organizations, and have not assessed their possible application for large-scale and mission-critical projects.

1.2 AIM

The aim of this research work is to explore possible ways of using Agile methods in the development of large-scale and mission-critical projects

1.3 OBJECTIVES

To propose ahybrid framework based on ambidexterity that would take advantage of their comparative strengths.

2.0 LITERATURE REVIEW

The review of both failures and successes in the literature will be beneficial in identifying the possible success factors in Agile software development projects as failure can contribute to the understanding of how to avoid certain serious pitfalls that are critical to the success of a project (Chow and Cao, 2007). Cohn and Ford (2003) studied problems in transitioning organization to ASDM processes, while Larman (2004) discussed in detail mistakes and misunderstandings occurring in ASDM projects.

Boehm and Turner (2005) emphasized on management challenges in implementing Agile projects; whereas Nerur, Mehapatra, Mangalaraji (2005) covered problems not only in management aspects, but also in people, process, and technology dimensions of migrating to Agile projects.

2.1 TRADITIONAL SOFTWARE DEVELOPMENT METHODS

The early stages of software development can be summarized as “code and fix, code-some-more, fix-some-more” (Fowler, 2005; Leffingwell, 2007). This was the first generation in the history of software development methods. The fundamental concept of the scheme was to write code first without considering pre-planning and pre-designing, and to fix bugs later. This method worked well for small scale and relatively simple projects. However, as the size of projects increased, developers realized that they were spending more time in fixing bugs and writing code; thus reducing efficiency and predictability of software development.

The traditional software development concept was borrowed from the engineering discipline, which puts heavy weights on precise planning. Engineering discipline –based development method was more or less plan driven where the documentation of a complete set of requirements preceded architectural and high-level design, development and implementation (Awad, 2005). These methods require extensive planning, codified processes, and rigorous reuse (Boehm, 2002). This plan also works best when developers know all the requirements in advance, and when requirements are relatively stable (Hickey and Davis, 2004; Schach, 2004).Due to these factors, the methods came to be known as heavyweight methods and are also considered traditional software development methods (TSDMs).

Practitioners and academia alike revisited alternative ways of developing software such as iterative, incremental development, and close customer involvement (Royce, 1970). Agile software development methods (ASDMs) were some of the methods that fitted into these characteristics, and even more (Goldman et al, 2004; Cockburn, 2006). Agile methods have had inroads into traditional software development methods(TSDM), even at areas where it was thought impossible. Such areas include (Mockus and Herbsleb, 2001):

- Interdependencies among work items that are distributed
- Difficulties in coordination
- Conflicting implicit assumptions
- Communication challenges,

The link between Agile methods and TSDM is needed for frequent communication (Nisar and Hameed, 2004; Simons, 2002). After all, what are new about Agile methods are not the practices they use, but their recognition of people as the primary drivers of project success, coupled with an intensive focus on effectiveness and how to overcome the problem (Highsmith and Cockburn, 2004). This results in a new combination of values and principles that define an Agile world view. These differences of Agile methodologies include:

- People oriented – Agile considers people – customers, developers, stakeholders, and end users as the most important factor of software methodology (Highsmith and Cockburn, 2005)
- Adaptive – The participants in Agile process welcome change at all stages of the project. Change to the requirements always gives the team members an opportunity to learn more of what it will take to satisfy the market (Fowler, 2004).
- Conformance to actual – Agile methods value conformance to actual results as against detailed plan. It was observed that each iteration or development cycle adds business value to the ongoing project, which is always propagated by the client (Highsmith, 2002).
- Balancing flexibility and planning. – Plans are good, but the truth is that software projects cannot be accurately predicted far into the future, because there are so many variables to take into account. Agile believes that a better strategy to use is to plan for a week or two, beyond that, the plan should be loose (Highsmith, 2000).
- Empirical process – In Engineering, processes are either defined or empirical. Agile processes choose to implement empirical processes, because of the expected changes. Williams and Cockburn (2003) opined that it is highly unlikely that any set of predefined steps will lead to a desirable, predictable outcome because requirements change,technology changes, people are added and taken off the teams.
- Simplicity – Agile methods always take the simplest path that is consistent with their goals. The reason for simplicity is, so that it will be easy to change the design when needed. Documents that predict the future will become outdated one day.
- Collaboration – Agile methods involve customer feedback on a regular and frequent basis. The customer works closely with the developers, providing feedback on their efforts.
- Small self-organizing teams – An Agile team is a self-organizing team. Responsibilities are communicated to the team as a whole, and the team members determine the best way to fulfil them. Agile teams are small (5-9 members), but occasionally, the teams can have up to 120 to 250 members.

Cultural challenges are common to traditional software development methods, but introducing Agile methods will change the command and control method used in companies. Developers would need more autonomy in decision making and the power to implement Agile principles (Fowler, 2004).

The issue of communication challenges will be resolved when appropriate alternative media such as wiki, skype, messaging, IRC (Internet Relay Chat), tele and video conferencing are used. The benefits of short iteration are numerous:

- It promotes transparency of work progress to all partners
- All stakeholders: developers, project managers, customers, can frequently get a good picture of how the project is progressing.
- Distributed developers can get instant feedback on their work through the aforementioned media.
- Seeing high quality work early and frequently builds trust, confidence, and respect between partners. This makes further collaboration easier.
- To the customer, Agile methods bring flexibility and tolerate changes to meet requirements.
- It encourages collaboration and cooperation during the early phases of the project rather than documentation.
- Frequent iteration and testing also ensures that all parties concerned understood the requirements, thus dissolving the cultural barriers.
- Learning from mistakes is fast and early, thus preventing problems from accumulating.
- Frequent and open communication between participants builds trust and breaks cultural barriers.
- Proxy customers are represented by the user company's system Architect who represents the company technically and also proffers business requirements.

Some of the many challenges of TSDMs are related to increased distance between people (Agerfalk, Fitzgerald, Holmstrom, Lings, Lundell, et al, 2005):

- Geographical distance
- Temporal distance
- Socio-cultural distance

To reduce these distances, the current technique of "near-shoring" is being practiced, where low-cost and not so far locations are being explored, such as US-Brazil and EU-Eastern Europe (Camel and Tjia, 2005).
Pair programming, a technique of having two programmers working together on all production codes using one computer, seemed to be impossible to practice in TSDM. However, through time–shifting, patterns and developers create an overlap and reduce the temporal distance (Holmstrom, Fitzeralde, Agerfalk, Conchuir, 2006). This way an Engineer in the US can work six hours a day paired with another Engineer in, say, Belgium. It is also believed that pair programming helped to increase knowledge sharing (Holmstrom et al, 2006) and reduce socio-cultural distance. Hence, not only did pair programming deliver the expected benefits, but the benefits turned out to allay the TSDM related fears of distributed development.
Distributed extreme programming (DXP) suggests that eight of the XP practices, such as small releases, metaphor, simple design, testing, refactoring, collective ownership, 40-hour week, and coding standards are independent of team locality and can be applied in TSDMs (Kirscher, Jain, Corsaro, Levine, 2001).

Sriram and Matthew (2012) presented a review of literature on applying ASDM methodologies in TSDMs, and how the methodologies fit into TSDMs. Three main ideas were identified:

- Performance of formal software development
- Governance related issues
- Software engineering process related issues

The authors concluded that various types of Agile methods were applied and tailored appropriately to produce optimum performance in the context of TSDM. However, empirical studies addressing TSDM-ASDM fit were not found.

Sletholt, Hannay, Pfahl, Benestad, Langtangen (2011) conducted literature review to investigate the effects of using Agile practices in scientific software projects, especially scrum and XP and compared with formal software development methods. The authors found that the projects that adopted Agile techniques had improved testing processes compared to the TSDMs.

TSDM brought up many challenges relating to distributed development, such as

- Interdependencies among work items that are distributed
- Difficulties of coordination
- Difficulties of dividing the work into modules that could be assigned to different locations.
- Conflicting implicit assumptions
- Communication challenges

Battin, Crocker, Kreidler, Subramanian (2001) suggested incremental integration plan, which would be based on clusters and shared incremental milestones, to avoid the “big-bang” integration and last minute problems. The authors explained that dividing the work into modules for distribution and later integrate the pieces, would bring unforeseen problems at the end. Battin and colleagues’ strategy was successfully used at Motorola and Alcatel companies, using one dedicated team to each increment. They also based their progress tracking on successfully integrated and tested customer requirements (Ebert and De Neve, 2001).

The authors reported that a stable build proved to be one of the key success factors, and that globally applied continuous builds improved project life cycle.

Karlsson, Andersson, Leion (2000) confirm the use of frequent builds, in fact, daily builds and feature-based development as success in distributed development projects. Incremental integration and frequent deliveries are core practices in ASDMs (Larman and Basili, 2003).

Offshore software development was successfully practiced using Agile methodologies, especially using iterative development, and frequent deliveries (Fowler, 2004; Simons, 2002). These techniques, according to the authors, increased project visibility and provided an avenue for the customer and project managers to follow project progress (Simons, 2002). Whereas 30 days iteration length are suggested using scrum, Fowler (2004) suggested the use of two-weekly iteration in XP, to reduce communication overhead in distributed development. Paasivaara and Lassenius (2004) opined that Agile principles of frequent deliveries and continuous iteration seem to suit traditional software development methods. Both Fowler and Simons opined that their successes with distributed development were due to high responsiveness to change and fast delivery of business values. These benefits, they concluded, outweighed the challenges of distributed development.

Nisar and Hameed (2004) reported that they used XP in offshore distribuyted software (ODSD) to achieve client satisfaction. While Xiaohu, Bin, Zhijun, Maddineni (2004) used XP in TSDM to reduce communication delay and improve communication quality between the customers and offshore development team. The authors concluded that the XP principles they followed proved very successful. Karlsson et al. (2000) and Farmer (2004) reported that they found Agile principles useful, but hard to implement in TSDMs. The authors found that Agile principles, having used XP in offshore distribution successfully, is possible with TSDM. Rather than migrate to Agile methods, it is better to hybridize the two methods, which would improve software development process.

Schwaber (2004) reported how scrum method could be scaled to large projects involving multiple scrum teams. These scaling mechanisms enabled the usage of scrum also in geographically distributed projects. Schwaber further suggested that in these kinds of projects, high bandwidth technologies for source code sharing, synchronized builds, alternative communication methods such as instant messaging, mailing lists, wiki, Internet Relay Chat (IRC), skype and both tele and video conferencing should be used.

3.0 DISCUSSIONS

It is the ability to respond to change that often determines the success or failure of a software project (Williams and Cockburn, 2003). This forms the main difference between traditional software development methods (heavyweight) and Agile software development (lightweight) methodologies. Traditional software development methods freeze product functionality and disallow change. While Agile processes respond to change at any stage of the project.

Facilitating change is more effective than attempting to prevent it. It is more important than planning for disaster (Boehm and Philip, 1988). Standish Group International (SGI) carried out a research on software projects and came up with, among others, three major reasons for a project to be successful, such as:

- User involvement
- Executive management support
- A clear statement of requirements.

The idea of planning for everything upfront can work for small projects, but for large and complex environments, this technique would fall apart (Fowler, 2004). Traditional software development methods (Heavyweight) handled complex projects differently. They planned ahead or upfront, and were bound to fail. The solution lies in simplicity. Agile software development methodologies promote simplicity, because it is easier to effect changes. SGI further holds an opinion that 45% of features found in an application are never used. This is another reason why the design and code should be as simple as possible (SGI, 1994).

Eisenhardt and Sull (2001) suggested that instead of following complex processes, using simple rules to communicate strategies is the best way to empower people to seize fleeting opportunities in rapidly changing markets. Documentation, which is very much valued in traditional software development methodologies, is unnecessary, according to Poppendieck (2005). She explained that documents, diagrams, and models produced as part of software development must be minimized, because once a working system is delivered, the user may care little about these deliverables. Agile

methods follow the same rules for their processes. Another reason is that excess documentation creates a waste of time in producing and reviewing the document. Rather than have a hundred page detailed specification, it is better to write a ten page set of rules and guidelines. This is what Agile methodologies maintain,
documentation should be kept to the barest minimum.
Rather than taking months or years to show the customer the final product,
use an iterative development where small but complete portions of a system are designed and delivered throughout the development cycle. This technique allows the customer to have a better idea of how the software works. This is what Agile methods adopted.
Software development practices, which keep requirements flexible and as close to the system delivery as possible, provide competitive advantages in a changing environment. In a similar way, Agile methods are designed to respond to change, not to predict it, and also have the ability to make decisions as late as possible. Agile methodologies empower team members. They provide both tools and authority to team members, other than managers, to take decisions. This is one of the problems with traditional software development methods, where all the decisions are made for developers.
Instead, Agile methodologies give developers guidance as well as freedom to take detailed design and programing decisions. Poppendieck (2005) opined that it is better to tell developers what needs to be done, but not how to do it.
In respect to iterative development, traditional software development methodologies keep customer feedback and testing until the last stage of their project life cycle. Agile believes otherwise, these should be embedded into the system as a daily exercise. SGI (1994) discovered that the delivery of software components early and often, within short time frames, increases the success rate.
However, Constantine (2001) opined that it is not every problem that can be sliced and diced into the right pieces for speedy incremental refinement. Therefore, Agile processes do not work in all cases. High smith and Cockburn (2005) opined that Agile methods are more difficult for larger teams, because as the team size grows, coordinating interfaces become a dominant issue. Also, with developers numbering 20 and above, the face-to-face communication breaks down and becomes more difficult and complex (Constantine, 2001). In contrast, plan-driven methods scale better with large projects.
Boehm (1988) contested that over focus on early and continuous delivery of valuable phases of a product can lead to a major rework, if the architecture doesn't scale up. He concluded that a plan driven process is most needed for high assurance software. In the same vein, Ambler (2005) opined thatit would be suspicious applying Agile modeling in life critical systems. This is because Agile methodologies do not support the TSDM goals of predictability, repeatability, and optimization, which are characteristics of reliable safety critical software development. Fowler (2004) opined that Agile methods are only good for business software.
Whereas Agile processes play down on documentation as a waste of time and resources, Boehm (1988) is of the opinion that it is only through documentation that external experts can diagnose problems. Proposing no documentation could lead to an increase in the risk factor when considering maintenance and usage. Other problems with using Agile methods include over emphasis towards customer collaboration. The truth is that, getting busy people involved in the development process is irritating and an odd job indeed. More so, it takes a dedicated customer to build quality software, using Agile methods.
Another problem with Agile methods is project cost. Agile projects have no fix prices, no fix schedule, and projects are open-ended and evolve as requirements change. Therefore, it becomes harder for managers and customers alike to accept this technique as they would rather know the total cost of the project, and overall project schedule beforehand.
In the past few years, many companies have turned to offshore development for faster, better, and cheaper development teams. Other benefits include increased productivity, competitive advantage, and internal customer satisfaction (Moore and Barnett, 2004). However, offshore development comes with no physical proximity, and that plan-driven approach is usually favored where business analysis, detailed requirement and design are done at the front office (on-shore) and sent to the back office (off-shore) for construction. These arrangements come with challenges for Agile methodologies.

Time zone differences and separation by thousands of miles reduce the communication between on-shore and off-shore teams. Anyway, Fowler (2005) proposed the following actions to be used for a successful implementation of Agile methodologies in offshore software development:

- Use distributed continuous integration
- Have each site send ambassadors to the other site.
- Use contact visits to build trust.
- Don't underestimate the cultural change. In Asian countries, command and control model is used. This discourages some countries adopting Agile methods offshore.
- Use test scripts to help understand the requirements – Acceptance tests help to communicate and clarify the requirements between offshore and onshore team members.
- Use regular builds to get feedback on functionality – according to Fowler (2005), the quicker the customer can look at a partial function, the quicker they can spot any miscommunications.

Having skilled and experienced people in a team is a key factor for Agile methodologies. With domain experts in the team, developers have rapid feedback on the implications to the users of their design choices.
With regards to applying ASDMs in large-scale and mission critical projects, it is not clear if Agile methods are used on large-scale projects, such that they can provide end users with the desired quality in timely manner (Marrington, Hogan, and Thomas, 2005). However, some researchers reported otherwise. They contend that large-scale and complex projects have benefited from suitably tailored Agile development methods (Bowers, May, Melander, Baarman, Ayood, 2002; Cao, Mohan, Xu, Ramesh, 2004; Lindvall,

Muthing, Dagnino, Wallin, Stupperich, et al., 2004). Bowers et al. (2002) examined if XP could be used in large-scale and mission critical projects and saw that it could be done, with some modifications. Lippert, Becker-Pachau, Breithing, Koch, Kornstadt, et al.(2003) had a similar experience with XP and confirmed the possibility of applying Agile software development methods on large-scale and mission-critical projects.

4.0 THE AMBIDEXTROUS FRAMEWORK

According to Nerur, Mahapatra, Mangalaraji (2005), ASDM and TSDM have conflicting organizational cultures, management styles, organizational forms, and reward systems. Ambidexterity is a technique that will allow the marriage of the two methods drawing from their strengths and reducing their weaknesses. Ambidexterity is an effective and viable solution to the seemingly extreme stands of "stability" for TSDM, and "agility" for ASDM (Katila and Ahuja, 2002; He and Wong, 2004). Figure 1 shows the ambidextrous framework.

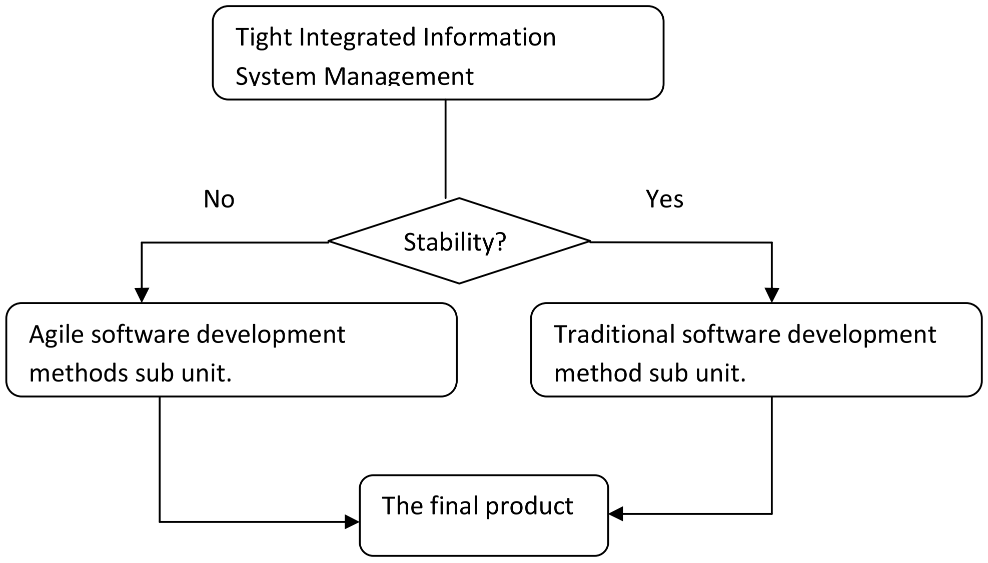
Tight Integrated Information System Management
No
Yes
Stability?
Agile software development methods sub unit.
Traditional software development method sub unit.
The final product

Figure1: Ambidextrous Framework (Source: Nerur et al., 2005),

The ambidextrous framework has sub units that are highly coupled within the subunits and loosely coupled across sub units, but are tightly integrated at the senior executive level (O'Reilly and Tushman, 2004). The tasks are highly consistent within each sub unit and highly different from other sub units (Benner and Tushman, 2003).

5.0 CONCLUSION

This research work surveyed the possible means of hybridizing STDM and ASDM and came up with an ambidextrous framework, which could be used to bridge the two. Both methods have their strengths and weaknesses. Traditional software development methods have the following advantages:

- Stability
- Planning upfront
- Detailed documentation
- Management control of decision making in the project
- Fixed cost estimates
- Fixed schedules
- Codified process
- Encourage reuse of pre-process modules
- Requirements known in advance, and fixed.

However, STDM has drawbacks, which include

- Planning upfront for large projects leading to failures
- Delayed customer feedback and modules testing to the last phase of the project.
- Not flexible to requirements changes
- Not all documented requirements are implemented
- Prone to litigations due to breach of contract terms, by not meeting schedule dates, among others

Agile software development methods (ASDM) prove to be viable alternatives to STDM with the following advantages:

- Simplicity
- Hitting the market on target time.
- Customer satisfaction guaranteed
- People oriented
- Iterative – delivery of small and complete pieces of software and on time.
- Face-to-face communication
- Close cooperation between developers and the clients
- Freedom to developers
- Regular adaptation to changes in requirements.

ASDM also have her fair share of the software development problems (short comings), which include:

- ASDMs are difficult to implement

- Limited documentation
- The customer must not only be knowledgeable, but interested for ASDM to work.
- Only skilled developers can implement ASDM.
- Unrealistic expectations
- Constant need of attention
- Testing is not cheap
- Not much empirical evidence to support ASDM's claims
- No fix project cost and schedule

These differences notwithstanding, ambidextrous framework proposed would cement their symbiotic relationships for better software development.

REFERENCES

Agerfalk, P. J., Fitzgerald, B., Holmstrom, H., Lings, B., Lundell, B., Conchuir, E. 2005. A framework for considering opportunities and threats in Distributed software Development. In proceedings of the International workshop on Distributed software Development DISD 2005, Paris, Austrian Computer Society, pp. 47-61.

Ambler, S. 2005, "When Does (not) Agile Modeling Make Sense?" *http://www.Agilemodeling.com/essays/whendoesamwork.htm#wontwork*

Awad, M. A. 2005. A comparison between Agile and traditional software development methodologies. Unpublished doctoral dissertation. The University of Western Australia. Australia.

Battin, R. D., Crocker, R., Kreidler, J., Subramanian, K. 2001. "Leveraging resources in global software development", IEEE Software,18(2): 70 – 77.

Bech, K., Beedle, M., Bennekum, A., Cockburn, A.S., Cunningham, W., Fowler, M, et al. 2001. Manifesto for Agile software development. *http://www.Agilemanifesto.org/*

Benner, M.J. and Tushman, M. L. 2003 Exploitation, Exploration, and process management: The productivity dilemma revisited. Ademy of Management Review 28(2): 238-256

Boehm, B. 1988 "A Spiral Model for Software Development and Enhancement," Computer, vol. 21, no. 5, pp. 61-72.

Boehm, B. 2002. Get ready for Agile methods with care. Computer.35 (1): 64- 69.

Boehm, B and Philip, P. 1998."Understanding and controlling software cost", IEEE Transactions on software Engineering,14 (10).

Boehm, B. and Turner, R. 2003.Using risk to balance Agile and plan-driven methods. Computer, 36(6): 57-66.

Boehm, B. and Turner, R. 2005, "Management challenges to implement Agile processes in traditional organizations", IEEE Software, 22(25):30-39.

Bowers, J., May, J., Melander, E., Baarman, M., Ayoob, A. 2002. Tailoring XP for large systems mission-critical software development. Proceedings of the second XP Universe and First Agile Universe Conference on Extreme Programming and Agile Methods, pp. 100-111.

Brooks, F.P. 1975. The Mythical Man-Month: Essay on Software Engineering. Addison Wesley.

Cao, L., Mohan, K., Xu, P., Ramesh, B. 2004. How extreme does extreme programming have to be? Adapting XP practices to large-scale projects. Proceedings of the 37th Hawaii International Conference on system sciences, pp. 1-10.

Carmel, E. and Tjia, P. 2005, Offshoring Information Technology: Sourcing and outsourcing to a Global Workforce: Cambridge University Press, Cambridge, NY.

Chow, T. and Cao, D-B. 2007. A survey study of critical success factors in Agile software projects. Journal of systems and software, 81(2008): 961-971.

Cockburn, A. 2006. "Agile software development: A cooperative Game", 2nd Edition, Addison-Wesley Professional, Boston.

Cohn, M. and Ford, D. 2003. Introducing an Agile process to an organization. IEEE Computer, 36(6):74-78.

Constantine, L. 2001, "Methodological Agility", IEEE Software Development, pp. 67-69

Ebert, C. and De Neve, P. 2001, Surviving Global Software Development. IEEE Software, pp. 62-69.

Eisenhardt, H and Sull, D. 2001 "Strategy as Simple Rules", Harvard Business Review, 79: 107-116.

Farmer, M. 2004. Decision space Infrastructure: Agile development in a large, distributed Team. Proceedings of the Agile Development Conference.

Fowler, M. 2004. Using Agile Software Process with Offshore Development. *http://martinfowler.com/articles/agileOffshore.ht*ml 7.1.

Fowler, M. 2005. The new methodology, Available online at :*http://matinfowler.com/articles/newmethodology.html*

Fruhling, A. and De Vreede, G. J 2006. Field experiences with extreme programming: Developing an emergency response system. Journal of Management Information Systems, 22(4): 39-64.

Goldman, A., Kon, F., Silva, P.J.S., Yoder, J. W. 2004, Being extreme in the classroom: Experiences leaching XP. Journal of the Brazilian Computer Society, 10, 5-21.

He, Z. and Wong, P. 2004. Exploration vs exploitation: An empirical test of the ambidexterity hypothesis. Organization Science, 15(4): 481-494.

Hickey, A.M. and Davis, A.M. 2004. A unified model of requirements elicitation, Journal of Management Information Systems, 20(4): 65-84.

Highsmith, J. 2002, Agile Software Development Ecosystem. Addison Wesley.

Highsmith, J. and Cockburn, A. 2001.Agile software development: The Business innovation. IEEE Computer, 34(9):120-122.

Highsmith, J. and Cockburn, A. 2004, "Agile Software Development: The People Factor", IEEE Computer, Available online at: *http://www.jimhighsmith.com/articles/IEEEArticle_1Final.pdf*

Highsmith, J. and Cockburn, A. 2005, "Agile Software Development: The Business of Innovation", IEEE Computer, Available online at: *http://www.jimhighsmith.com/articles/IEEEArticle_2Final.pdf*

Highsmith, J. A. 2000, Adaptive Software Development: A collaborative Approach to Managing Complex Systems. New York: Dorset House Publishing.

Holstrom, H., Fitzgerald, B., Agerfalk, P. J., Conchuir, E. O. 2006 Agile Practices Reduce Distance in Global Software Development. Information System Management, 23(3): 7-18, Available online at: *http:/www.therationaledge.com/content/jan-01/f_rup_pk.html*

Karlsson, E., Andersson, L., Leion, P. 2000, Daily Build and Feature Development in large Distributed Projects.Proceedings of the International Conference on Software Engineering, Limeric, Ireland, pp. 649-658

Katila, R. and Ahuja, G. 2002. Something old, something new: A longitudinal study of search behavior and new product introduction. Academy of Management Journal 45(6): 1183-1194.

Kirscher, M., Jain, P., Corsaro, A., Levine, D. 2001, Distributed Extreme Programming. In: Proc. International Conference on eXtreme Programming and Flexible Processes in Software Engineering.

Kruchten, P. 2004. The rational Unified Process: An Introduction (3rd ed.). Reading, MA: Addison-Wesley, Longman.

Larman, C. 2004. Agile and Iterative Development: A Manager's Guide. Addison Wesley

Larman, C. and Basili, V. 2003, Iterative and Incremental Development: A Brief History. IEEE Computer, pp. 47-56.

Leffingwell, D. 2007. Scaling software agility: Best practices for large enterprises. Upper Saddle River. NJ: Addison-Wesley.

Lindvall, M., Muthing, D., Dagnino, A., Wallin, C., Stupperich, M., Kiefer, E, et al. 2004. Agile software development in large organizations. Computer, 37(12): 26-34.

Lippert, M., Becker-Pechau, P., Breithing, H., Koch, J., Kornstadt, A., Roock, S., et al. 2003. Developing complex projects using XP with extensions. Computer, 36(6): 67-73.

Marrington, A., Hogan, J.M., Thomas, R. 2005. Quality assurance in a student- based Agile software engineering process. Proceedings of the 2005 Australian Software engineering conference,pp. 324-331.

Mockus, A. and Herbsleb, J. 2001, Challenges of Global Software Development. Proceedings of the seventh International Software Metrics Symposium,(METRICS 2001, IEEE) pp.182-184.

Moore, S. and Barnett, L. 2004, "Offshore outsourcing and Agile Development", Forester Best Practices.

Nerur, S., Mahapatra, R., Mangalaraji, G. 2005. Challenges of migrating to Agile methodologies communications of the ACM, 48(5): 73-78.

Nisar, M and Hameed, T. 2004, Agile Methods Handling Offshore Software Development Issues. Proceedings of INMIC 2004, 8th International Multi-topic Conference, pp.417- 422.

O'Reilly, C.A III and Tushman, M.L. 2004.The ambidextrous organization. Harvard Business Review 83(4):74-81.

Paasivaara, M and Lassenius, C. 2004, Synchronizing Global Software Development by using Incremental Development and frequent Deliveries. Proceedings of the ICSE International workshop on Global Software Development, Edinburg.

Poppendieck, M. 2005, "Lean Programming". *http://www.poppendieck.com/lean.htm*

Royce, W.W. 1970. Managing the development of large software systems, In: Proceedings of WESCON, pp. 1-9.

Satzinger, J.W., Jackson, R.B, Burd, S.D. 2005. Object oriented Analysis and design with Unified process. Boston: Thomson Course-Technology.

Schach, S.R. 2004. An Introduction to Object-Oriented systems analysis and design with UML and the Unified process, Boston: McGraw-Hill.

Schwaber, K. and Beedle, M. 2002.Agile software development with scrum, Upper Saddle River, NJ: Prentice Hall.

Simons, M. 2002."Internationally Agile", InformIT.

Sletholt, M. T., Hannay, J., Pfahl, D., Benestad, H. C., Langtangen, H. P. 2011, A literature Review of Agile Practices and their Effects in Scientific Software Development. Proceedings of the 4th International Workshop on Software Engineering for Computational Science and Engineering, Wuhan, 9-11 December, 2011, pp. 1-9.

Sriram, R. and Matthew, S. K. 2012, Global Software Development using AgileMethodologies: A Review of Literature. 2012 IEEE International Conference on Management of Innovation and Technology, Bali, 11-13 June, 2012, pp. 389-393.

Standish Group International 1994. "The CHAOS Report, *http://www.standishgroup.com/sample research/chaos_1994_1.php*

Williams, L. and Cockburn, A. 2003, "Agile Software Development: It's about Feedback and Change, IEEE Computer, pp. 39-43.

Willson, C. D. 2009. A brief Introduction to SCRUM: AN Agile Methodology. Martincor Inc. Information Technology Management Consulting.

Xiaohu, Y., Bin, X., Zhijun, H., Maddineni, S. 2004 Extreme Programming in Global Software Development. Proceedings of the Canadian Conference on Electrical and Computer Engineering,4:1845-1848.