IOSR Journal of Engineering (IOSR JEN) **www.iosrjen.org**
ISSN (e): 2250-3021, ISSN (p): 2278-8719
PP 84-92

An overview of Agile Software Development and Future Scope

Manishakaikadi[1], Dhiraj Rane[2]
MCA 6th sem, Department Of Computerscience, GHRIIT, Nagpur
Asst. Prof, Department Of Computer Science, GHRIIT, Nagpur

Abstract: *Agile software development has rapidly gained a lot of interest in the field of software engineering. Agile software development, despite its novelty, is an important domain of research within software engineering discipline. Agile software development methods have caught the attention of software engineers and researchers worldwide. Scientific research is yet scarce, there has been little detailed reporting of the usage, penetration and success of agile methodologies in traditional, professional software development organizations. Report on the results of an empirical study conducted at Microsoft to learn about agile development and its perception by people in development, testing, and management. This paper reports results from a study, which aims to organize, analyses and make sense out of the dispersed field of agile software development methods.*
Keywords: *Agile, Scrum, Kanban, DSDM Atern, DAD, AUP.*

I. Introduction

Agile is set of values and principles.Agile use to developing general's specific software and tools. **Agile software development** is an approach to software development under which requirements and solutions evolve through the collaborative effort of self-organizing and cross-functional teams and their customer(s)/end user(s). It advocates adaptive planning, evolutionary development, early delivery, and continual improvement, and it encourages rapid and flexible response to change. The term agile (Agile sometimes written) was popularized, in this context, by the ManifestoforAgileSoftwareDevelopment. The values and principles espoused in this manifesto were derived from and underpin a broad range of software development frameworks, including Scrum and Kanban. There is significant anecdotal evidence that adopting agile practices and values improves the agility of software professionals, teams and organizations; however, some empirical studies have found no scientific evidence. All Agile methodologies share a set of core ideas which are prescribed from method to method in subtly different ways; iterative and incremental delivery of working code, frequent collaboration with stakeholders, closely working, self-organizing teams, and the ability to embrace change late in the project. Agile methods are shamelessly incestuous, borrowing from each other and using existing ideas in slightly different ways so it becomes very difficult to tell whether a project is following any one method as even the slightest adaptation of any aspect of a process can make it seem more like another. Fortunately, the majority of the concepts are compatible so that it becomes very easy to use Agile methods like a smorgasbord, picking and choosing parts at will Some Agile methods have little up-front design effort which can lead to considerable rework or integration problems. User involvement is also a double-edged sword when frequently changing ideas lead to more requirements churn than even Agile processes are prepared for.

II. Agile Software Development Values

Based on their combined experience of developing software and helping others do that, the seventeen signatories to the manifesto proclaimed that they value.

- **Individuals and Interactions** over processes and tools
- **Working Software** over comprehensive documentation
- **Customer Collaboration** over contract negotiation
- **Responding to Change** over following a plan

That is to say, the items on the left are valued more than the items on the right.

- Tools and processes are important, but it is more important to have competent people working together effectively.
- Good documentation is useful in helping people to understand how the software is built and how to use it, but the main point of development is to create software, not documentation.
- A contract is important but is no substitute for working closely with customers to discover what they need.
- A project plan is important, but it must not be too rigid to accommodate changes in technology or the environment, stakeholders' priorities, and people's understanding of the problem and its solution.

III. Agile Software Development Principles

The Manifesto for AgileSoftwareDevelopment is based on 12 principles:

1. Customer satisfaction by early and continuous delivery of valuable software.
2. Welcome changing requirements, even in late development.
3. Deliver working software frequently (weeks rather than months)
4. Close, daily cooperation between business people and developers
5. Projects are built around motivated individuals, who should be trusted
6. Face-to-face conversation is the best form of communication (co-location)
7. Working software is the primary measure of progress
8. Sustainable development, able to maintain a constant pace
9. Continuous attention to technical excellence and good design
10. Simplicity—the art of maximizing the amount of work not done—is essential
11. Best architectures, requirements, and designs emerge from self-organizing teams
12. Regularly, the team reflects on how to become more effective, and adjusts accordingly

IV. Overview

Iterative, incremental and evolutionary

Most agile development methods break product development work into small increments that minimize the amount of up-front planning and design. Iterations, or sprints, are short time frames that typically last from one to four weeks. Each iteration involves a cross-functional team working in all functions: planning, analysis, design, coding, unit testing, and acceptance testing. At the end of the iteration a working product is demonstrated to stakeholders. This minimizes overall risk and allows the product to adapt to changes quickly an iteration might not add enough functionality to warrant a market release, but the goal is to have an available release (with minimal bugs) at the end of each iteration.Multiple iterations might be required to release a product or new features. Working software is the primary measure of progress.

Efficient and face-to-face communication

The principle of co-location is that co-workers on the same team should be situated together to better establish the identity as a team and to improve communication. This enables face-to-face interaction, ideally in front of a whiteboard, that reduces the cycle time typically taken when questions and answers are mediated through phone, persistent chat, wiki, or email.

No matter which development method is followed, every team should include a customer representative ("Product Owner" in Scrum). This person is agreed by stakeholders to act on their behalf and makes a personal commitment to being available for developers to answer questions throughout the iteration. At the end of each iteration, stakeholders and the customer representative review progress and re-evaluate priorities with a view to optimizing the return on investment (ROI) and ensuring alignment with customer needs and company goals.

In agile software development, an **information radiator** is a (normally large) physical display located prominently near the development team, where passers-by can see it. It presents an up-to-date summary of the product development status. A build light indicator may also be used to inform a team about the current status of their product development.

V. Agile Vs. Waterfall

One of the differences between agile software development methods and waterfall is the approach to quality and testing. In the waterfall model, there is always a separate testing phase after a build phase; however, in agile software development testing is completed in the same iteration as programming.

Another difference is that traditional "waterfall" software development moves a project through various Software Development Lifecycle (SDLC) phases. One phase is completed in its entirety before moving on to the next phase.

Because testing is done in every iteration—which develops a small piece of the software—users can frequently use those new pieces of software and validate the value. After the users know the real value of the updated piece of software, they can make better decisions about the software's future. Having a value retrospective and software re-planning session in each iteration—Scrum typically has iterations of just two weeks—helps the team continuously adapt its plans so as to maximize the value it delivers. This follows a pattern similar to the PDCA cycle, as the work is planned, done, checked (in the review and retrospective), and any changes agreed are acted upon.

This iterative approach supports a product rather than a project mindset. This provides greater flexibility throughout the development process; whereas on projects the requirements are defined and locked

down from the very beginning, making it difficult to change them later. Iterative product development allows the software to evolve in response to changes in business environment or market requirements.
Because of the short iteration style of agile software development, it also has strong connections with the learn start up concept.

Agile software development methods support a broad range of the software development life cycle. Some focus on the practices (e.g., XP, pragmatic programming, agile model), while some focus on managing the flow of work (e.g., Scrum, Kanban). Some support activities for requirements specification and development (e.g., FDD), while some seek to cover the full development life cycle (e.g., DSDM, RUP).
Popular agile software development frameworks include (but are not limited to):

- Adaptive software development (ASD)
- Agile model
- Agile unified process (AUP)
- Disciplined agile delivery
- Dynamic systems development method (DSDM)
- Extreme programming (XP)
- Feature-driven development (FDD)
- Lean software development
- Kanban
- Rapid application development (RAD)
- Scrum

1. Agile Management

The term agile management is applied to an iterative, incremental method of managing the design and build activities of engineering, information technology and other business areas that aim to provide new product or service development in a highly flexible and interactive manner, based on the principles expressed in the Manifesto for Agile Software Development.

Agile X techniques may also be called extreme project management. It is a variant of iterative life cycle where deliverables are submitted in stages.

The main difference between agile and iterative development is that agile methods complete small portions of the deliverables in each delivery cycle (iteration), while iterative methods evolve the entire set of deliverables over time, completing them near the end of the project.

Both iterative and agile methods were developed as a reaction to various obstacles that developed in more sequential forms of project organization. For example, as technology projects grow in complexity, end users tend to have difficulty defining the long-term requirements without being able to view progressive prototypes. Projects that develop in iterations can constantly gather feedback to help refine those requirements.
Agile management also offers a simple framework promoting communication and reflection on past work amongst team members. Teams who were using traditional waterfall planning and adopted the agile way of development typically go through a transformation phase and often take help from agile coaches who help guide the teams through a smooth transformation.

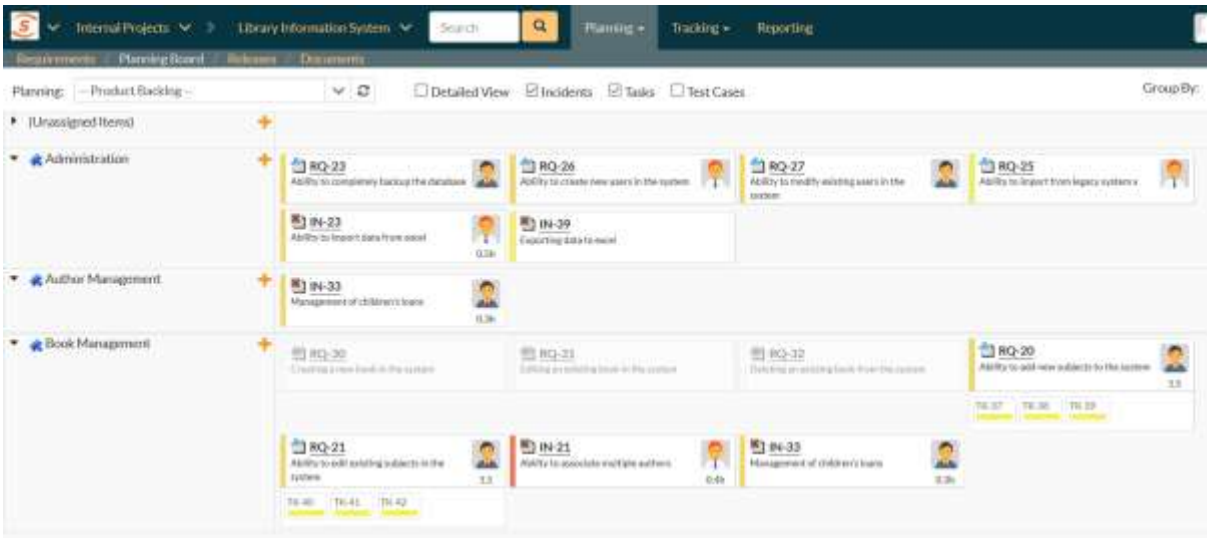
There are typically two styles of agile coaching: push-based and pull-based agile coaching. Agile management approaches have also been employed and adapted to the business and government sectors.
Agile methods are mentioned in the Guide to the Project Management Body of Knowledge (PMBOK Guide) under the Project Lifecycle.

2. Scrum

Scrum has become one of the ‘go-to’ techniques for those striving to become Agile. Along with XP, it is perhaps the most well-known of the Agile methods, and also the most precisely defined which means that there is a lot of documentation and pre-built process for teams that are willing to adopt the methodology completely.

However, Scrum’s greatest strength is also its greatest weakness. With specifically described roles, short iterations, tight schedules, daily meetings and an insistence on a release-quality product after each iteration (or sprint in Scrum terms), Scrum can seem so very foreign to inexperienced teams that it can be difficult to achieve early success unless you have some in-house knowledge.

For those who are old-hands at the method however, it can prove quite effective. As with Agile methods generally, requirements begin as a simple prioritized list of needs with little detail, known as the backlog. Only enough of these needs are pulled from the backlog to fill a single sprint and to begin their short journey from refinement to tested software. A typical product backlog is shown below:



(Unassigned Items)

Administration
RQ-23 Ability to completely backup the database
RQ-26 Ability to create new users in the system
RQ-27 Ability to modify existing users in the system
RQ-25 Ability to import from legacy systems
IN-23 Ability to import data from excel 0.3h
IN-39 Exporting data to excel

Author Management
IN-33 Management of children's loans 0.3h

Book Management
RQ-30 Creating a new book in the system
RQ-31 Editing an existing book in the system
RQ-32 Deleting an existing book from the system
RQ-20 Ability to add new subjects to the system 13
TK-37 TK-38 TK-39
RQ-21 Ability to edit existing subjects in the system 13
TK-40 TK-41 TK-42
IN-21 Ability to associate multiple authors 0.4h
IN-33 Management of children's loans 0.3h

Figure1. Scrum View

On their way, these needs will be elaborated by stakeholders, discussed in daily meetings (scrums), in which they are subjected to design proposals, used to create test cases, implemented, tested and delivered to stakeholders to review and provide input for subsequent sprints - and all this in just 2 to 4 weeks. Progress throughout the sprint is measured by the number of story points left to complete in that sprint and displayed using a 'burndown chart'. A story point is an arbitrary measure of the complexity or effort required to implement a user story. The only requirement is that story points are consistently applied. The rate at which story points are completed is called the velocity. After each sprint there is a review and weaknesses in the process or team performance are identified and changes made. In this way the project should improve from one sprint to the next.

The three most prominent roles in Scrum are the ScrumMaster, the Product Owner and the team member. The ScrumMaster acts as a super team leader, (but not a team member) keeping the team focused on the goals of the sprint, ensuring Agile principles continue to be followed, working with the Product Owner to keep the backlog up to date, and resolving any issues which might put the schedule at risk.
While Scrum does not describe the whole process, what it does define it does so very specifically. To fill the gaps, most projects pull in ideas from other processes resulting in Agile projects that use a hybrid of more than one concept.

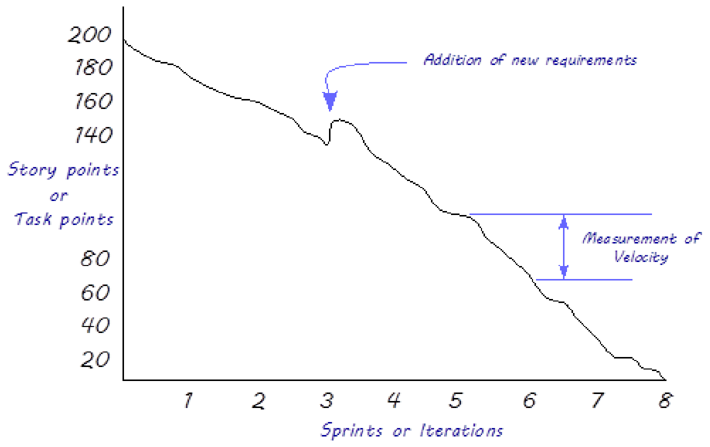


Figure2. Typical burndown chart

3. Extreme Programming (XP)

The idea developed by Kent Beck was to use best programming practices but take them to the extreme – hence the name. As such, none of the individual concepts of XP are really new, but their application and combination is what makes XP different, or at least did in the 2000s, which is when it initially became popular.

XP advocates adaptability; recognizing that each project faces different problems, applying those methods that will work and discarding those that will not. Partial XP therefore becomes a true possibility, often combined with Scrum, which has gaps in its definition particularly suited to adoption of XP practices.

XP embraces standard Agile principles such as short iterations; requirements being loosely defined with user stories until they are needed; almost immediate and regular testing; close, on-site stakeholder involvement with a rapid feedback loop; and team responsibility for success or failure.

As is necessary for Agile projects, the XP team expects and accepts changes in requirements based on feedback from stakeholders and the obligatory high level of communication within the team. Intense communication is necessary because everyone is required to understand the system as a whole so that anyone can work on and change any of the code. This fact alone makes XP difficult (but not impossible) to apply to large projects. Quickly written code is often used as a means to explain ideas and try out designs. With short iterations comes regular layering of new functionality, thus refactoring is encouraged in order to consolidate separately written parts into a more efficient whole.

Testing taken to the extreme means using as many testing techniques as necessary, as often as possible. Methods include unit testing, acceptance testing, integration testing and test-first programming, (although this is debatably a coding technique not a testing technique.) Initially, daily integration testing was advocated, however this has sometimes proven impractical and the frequency practiced is often longer, perhaps weekly; a good example of taking from XP only that which works for a particular project.

Finally: simplicity. The simplest solutions are encouraged, addressing the immediate problems, not problems that might arise tomorrow. This helps everyone to remain focused and meet deadlines but can result in considerable redesign and refactoring over time.

One of the reasons XP is so popular is its flexible nature. It is easy to combine features of XP with other ideas; indeed, XP is more about technique than process and so dovetails well with process-centric approaches such as that of Scrum.

4. Kanban

Kanban has the dubious distinction of being Agile without being iterative. Processes like Scrum have short iterations which mimic a project lifecycle on a small scale, having a distinct beginning and end for each iteration. Kanban requires that the software be developed in one large development cycle. Considering this, it is surprising to learn that Kanban is considered Agile at all, and yet it fulfils all twelve of the principles behind the Agile manifesto, because while it is not iterative, it is incremental.

The principle behind Kanban that allows it to be incremental and Agile, is limited throughout. With no iterations a Kanban project has no defined start or end points for individual work items; each can start and end independently from one another, and work items have no pre-determined duration for that matter. Instead, each phase of the lifecycle is recognized as having a limited capacity for work at any one time. A small work item is created from the prioritized and un-started requirements list and then begins the development process, usually with some requirements elaboration. A work item is not allowed to move on to the next phase until some capacity opens up ahead. By controlling the number of tasks active at any one time, developers still approach the overall project incrementally which gives the opportunity for Agile principles to be applied. A typical Kanban board is illustrated below:

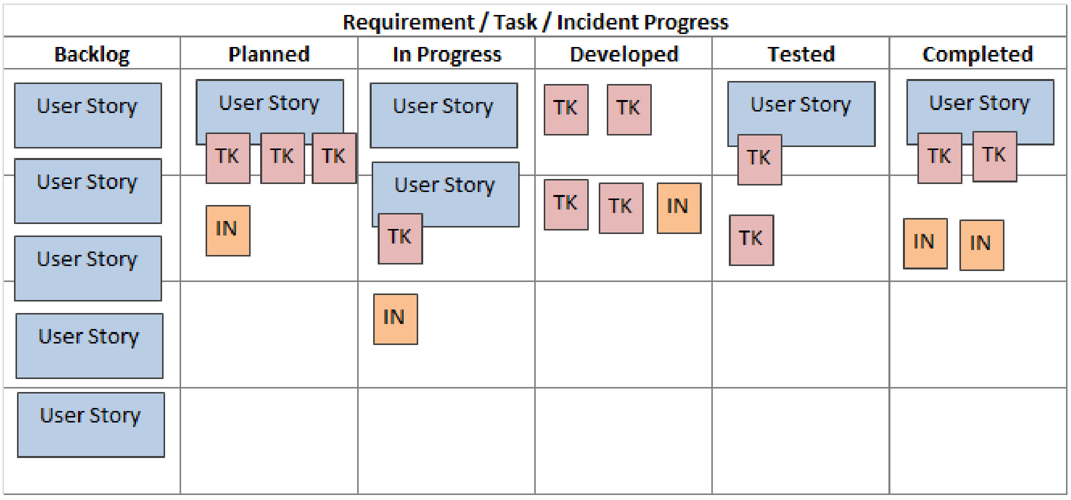


Figure3: Kanban Task View

Kanban projects have Work In Progress (WIP) limits which are the measure of capacity that keeps the development team focused on only a small amount of work at one time. It is only as tasks are completed that new tasks are pulled into the cycle. WIP limits should be fine-tuned based on comparisons of expected versus actual effort for tasks that complete.

Kanban does not impose any role definition as say, Scrum does and along with the absence of formal iterations, role flexibility makes Kanban attractive to those who have been using waterfall-style development models and want to change but are afraid of the initial upheaval something like Scrum can cause while being adopted by a development team.

5. DSDM/DSDM Atern (Incorporating RAD - Rapid Application Development)

Rapid Application Development (RAD) was championed by James Martin in his book of the same name in 1991, although the process had been around for some time before that. RAD is an iterative and incremental method which relies heavily on prototyping in order to obtain feedback from stakeholders. However, this requires early development of the GUI which can produce wasteful discarded versions and de-emphasize underlying functionality. RAD is not always considered an Agile approach and in 1994 the relative lack of discipline in the method led to the creation of the DSDM Consortium to develop a formalized independent RAD framework incorporating Agile principles. The result was the release of version 1 of the Dynamic Systems Development Method (DSDM) in1995 since when, it has continuously evolved leading to the launch of a new version in 2007 called DSDM Atern.

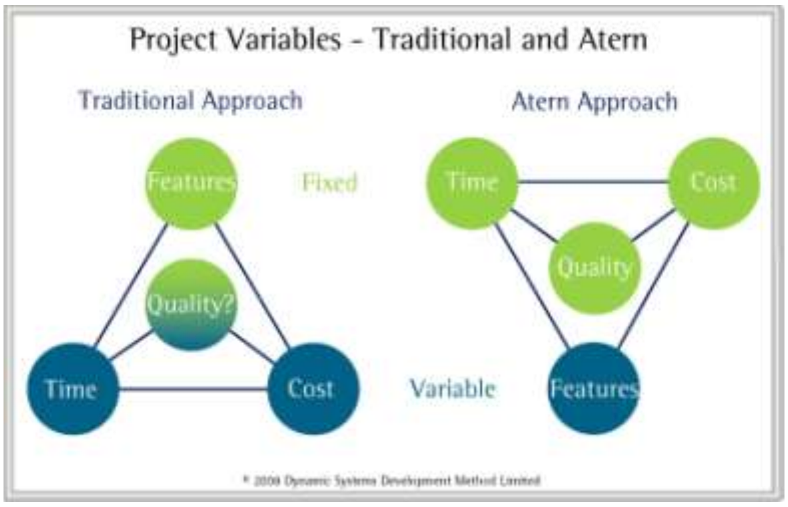


Figure.4:as variable in DSDM Atern.

Like most Agile methods, DSDM Atern puts quality and schedule first, leaving functionality as the lone variable. The method of prioritization used is called Moscow, offering four simple requirements categories:

- Must have;
- Should have;
- Could have; and
- Won’t have this time around.

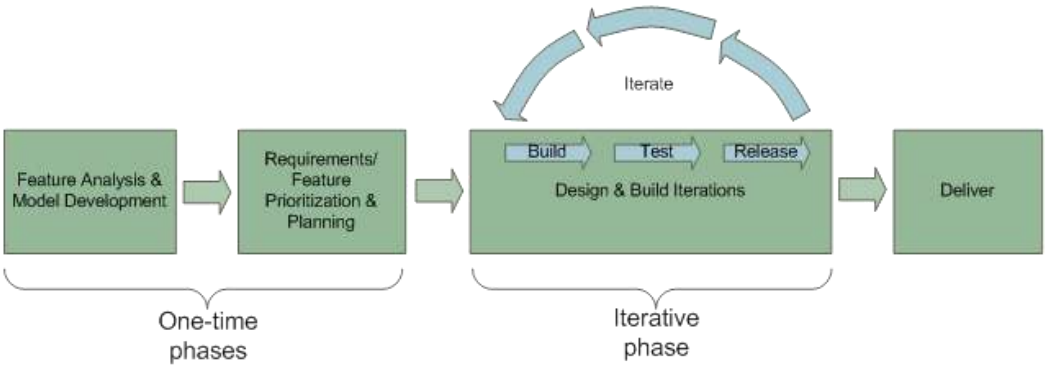

Iterate
Feature Analysis & Model Development
Requirements/ Feature Prioritization & Planning
Build
Test
Release
Design & Build Iterations
Deliver
One-time phases
Iterative phase

Figure 5: The relative sizes of pre and post-development phases varies between methods

Alongside all the standard principles that define Agile processes such as stakeholder involvement, and build early and often, DSDM Atern also advocates a degree of formal tracking and reporting which is not so common amongst Agile methods.

DSDM Atern addresses the narrow scope of some other methods such as Scrum by including pre and post-development phases in its purview making it a true project management process as opposed to a focused development process. It is also a method with a detailed process description and therefore it can take some time to embrace DSDM Atern fully.

6. Disciplined Agile Delivery (DAD)

To understand DAD well we need a little bit of history. In 1996 Rational Software Corporation created the Rational Unified Process (RUP) which was mostly a combination of UML and ideas developed by others such as Grady Booch and Jim Rumbaugh. RUP was a framework from which elements could be used for each project as necessary, which was a good job really as over the years it grew to be humongous and it is highly improbable that any single project used it all. The Agile Unified Process was developed in 2005 as a simplified version of RUP with work attributed to Scott Ambler who in 2012 wrote the book 'Disciplined Agile Development' with Mark Lines taking us from AUP to DAD.

A fair criticism of Scrum, XP and some other Agile methods is that they only address the software creation aspect of a project and leave other aspects, especially those at the start and end of the project, as a sort of 'exercise for the reader.' The DAD process framework attempts to expand the Agile influence beyond mere software generation to the entire product process while additionally addressing the needs of the enterprise such that the methods are scalable. As with its early forbears, DAD offers more than any single project would want and, in some cases, even proposes a number of alternative solutions from which to choose. Because DAD is rather comprehensive it is not easy to describe concisely in just a few paragraphs, but the following is a list of DADs characteristics summarized:

- Self-organizing, cross functional teams of individuals with multiple skills often called generalizing specialists;
- An environment that promotes learning to include user needs, how to improve processes and a growth in technical knowledge;
- An adherence to the principles of the Agile manifesto;
- The borrowing of ideas and principles from other Agile, iterative or lean methods such as XP, Kanban and Scrum;
- An expansion of the vision from just software to all aspects of a product such as hardware and user process improvement and in doing so, extending agile principles to all project disciplines;
- Encompassing the full project lifecycle from initiation to production delivery with an understanding that the nature of activities within iterations will change as the product matures;
- Sufficient guidelines to help those starting up but not too many to put them into straightjackets, essentially trying to provide balance between too little and too much guidance;
- Methods to address risk, evaluate product viability and ensure value through regular production of potentially deliverable solutions; and
- Accommodation of demands from the enterprise such as governance, corporate vision, and other active projects teams.

Being relatively new, it remains to be seen whether DAD gains sufficient popularity to earn its place as a 'standard' alongside Scrum and XP. Even if it is not adopted in full, DAD offers ideas that can be integrated into other project environments.

7. Agile Unified Process (AUP)

The Agile Unified Process (AUP) was developed in 2005 as a simplified version of RUP with work attributed to Scott Ambler. It was subsequently updated in 2006 before Scott Ambler moved on to the work which became Disciplined Agile Delivery (DAD). RUP is process-heavy, and although the AUP is intended to conform to all the principles of the Agile Manifesto, it debatable how well it succeeds. For example, Agile methods should support self-organizing teams with no management hierarchy, however it is hard to ignore the fact that the AUP has, to some degree, inherited the process-heavy nature of its parent, RUP. The counter-argument is that the AUP does not enforce any of the process guidelines it offers. However, Project Management is still a major discipline in the AUP and if you begin to leave bits out to claim 'agility' are you really following that process?

As with some other Agile methods, initial requirements elicitation are excluded as is any delivery process at the back end. The Agile Unified Process is more up-front loaded than most Agile methods, requiring a considerable amount of model before implementation begins, which in turn demands some degree of early requirements analysis. While this is not a bad thing, care must be taken not to go too far and do waterfall-like detailed requirements analysis. Further, the proposed incremental development releases between production releases are not necessarily production quality and so again, the lifecycle can appear more waterfall than Agile.

Although the AUP is a good option for those trying to apply some Agile techniques to large-scale software projects, Scott Ambler's relatively quick follow-up publication of the Disciplined Agile Delivery process in 2010 might be seen as an acknowledgement that the AUP was merely a stepping stone to something more viable.

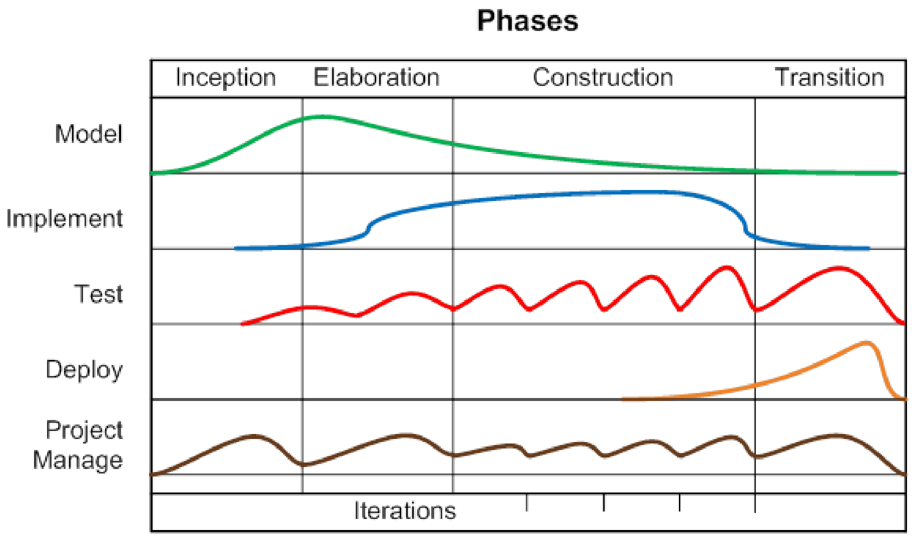


Figure 6: Effort expended over the lifecycle for some of the AUP disciplines

8 . Feature-driven Development

There are a number of factors which create some doubt as to whether Feature-driven Development is actually an Agile process, at least as it is defined by the Agile Manifesto. In FDD functionality is implemented iteratively and not according to business needs as might be decided by a Product Owner, but by functional area.
To see how requirements can be organized by functional area, they must be fairly well understood and so a design is created. From the design is derived a feature list at which point the iterative implementation cycles can begin. Consequently, and as with some other methods, the first two phases of the FDD process are not iterative but take place just once which narrows the iterative part of the project, and so it could be argued that the agility of the overall project is reduced. At the same time there are some real benefits to this approach. For example, with each functional area written all at once there is less need to refactor than with a process where the same code might be repeatedly changed. Not surprisingly, less refactoring is one of the benefits of traditional development methods so in some sense FDD can claim to combine both traditional and Agile with some success.

9 . Lean Software Development

Lean Software Development is less a process and more a set of principles to deliver by and consequently, the principles can be overlaid onto most processes that are truly Agile. For those familiar with Agile processes in general, some of the Lean philosophy seem very familiar. Reducing the time, it takes for work to flow through the process cycle matches very well with Agile's short iterations. Allowing teams to make decisions within acceptable constraints and encouraging individuals to take the initiative are both Agile values. The elimination of unnecessary and wasteful work is also a value common to both Lean and Agile ideals.
Lean Software Development is gaining interest as a method in its own right but to date remains most widely found as a facet of projects using other process

VI. Summary

While Agile methods have more similarities than differences, it is possible to pick out one or two distinguishing factors for each.

Scrum	Requires a ScrumMasterBurndown chartsLimited tocode production
XP	Pair programming Test-first programming Selectable ideas
Kanban	Incremental but not iterative Work In Progress (WIP) limits
DSDM Atern	MoSCoW prioritization Heavy in detail Addresses full lifecycle
DAD	Relatively new Addresses full life cycle Borrows from elsewhere
AUP	Grew from RUP Heavy on pre-iterative phases
Feature Driven Development	Implementation by functional area
Lean Development	Ideas to overlay on other process

VII. CONCLUSION

Agile software development is on the path of the rapid software development. It's providing support for extension of the new process models. Agile software development, despite its novelty, is an important domain of research within software engineering discipline and more research will be get in coming era. Scientific research is yet scarce, there has been little detailed reporting of the usage, penetration and success of agile methodologies in traditional, professional software development organizations. Report on the results of an empirical study conducted at Microsoft to learn about agile development and its perception by people in development, testing, and management. This paper reports results from a study, which aims to organize, analyses and make sense out of the dispersed field of agile software development methods

Reference

[1]. https://www.agilealliance.org/agile101/

[2]. https://en.wikipedia.org/wiki/Agile_software_development

[3]. https://www.inflectra.com/ideas/whitepaper/introduction%20to%20agile%20development%20methods.aspx

[4]. http://www.inflectra.com/Agile-Software-Development.aspx

[5]. www.scrum.org

[6]. www.scrumalliance.com

[7]. http://disciplinedagiledelivery.com/

[8]. A Practitioner's Guide to Agile Software Delivery in the Enterprise by Scott W. Ambler and Mark Lines, 2012; http://www.ambysoft.com/books/dad.html

[9]. Going Beyond Scrum - Disciplined Agile Delivery by Scott Ambler, 2013; http://disciplinedagileconsortium.org/Resources/Documents/BeyondScrum.pdf

[10]. http://www.ambysoft.com/unifiedprocess/agileUP.html