A SURVEY OF AGILE METHODOLOGIES

E. Mnkandla* and B. Dwolatzky**

*School of Information Technology, Monash University, South Africa, Private Bag X60, Roodepoort, Johannesburg, 1725, South Africa, Ernest.Mnkandla@infotech.monash.edu  
**School of Electrical and Information Engineering, University of the Witwatersrand, P.O. Box 542, Wits, 2050, Johannesburg, South Africa, b.dwolatzky@ee.wits.ac.za

Abstract: The failure of software development projects has been a cause for concern for a long time. South Africa’s software development industry is no exception. In fact, in South Africa the majority of the software development organizations are small to medium sized. Some even have one developer developing large systems. As is common with software development organizations, most of the applications are developed without following any development methodology and sometimes without following any project management methodology. This scenario leaves the South African industry prone to failures and catastrophes that have devastated the software industry for many years. This review aims to bring an awareness of an emerging software development discipline called Agile methodologies that claims to deal adequately with issues such as the social side of software development, where the interactions among developers, users, customers, managers and the proposed system actually affect the quality, cost, and time of software development.

Key words: Agile methodologies, traditional methodologies, rigorous methodologies, iterative incremental development.

1 INTRODUCTION

As the South African government continues to fight the former regime’s ills the IT industry has not been left behind. The software development industry has seen an increase in the number of empowerment companies, in which the formally disadvantaged communities are coming forward to take part in this blooming IT niche. It is often said that all you need is a properly trained brain and a PC to produce software. It is therefore inevitable that as this relatively young industry grows it is armed with the correct methodologies and practices of software development. In this survey are outlined the practices that constitute Agile methodologies. These practices have a lot of relevance to the South African industry and any other developing countries for that matter, because Agile methodologies seem to address the problems that characterize this industry.

The principles of software engineering are very important for the development of quality, reliable, and maintainable software, however, they have been found to be too rigorous and lacking innovation leading to longer time spans for the development of software applications. This is undesirable for the very short development time frames that characterize Information Age Applications (i.e. Internet and Web applications).

The fray on software development methodologies is an old one with suggestions that it may have started as early as 1968 [18] with Dijkstra’s argument that considered Go To statements in computer programs as harmful. This was followed by several publications by Dijkstra and his colleagues on structured programming. According to Highsmith [13] over the last 25 years the software industry has been involved in debates on issues such as; processes that are data-driven versus those that are process-driven, information engineering versus structured development and relational database design versus object database design.

Today’s information and communication technology is largely dependent on the Internet and the World Wide Web; hence the development of most computer software today has to be Internet-friendly. The Internet has brought a lot of advantages in most fields that affect humanity, but it has also brought about drastic changes in some areas. One of these areas is software development. The traditional methodologies of software development were plan-driven (big design upfront) and rigorous [5]. On the other hand, modern light-weight software development methodologies have been given the name Agile Methodologies [20] and are characterized by little initial planning and short iterations. The present debate is on Agile Software Development versus rigorous software development.

Published literature on Agile software development methodologies is mainly inundated with strong marketing initiatives and provides less information about detailed representation of these methodologies. This works to their disadvantage because people either have incorrect information or insufficient information about the Agile methodologies. This explains in part the reason for so much scepticism (especially from those who have not
practically applied these methodologies) about this new approach to developing software.

Changing from the traditional methodologies of software development that are process and document driven must not be done in a radical way and the entire organization must be in agreement. Considering the two main camps in the battle for methodology supremacy, i.e. the traditional methodologies camp and Agile methodologies camp, it makes sense to agree with Rosenberg [24] that a methodology in between RUP and XP would be more relevant. While constructive criticism in the academia and research in particular is not only inevitable but very welcome, as Glass [9] puts it polarizing the debate does not help, instead combining ‘forces’ i.e. the strengths from both sides would produce a more appropriate environment. It would not surprise most players in the software development industry if in fact a new engineering discipline that values the softer skills of people and their innovativeness could be in the making. Call it Agile Software Engineering (ASE)? May be Agile Engineering without the software part would confuse many as the term ‘Agile’ has become a buzzword even in none software industries.

The rest of the paper starts with the most common definitions of Agile methodologies, then in section two starting with Extreme Programming a following Agile methodology is reviewed; ICONIX, Serum, Feature-Driven Development, Dynamic Software Development, Adaptive Software Development, Crystal, Lean Development and Agile Modeling is the last one in section ten. Section eleven looks at some challenges that still face Agile methodologies and ends with the further research possibilities. Section twelve concludes the review by stating our findings and recommendations.

1.1 Definition

When more than twenty Agile methodologies experts from around the globe were brought together through the first eWorkshop on Agile methodologies they started by defining what Agile methodologies are. The results of the eWorkshop were summarized by Lindvall et al [15] in particular the working definition of Agile methodologies is given as; iterative, i.e. delivers a full system at the very beginning and then changes the functionality of each subsystem with each new release, incremental, i.e. the system as specified in the requirements is partitioned into small subsystems by functionality and new functionality is added with each new release, self-organizing, i.e. the team has the autonomy to organize itself to best complete the work items, emergent, i.e. technology and requirements are “allowed” to emerge through the product development cycle.

All Agile methodologies follow the four values and twelve principles as outlined in the Agile Manifesto. They are characterized by their value of people and their interactions as opposed to processes and tools, working code as opposed to documentation, involvement of the customer in the development process as opposed to contract negotiation and accepting change as opposed to following a plan. They are made up of a variety of methodologies. The review structure for each methodology starts from a brief overview of the methodology, followed by a short description of the methodology’s process (life cycle), then comes a listing of the main project management features, for the team size we have not included the scaling values because we realize that as the Agile methodologies are scaled up to larger teams they begin to include more aspects of the rigorous group and hence become less agile. The last feature is the scope of use of the methodology.

2 EXTREME PROGRAMMING (XP)

Extreme Programming is a lightweight methodology for small-to-medium-sized teams developing software based on vague or rapidly changing requirements [4]. Software development using XP starts from the creation of stories by the customer to describe the functionality of the software. These stories are small units of functionality taking about a week or two to code and test. Programmers provide estimates for the stories, the customer decides, based on value and cost, which stories to do first. Development is done iteratively and incrementally. Each two weeks, the programming team delivers working stories to the customer. Then the customer chooses another two weeks worth of work. The system grows in functionality, piece by piece, steered by the customer. This means that progress is measured and tracked based on the observable behavior of the system, rather than judging design artifacts. XP relies on evolutionary design techniques to keep the software design good while adding new functionality. XP is a collection of twelve best practices [17].

2.1 XP Life Cycle

Brewer [6] outlines the twelve core practices of XP as: The Planning Game: Business and development cooperate to produce the maximum business value as rapidly as possible. The planning game happens at various scales, but the basic rules are always the same. The user comes up with a list of desired features for the system. Each feature is written out as a user story, which gives the feature a name, and describes in broad strokes what is required. User stories are typically written on 4" x 6" cards. Developers estimate how much effort each story will take, and how much effort the team can produce in a given time interval (the iteration). Users then decide which stories to implement in what order, as well as when and how often to produce production releases of the system.
Small Releases: Start with the smallest and simplest useful feature set. Release early and often, adding a few features each time.

System Metaphor: Each project has an organizing metaphor, which provides an easy to remember naming convention, using a simple shared story of how the system works.

Simple Design: Always use the simplest possible design that gets the job done. The requirements will change tomorrow, so only do what's needed to meet today's requirements.

Continuous Testing: Before programmers add a feature, they write a test. When the suite runs, the job is done. Tests in XP come in two types: unit tests and acceptance tests.

Refactoring: is done by Refactoring out any duplicate code generated in a coding session.

Pair Programming: All production code is written by two programmers sitting at one machine. This provides reviewing as code is written.

Collective Code Ownership: No single person "owns" a module. Any developer is expected to be able to work on any part of the system code at any time.

Continuous Integration: All changes are integrated into the system code at least daily. All tests must pass. The tests have to run one hundred percent both before and after integration.

Forty-Hour Work Week: Programmers go home on time.

On-site Customer: Development team has continuous access to a real live customer, that is, someone who will actually be using the system. For commercial software with lots of customers, a customer proxy (usually the product manager) is used instead.

Coding Standards: Everyone codes to the same standards. Ideally, it should not be possible to tell by looking at the code that it has been changed.

2.2 Strengths and Limitations of XP

Strengths: values communication, simplicity and feedback, intensively social activity, it is iterative, based on the software industry’s best practices, and puts emphasis on design.

Limitations: no design documentation, which can be a problem at the maintenance stage, it differs from other Agile methodologies by being much more prescriptive, the use of pair programming is quite controversial due to an added overhead on resources, though it can be argued that it leads to fewer defects and reduced cycle time, and XP does not address the issue of deployment [22]. XP is also weak in the management process, risk, and measurement practices [27].

2.3 Project Features

Team size: Because the development team needs to be co-located, team size is limited to the number of people that can fit in a single room, generally agreed to be from two to ten, though there has been evidence of use of up to five hundred people with scaling [15].

Iteration length: XP has the shortest recommended iteration length of the Agile methodologies under consideration, two weeks and not exceeding three weeks according to Boehm [27].

Support for distributed teams: Because of XP’s focus on community and collocation, distributed teams are not supported. It has however been used with some introduction of plan-driven methods especially in architectural planning and design [27].

System criticality: XP is not necessarily geared for one system or another. However, most agree that there is nothing in XP itself that should limit its applicability.

Application: applicable to projects that require at least two programmers up to maximum of ten and driven by the need for quick delivery. It also been used successfully for large complex projects where plan-driven methods had been ineffective [27].

3 ICONIX SOFTWARE DEVELOPMENT PROCESS

ICONIX is a medium-sized software development methodology whose analysis and design strength is based on UML. It is said to lie somewhere between RUP and XP [24]. ICONIX offers a stream lined approach to software development that includes a minimal set of diagrams and techniques that a project team can use to get from use cases to code quickly and efficiently.

The ICONIX process starts with domain modeling, which involves identifying the objects in the real world that will serve as the vocabulary for the use cases. This gets much of the team meaningfully involved in the project right away [24]. The use cases in ICONIX are made up of small pieces of straightforward text that captures functional requirements in a manner that’s easy for everyone to understand.

3.1 ICONIX Life Cycle

ICONIX has three significant features. First, the approach offers streamlined usage of the Unified Modeling Language (UML). Second, the approach offers a high degree of traceability. At every step along the way, you refer back to the requirements in some way. There is never a point at which the process allows you to stray too far from the user’s needs. This also means that objects can be tracked from step to step into a set of design artifacts. Third, the approach is iterative and incremental. There are multiple iterations that occur between developing the domain model and identifying and analyzing the use cases. The static model gets refined incrementally during the successive iterations through the dynamic model (composed of use cases, robustness analysis, and sequence diagrams). The approach does not require formal milestones and a lot of bookkeeping;
rather, the refinement efforts result in natural milestones as the project team gains knowledge and experience [25]. The project monitoring and control system emerges on its own as work continues.

3.2 Strengths and Limitations of ICONIX

ICONIX has the following strengths:

- Use of UML gives it a highly visual OOA and design approach
- It is a medium-sized process
- A clear link between what the system being modeled is supposed to (results of analysis) and how the system being modeled is going to function (the results of detailed design).

Limitations of ICONIX: There may be need to train developers in OO technology such as UML before they can use ICONIX. There is little existing literature or published research work on ICONIX.

3.3 Project Features

Team size: Since ICONIX is a modeling methodology the team size is not specified, but it is said to be customizable and scalable [24].

Iteration length: The length is not clearly specified iterations are marked by a set of use cases that should be converted to code.

Support for distributed teams: Face to face communication is not a major factor in ICONIX hence it would work well for distributed systems. In fact in Rosenberg and Scott [24] the dramatized story used to illustrate ICONIX is for a distributed team.

System criticality: ICONIX is very heavy on system design and analysis and should be able to deal with critical systems.

Application: can be applied in any software development projects where object orientation is the fundamental technology.

4 SCRUM METHODOLOGY

Scrum has been around for a while in object-oriented circles. Scrum, along with XP, is one of the more widely used Agile methodologies. Scrum’s focus is on the fact that defined and repeatable processes only work for tackling defined and repeatable problems with defined and repeatable people in defined and repeatable environments [8], which is obviously not possible. To solve the problem of defined and repeatable processes Scrum divides a project into iterations (which are called sprints) of thirty days. Before a sprint begins the functionality required is defined for that sprint and the team is left to deliver it. The point is to stabilize the requirements during the sprint. Scrum emphasizes project management concepts. The term Scrum is borrowed from Rugby: “A Scrum occurs when players from each team clump closely together...in an attempt to advance down the playing field” [10].

4.1 Scrum Life Cycle

The Scrum process is made up of three main phases; pre-sprint planning, sprint and post-sprint planning. These phases can be defined as follows:

Pre-sprint planning: All work to be done on the system is kept in what is called the “release backlog.” During the pre-sprint planning, features and functionality are selected from the release backlog and placed into the “sprint backlog,” or a prioritized collection of tasks to be completed during the next sprint [10].

Sprint: Upon completion of the pre-sprint planning, teams are handed their sprint backlog and “told to sprint to achieve their objectives”. At this point, tasks in the sprint backlog are frozen and remain unchangeable for the duration of the sprint. Team members choose the tasks they want to work on and begin development. Short daily meetings are critical to the success of Scrum. Scrum meetings are held every morning to enhance communication and inform customers, developers, and managers on the status of the project, identify any problems encountered, and keep the entire team focused on a common goal [10].

Post-sprint meeting: After every sprint, a post-sprint meeting is held to analyze project progress and demonstrate the current system.

Schwaber [26] summarizes the key principles of Scrum:

- Small working teams that maximize communication, minimize overhead, and maximize sharing of implicit informal knowledge.
- Adaptability to technical or marketplace (user/customer) changes to ensure the best possible product is produced.
- Frequent ‘builds’, or construction of executables, that can be inspected, adjusted, tested, documented, and built on.
- Partitioning of work and team assignments into clean, low coupling partitions, or packets.
- Constant testing and documentation of a product as it is built.
- Ability to declare a product ‘done’ whenever required.

4.2 Strengths and Limitations of Scrum

Strengths: Scrum’s advantages are:

- Detailed project management process [27].
- Product becomes a series of manageable chunks,
- Progress is made even if requirements are not stable,
- The daily Scrum meeting provides a very good project communication technique.
- Customers see on-time delivery of increments.

Limitations: No detailed information on integration and acceptance tests [1]. There is little information on
supporting tools. Scrum does not define well the business system, business enterprise, maintenance, design, technical practices, measurement practices, and concept development [27].

4.3 Project Features

Team size: Development personnel are split into teams from seven up to ten people. A complete team should at least include a developer, quality assurance engineer, and a documenter.

Iteration length: Schwaber [26] originally suggested sprint lengths to be between one to six weeks and Highsmith [10] has found that durations are commonly held at 4 weeks.

Support for distributed teams: While Scrum’s prescription does not explicitly mention distributed teams or provide built-in support; a project may consist of multiple teams that could easily be distributed.

System criticality: Scrum does not explicitly address the issue of criticality.

Application: The use of Scrum seems to extend to a wide range of software projects of various sizes. It can also be used in non-software projects.

5 FEATURE DRIVEN DEVELOPMENT (FDD)

Feature Driven Development was developed by Jeff De Luca and long time Object-Orientation guru Peter Coad. Like the other adaptive methodologies, it focuses on short iterations that deliver tangible functionality. According to Palmer and Felsing [19], the FDD approach does not cover the entire software development process, but rather focuses on the design and building phases, but has however been designed to work with the other activities of a software development project. FDD has iterations that are two weeks long.

FDD has two kinds of developers; class owners and chief programmers. The chief programmers are the most experienced developers. They are assigned features to build. However they don’t build them alone. Instead the chief programmer identifies which classes are involved in implementing the feature and gathers their class owners together to form a feature team for developing that feature. The chief programmer acts as the coordinator, lead designer, and mentor while the class owners do much of the coding of the feature [8].

5.2 Strengths and Limitations of FDD

Strengths: FDD has strong modeling features that have a bias towards ULM. It provides detailed guidelines for multi-team projects, system design, development, and technical practices [27].

Limitations: FDD focuses only on design and implementation, hence there is need for other supporting approaches [1]. Requires the engagement of highly experienced experts in modeling. There is lack of guidelines that define the business enterprise, concept development, requirements, and risk control [27].

5.3 Project Features

Team size: Team size varies from four to twenty people depending on the complexity of the feature at hand.

Iteration length: One to four weeks [16].

Support for distributed teams: FDD is designed for multiple teams and, while it does not have built-in support for distributed environments, it should be adaptable.

Criticality: The FDD prescription does not specifically address project criticality.

Application: First used for a large complex banking project [19]. Literature on FDD experience is hard to get.
6 DYNAMIC SOFTWARE DEVELOPMENT METHODOLOGY (DSDM)

According to their website (www.dsdm.org) DSDM, is more of a framework than a methodology. It was developed in the early 1990s. Jim Highsmith[10] calls it a formalization of Rapid Application Development (RAD) practices. According to Rietmann [23] however, DSDM is sometimes called a methodology because it defines a process and a couple of to be produced products during the DSDM approach. Nonetheless, these products are very global and can be adapted for every technique and industry environment. The DSDM process provides strong emphasis for project management activities. Planning is inherent in each phase, as plans evolve based on the increments and their results. The method also provides scripts that define management activities throughout the life cycle [27].

6.1 DSDM Life Cycle

The DSDM development process framework’s five stages are defined by Rietmann [23] as:

Feasibility study: DSDM stresses that the feasibility study should be just a few weeks. Along with the usual feasibility activities, this phase should determine whether DSDM is the right approach for the project.

Business study: The business study phase is “strongly collaborative, using a series of facilitated workshops attended by knowledgeable and empowered staff that can quickly pool their knowledge and gain consensus as to the priorities of the development” [dsdm.org]. The result of this phase is the Business Area Definition, which identifies users, markets, and business processes affected by the system.

Functional model iteration: The functional model iteration aims to build on the high level requirements identified in the business study. The DSDM framework works by building a number of prototypes based on risk and evolves these prototypes into the complete system. This phase and the design and build phase have a common process that identifies what is to be produced, agrees on how and when to do it, creates the product, checks that it has been produced correctly.

Design and build iteration: The prototypes from the functional model iteration are completed, combined, and tested and a working system is delivered to the users.

Implementation: During this phase, the system is transitioned into use. An Increment Review Document is created during implementation that discusses the state of the system. If there is still work to be done on the system, the functional model, design and build, and implementation phases are repeated until the system is complete.

DSDM was developed with the aim of preventing project failure. It is based on two assumptions; that nothing is built perfectly the first time (based on eighty - twenty rule) and that each process step needs to be completed just enough to move on to the next step, because requirements will change anyway as understanding increases and the current step can be finished in a later iteration [23]. The foundations of DSDM spell out the need for user involvement and a team empowered to make decisions that will lead to the rapid delivery of products that are fit for business purpose [23].

6.2 Strengths and Limitations of DSDM

Strengths: Highly dependable because it is strongly based on Rapid Application Development (RAD) principles. DSDM boasts of well defined guidelines for the business system, multi-team projects, single-team projects, concept development, requirements, project management processes, and issues of risk control [27].

Limitations: The details of the methodology and its whitepapers with research information are available only to DSDM consortium members [1]. There may be a tendency for bureaucracy in the DSDM process due to the relatively large number of roles in its team.

6.3 Project Features

Team size: Varies from two to six people, minimum requirement is one user and one developer and the maximum requirement of six is found from experience [1]. A project may however have more than one team. Because of DSDM’s framework nature, it does not specifically address, exact iteration lengths, distributed teams support, or system criticality [13].

Application: Has been in use in Europe since the mid 1990s. DSDM has proved to be a good alternative RAD.

7 ADAPTIVE SOFTWARE DEVELOPMENT (ASD)

Adaptive Software Development is one of the modern software development methodologies that are geared towards today’s high speed high change e-business projects. Highsmith [12] outlines the ASD philosophy as being rooted in an underlying conceptual base of complex adaptive systems (CAS) theory, which Brian Arthur and his colleagues at the Santa Fe Institute have used to revolutionize the understanding of physics, biology, evolution, and economics. It is rooted in agents, self-organization, and emergent outcomes. ASD was designed for extreme projects in which high-speed, high-change, and uncertainty reign. Though many IT projects are not extreme, for those that are extreme, ASD fits well.

ASD practices are driven by a belief in continuous adaptation and a different life cycle model (adaptive life cycle) from the traditional models that were process driven. This model is based on the incremental/iterative development concept, which is basically the process for
constructing several partial deliverables, each having incrementally more functionality. The real drive behind the adaptive life cycle is change management. ASD has combined forces with Crystal and between these two lies the profound philosophy of Agile methodologies.

7.1 ASD Life Cycle

Highsmith [14] believes that for many e-business projects the final results may be fuzzy in the beginning, but the overall mission that guides the team is always well articulated. He therefore declares that mission artifacts provide direction and are also used in making critical decisions in a project, hence mission focus is one of the main characteristics of the Adaptive Lifecycle, which are; mission focused, component based, iterative, time-boxed, risk-driven, and change tolerant.

Application Components: a group of features (or deliverables excluding documents) to be developed during an iterative cycle.

Iterative Cycles: the emphasis here is on re-doing, which is considered to be just as important as doing.

Time-boxing: this is not about time deadlines though the name may suggest otherwise, but it is all about focusing and forcing hard tradeoff decisions. So it actually forces the project team and the customers to re-evaluate the validity of the project’s mission.

Critical Risk: this involves the analysis of critical risks to the entire project and making decisions on averting the risks.

Change Tolerant: accepting inevitable changes from the customer as an advantage not a problem.

These characteristics are implemented within ASD’s three phases; speculate, collaborate, and learn as defined below.

Speculating involves the following:

- **Initiation**, i.e. setting project: mission, objectives, constraints, organization, key players, requirements, size and scope, and risks.
- **Project Time-box**, time is based on the scope, requirements, estimates and resources.
- **Cycles**, the length of a cycle is determined by the overall project schedule and the degree of uncertainty.
- **Objective**, write an objective statement for each cycle.
- **Components per Cycle**, components must be assigned to each cycle to provide visibility to the development team.
- **Technology & support Components**, this is done to make sure each cycle delivers something useful to the customer and to balance resource utilization.
- **Task list**, develop a project task list based on the components.

Collaborate: involves striking a balance between managing and doing. A good example of collaboration is found in XP’s pair programming and collective ownership of code.

Learn: involves Collaborative work in building products. Learning activities expose the products to a variety of stakeholders to ascertain value. The stakeholders in this case are groups and practices such as: customer focus, technical reviews, beta testing and postmortems. Learning challenges the assumptions made and encourages the teams to use the results of each development cycle to learn the direction of the next [14].

7.2 Strengths and Limitations of ASD

Strengths: ASD practices are based on the philosophy of complex self-adaptive systems, which makes it very strong in the non-technical aspects of software development that affect the development process and the final product. ASD has well defined risk control guidelines [27].

Limitations: ASD Methodology has little to do with the software practice and mainly provides a management culture for controlling complex adaptive systems. Does not provide guidelines for individual development projects. Technical and measurement practices are also not addressed [27].

7.3 Project Features

**Team size:** ASD team size is determined by the scope and the size of the project, which can range from small, medium to large [10].

**Iteration length:** Each iteration is time-boxed and the length is determined by the project schedule and the degree of uncertainty. For small to medium-sized applications, iterations usually vary from four to eight weeks [10].

Support for distributed teams: ASD does not specify co-located teams hence it can be applied to distributed teams though the collaboration phase becomes more challenging.

System criticality: ASD includes risk analysis and aversion techniques.

Applications: Has not been used as a methodology to develop a system, but has been merged with Crystal methodologies.

8 CRYSTAL METHODOLOGIES

Crystal Methodologies have a heavy bias towards planning and project management. Crystal Methodologies which were developed by Alistair Cockburn in the early 1990s. The fundamental problem that the Crystal Methodologies actually solve is poor communication facing product development. Cockburn’s philosophy is that face-to-face interactions can replace written documentation, which can reduce the reliance on written work products and improve the likelihood of delivering the system. This is possible with more frequent delivery of running, tested slices of the system [14].
8.1 Crystal Life Cycle

Highsmith [10] adds that Crystal Methodologies focus on people, interaction, community, skills, talents, and communication as first order effects on performance. Process remains important, but secondary. Cockburn’s methodologies are named “crystal” to represent a gemstone, i.e., each facet is another version of the process, all arranged around an identical core [10]. The different methodologies are assigned colors arranged in ascending cloudiness. The most Agile version is Crystal Clear, followed by Crystal Yellow, Crystal Orange, Crystal Red, etc. The version of crystal used depends on the number of people involved, which translates into a different degree of emphasis on communication.

As more people are added to the project, there is a translation towards more opaque versions of crystal. As project criticality increases, the methodologies become more rigorous. The methodologies can also be altered to fit other priorities, such as productivity or legal liability.

All Crystal methodologies begin with a core set of roles, work products, techniques, and notations, and this initial set is expanded as the team grows or the methodology gets more rigorous. As a necessary effect, more restraints lead to a less Agile methodology, but Highsmith stresses that they are Agile nonetheless because of a common mindset [10].

8.2 Strengths and Limitations of Crystal

Strengths: communication happens to be one of the major variables that determine the success of a software project or any other project for that matter. Crystal methodologies are based on a philosophy that strives to solve communication problems. There are well-defined guidelines for multi-team and single-team projects. Risk control and technical practices are also well defined [27].

Limitations: of all the proposed Crystal methodologies only two exist, which makes it difficult to identify limitations [1]. An obvious limitation is the lack of guidelines for the business enterprise [27].

8.3 Project Features

Team size: The Crystal Family accommodates any team size; however, Cockburn puts emphasis on the presence of highly skilled and experienced people.

Iteration length: Up to four months for large, highly critical projects.

Support for distributed teams: Crystal Methodologies have been built in support for distributed teams.

System criticality: Crystal supports four basic criticalities: failure resulting in loss of comfort, discretionary money, essential money, and life.

Application: Crystal clear has been used in a number of projects including Internet banking.

9 LEAN SOFTWARE DEVELOPMENT (LD)

Lean Software Development like DSDM and Scrum is more a set of project management practices than a definite process. It was started by Bob Charette and it draws on the success that Lean Manufacturing gained in the automotive industry in the 1980s. While other Agile methodologies look to change the development process, Charette believes that to be truly Agile there is need to change how companies work from the top down. Lean Development is targeted at changing the way CEOs consider change with regards to management of projects. LD is based on lean thinking whose origins are found in lean production started by Toyota Automotive manufacturing company [21].

9.1 LD Life Cycle

Lean Software Development’s twelve principles focus on management strategies [10]:

- Satisfying the customer is the highest priority.
- Always provide the best value for the money.
- Success depends on active customer participation.
- Every LD project is a team effort.
- Everything is changeable.
- Domain, not point, solutions.
- Complete, do not construct.
- An eighty percent solution today instead of one hundred percent solution tomorrow.
- Minimalism is essential.
- Needs determine technology.
- Product growth is feature growth, not size growth.
- Never push LD beyond its limits.

These principles are derived from seven lean principles which Poppendieck and Poppendieck [21] define as follows:

Eliminate Waste: Waste is anything that does not add value to a product, value as perceived by the customer. In lean thinking, the concept of waste is a problem. If a component is sitting on a shelf gathering dust, that is waste. If a development cycle has collected requirements in a book gathering dust, that is waste. The ideal is to find out what a customer wants, and then make or develop it and deliver exactly what they want, virtually immediately.

Amplify Learning: Development is an exercise in discovery, while production is an exercise in reducing variation, and for this reason, a lean approach to development results in practices that are quite different than lean production practices. Development is like creating a recipe, while production is like making the dish.
Decide as Late as Possible: Development practices that provide for late decision-making are effective in domains that involve uncertainty, because they provide an options-based approach. In the face of uncertainty, most economic markets develop options to provide a way for investors to avoid locking in decisions until the future is closer and easier to predict. In other ways decide based on fact, not speculation.

Deliver as Fast as Possible: Rapid development has many advantages. Without speed you cannot delay decisions. Without speed you do not have reliable feedback. In development, the discovery cycle is critical for learning: design, implement, feedback, and improve. The shorter these cycles are the more that can be learned. Speed assures that the customer gets what they need now, not what they needed yesterday. It also allows them to delay making up their mind about what they really want until they know more. Compressing the value stream to as short a service time as possible is a fundamental lean strategy for eliminating waste.

Empower the Team: Top-notch execution lies in getting the details right, and no one understands the details better than the people who actually do the work. Involving programmers in the details of technical decisions is fundamental to achieving excellence. The people on the front line combine the knowledge of the minute details with the power of many minds. When equipped with necessary expertise and guided by a leader, they will make better technical decisions and better process decisions than anyone can make for them. Because decisions are made late and execution is fast, it is not possible for a central authority to orchestrate activities of workers. Thus, lean practices use pull techniques to schedule work, and contain local signaling mechanisms so workers can let each other know what needs to be done. In lean software development, the pull mechanism is an agreement to deliver increasingly refined versions of working software at regular intervals. Local signaling occurs through visible charts, daily meetings, frequent integration and comprehensive testing.

Build Integrity In: A system is perceived to have integrity when a user thinks – “YES! That is exactly what I want. Somebody got inside my mind!” Market share is a rough measure of perceived integrity for products, because it measures customer perception over time. Conceptual integrity means that the system's central concepts work together as a smooth, cohesive whole and it is a critical factor in creating perceived integrity. Software needs an additional level of integrity – it must maintain its usefulness over time. Software is usually expected to evolve gracefully as it adapts to the future. Software with integrity has a coherent architecture, scores high on usability and fitness for purpose, is maintainable, adaptable and extensible. Research has shown that integrity comes from wise leadership, relevant expertise, effective communication and healthy discipline; processes, procedures, and measurements are not adequate substitutes.

See the Whole: Integrity in complex systems requires a deep expertise in many diverse areas. One of the most intractable problems with product development is that experts in any area (e.g. database or GUI) have a tendency to maximize the performance of the part of the product representing their own specialty, rather than focusing on overall system performance. It is challenging to implement practices that avoid sub-optimization in a large organization, and an order of magnitude more difficult when contracts are involved.

9.2 Strengths and Limitations of LD

Strengths: LD approach to change focuses on the CEO, which if accepted would prove more successful than any other Agile methodology. Guidelines on business system concerns are well defined, and the risk control is also provided for in detail [27].

Limitations: The methodology allows little change in requirements [7]. Though literature in Lean Thinking has been in existence for decades, its application to software development has limited literature, with the first book having been published in May 2003. the method has no guidelines for individual development and single team projects. The development, technical practices and measurement practices are also not defined [27].

9.3 Project Features

Since Lean Development is more of a management philosophy than a development process, team size, iteration length, team distribution, and system criticality are not directly addressed [13].

Application: Based on Lean Manufacturing with origins as far back as 1935, LD has been applied in a number of organizations [21].

10 AGILE MODELING (AM)

Agile Modeling (AM) is proposed by Scott Ambler [3]. It is a methodology based on values, principles and practices that focus on modeling and documentation of software. AM recognizes that modeling is a critical activity for a project success and addresses how to model in an effective and Agile manner [2]. The three main goals of AM are:

- To define and show how to put into practice a collection of values, principles and practices that lead to effective and lightweight modeling.
- To address the issue on how to apply modeling techniques on Agile software development processes.
- To address how modeling techniques can be applied effective independently of the software process in use.

10.1 Agile Modeling Life Cycle
AM is not a complete software development methodology. Instead, it focuses only on documentation and modeling and can be used with any software development process. The work starts with a base process which is tailored to use AM. Ambler [3] illustrates, for example, how to use AM with both XP and Unified Process (UP). The values of AM include those of XP – communication, simplicity, feedback and courage – and also include humility. It is critical for project success to have effective communication in a team and also with the stakeholder of the project.

10.2 Strengths and Limitations of AM

Strengths: AM is potentially adaptable for use with any modeling techniques.

Limitations: Can only be used with other Agile methodologies [1]. It is quite new (introduced in 2002), there is a lot that is not yet clear about the methodology.

10.3 Project Features

Since AM is not a complete software development methodology and should be used with other development methodologies, the team size, exact iteration lengths, distribution and system criticality will depend on the development process being used.

Application: No record of current or previous use of the methodology in the reviewed literature.

11 HARNESSING AGILITY FOR COMPETITIVE ADVANTAGE

This review is aimed at giving vital information about some of the new technologies in software development in order to assist organizations that want to utilize these new technologies to their competitive advantages. It is however, important to note that Agile methodologies are not based on new concepts but some of the principles have been in use since the 1970s. All Agile methodologies derive their strengths from iterative and incremental development, concepts which have been in use in the software development field for decades. Larman [28] in fact classifies software development methodologies into two groups namely; Iterative Incremental Development (IID) and Waterfall. He then mentions that Agile methodologies are a subset of the IID group.

We have not included the Rational Unified Process (RUP), Evo, Team Software Process (TSP), Personal Software Process (PSP), Capability Maturity Model (CMM), Capability Maturity Model Integration (CMMI), Capability Maturity Model for Software (SW-CMM), Cleanroom and a few others in this survey for the simple reason that they do not belong to the Agile Alliance though they have a lot in common with Agile methodologies and in fact belong to the bigger group of IID.

While limitations of each Agile methodology have been mentioned under their respective sections we would like to mention some limitations that are common to the Agile group. You have probably noticed that some of the limitations may apply even to the waterfall group.

Organizations that are keen on using Agile methodologies or have already started using them partially or in full may do well to consider the following limitations:

- Due to the high level of independence and discipline required in the use of Agile methodologies a highly skilled and experienced team is required. If that is not possible then include at least one highly skilled and experienced member. As Ken Schwaber [26] often mentions in his Scrum training sessions, a bunch of novices will delivery a useless product, the choice is yours. Personnel of that caliber may be difficult to get in South Africa because they are very few.

- Agile methodologies were originally meant for small to medium collocated teams of up to ten people and certainly not exceeding twenty. There has been a lot of work in scaling up of teams, and some XP and Crystal projects have used up to five hundred people successfully [15]. Depending on your project priorities you may decide to include more process ceremony or a heavier methodology according to Cockburn’s framework [29].

- One of the limitations to be cautious of according to Bradley [30] is the apparent serious problem with the no-design-up-front schools of Agile development, i.e. evolving from simpler mechanisms frequently results in less than optimal (and not infrequently deeply flawed) end products. You may assume that these kinds of problems can be solved through refactoring, but refactoring may have serious limitations as you move further into code and deployment where it becomes harder to change things. Bradley [30] suggests that an up front design through domain modeling as implemented in FDD (one of the Agile methodologies) combined with refactoring may solve the problem.

- Agile methodologies have a limited support for subcontracting when it comes to outsourcing of software development. Contracts are often based on a plan that includes a process, with milestones and deliverables in sufficient detail to determine a cost estimate [32]. However, if the subcontractor has a good reputation, a contract can be written to allow some degree of flexibility in how the product is developed within time and cost constraints. Such a contract usually has two parts: a fixed part that defines a framework that constrains how changes will be incorporated into the product and the variable part which defines the requirements and deliverables.
that can vary within the boundaries defined by the fixed part [32]. For more details on Agile contracts read references [21, 32-35].

All we are trying to lay forward to the potential Agile users is that Agile processes work well, but we need to recognize that there are situations where they don't work well though some may work better than others. There is a lot of subjectivity in other limitations that Agile methodologies have, but for a more definitive analysis on Agile processes read references [27, 28, 32].

12 CONCLUSION

The purpose of this paper was to give an overview of existing literature on Agile methodologies. In this paper each methodology was described in a way that reveals the historical beginnings, the process and how the methodology defines team size, iteration length, support for distributed teams, system criticality and use in industry. Limitations of each of the methodologies were also mentioned. In section eleven we then gave a challenge to organizations that are interested in using Agile methodologies.

This review also shows that there is a lot of discipline in the Agile approach as opposed to what some critics think. We have also found that Agile methodologies have obvious strengths in the development of e-applications using small teams and characterized by short development timeframes. The waterfall-based approaches are stronger in critical systems, large projects, projects with relatively stable requirements and longer development timeframes. Most of the literature also reflects that most organizations that changed their processes to Agile did so after some of their projects hit some serious snags such as budget overruns, delivery time extensions, scope creep, etc. Few organizations in South Africa have tried Agile methodologies. You may want to read more Agile success stories from [10, 15, 27], so as to consider their relevance to your specific situations.

Finally after some meaningful research work that we have done in the utilization of emerging computing technologies [31] we feel that it is important for organizations that want to lead through innovation and change management to assume a balanced approach between the adaptation of emerging technologies and their existing legacy technologies. Our own idea of agility would therefore be that:

- The organization has the ability to customize its own way of working so as to balance between customer requirements and the development team’s needs as the product evolves.
- There be a balance between extreme chaos (i.e. undisciplined agility) and extreme predictive planning.
- The way forward is determined by the values of the project as set by the customer at every stage of the project’s development.
- The developer is flexible enough to deliver necessary documentation when asked to do so by the customer without drawing heavily on the project’s budgetary constraints.
- The development team must work in such a way that they deliver on time, but still have enough energy and courage to tackle the next project probably using a similar methodology.

13 REFERENCES


