Cumhuriyet Üniversitesi Fen Fakültesi
Fen Bilimleri Dergisi (CFD), Cilt:36, No: 3 Özel Sayı (2015)
ISSN: 1300-1949

Cumhuriyet University Faculty of Science
Science Journal (CSJ), Vol. 36, No: 3 Special Issue (2015)
ISSN: 1300-1949

Comparison between Agile and Traditional software development methodologies

Mahdi JAVANMARD[1],Maryam ALIAN[1]

[1]*Tehran Payam Noor University, IRAN*

Received: 01.02.2015; Accepted: 05.05.2015

Abstract. This paper is a review research about the features of the agile and heavy (traditional) software development methodologies and comparisons between them based on: Approach, Success Measurement, Project size, Management Style, Perspective to Change, Culture, Documentation, Emphasis, Cycles, Domain, Upfront Planning, Return on Investment and Team Size. The important difference between agile and heavy methodologies is the adaptability for project changes. In agile methodologies changes do easily but heavy methodologies are static and do not adapt with changes. Also in this paper there are the usage of each groups of methodologies and how can work two groups together. In addition, some types of heavy and agile methodologies have been described.

Keywords: Methodology, Agile, Heavy, Traditional, Plan Driven, Feature Driven.

1. INTRODUCTION

Traditional methodologies are plan driven in which work begins with the elicitation and documentation of a complete set of requirements, followed by architectural and high level design development and inspection. Due to these heavy aspects, this methodology became to be known as heavyweight. Some practitioners found this process centric view to software development frustrating and pose difficulties when change rates are still relatively low. As a result, several consultants have independently developed methodologies and practices to embrace and respond to the inevitable change they were experiencing. These methodologies and practices are based on iterative enhancements, a technique that was introduced in 1975 and that has become known as agile methodologies [1].

The name “agile” came about in 2001, when seventeen process methodologists held a meeting to discuss future trends in software development. They noticed that their methods had many characteristics in common so they decided to name these processes agile, meaning it is both light and sufficient. In consequence to this meeting, the “Agile Alliance” and its manifesto for agile software development emerged [1].

Agile exposes organizational dysfunction. Unlike traditional methods, agile methodologies embrace iterations rather than phases. Agile employ short iterative cycles, small/short releases, simple design, refactoring continuous integration and rely on tacit knowledge within a team as opposed to documentation. Some of the popular agile methods are Extreme Programming, Scrum, Lean, Kanban, Dynamic System Development Method, Feature Driven Development and Adaptive Software Development [2].

The key difference between heavyweight and agile methodologies is the adaptability factor. In an agile methodology if any major change is required, the team doesn't freeze its work process; rather it determines how to better handle changes that occur throughout the project. The verification process in agile method occurs much earlier in the development process. On the other hand heavyweight methods freeze product requirements and disallow change. It

*Corresponding author. Email address: Javanmard@pnu.ac.ir

Special Issue: The Second National Conference on Applied Research in Science and Technology

implements a predictive process and relies on defining and documenting a stable set of requirements at the beginning of a project [2].

2. TRADITIONAL METHODOLOGY

a. The nature of systems development [3]

Software development is a highly complex activity. It is characterized by variable requirements, the need for specialized and diverse skills, changeable and sophisticated technology used to develop and deploy software, and difficulty in management of the people who deal with such complexity every day. It is not uncommon to find organizations overwhelmed by the inherent complexity in implementing systems development projects [3].

Therefore, systems development processes can be described as being complex, unpredictable, and poorly defined. In other words, these processes don't have well-defined inputs and outputs and therefore are considered unrepeatable [3].

b. The traditional systems development model [3]

Almost since its inception, a rational, engineering-based approach, such as the Waterfall method, has been used to develop projects. This choice seems to have been grounded in "hard-systems thinking," which assumes that problems can be well defined, processes can be optimized, and results can be well predicted. Extensive up-front planning is done to measure and control the variations in the development life cycle [3].

The essential nature of the traditional software development life cycle is as follows [3]:

- The goals are to thoroughly understand users' needs, craft a solid design, develop software flawlessly, and implement a functional system that satisfies user needs [3].
- There is a heavy emphasis on thorough planning to deal with risks [3].
- It is based on the principles of hard-systems thinking — identifying alternate ways of reaching the desired state (S1) from the initial state (S0) and choosing the best way to achieve it (S0 – S1) [3].
- Such an approach assumes that problems are well defined and that an optimum solution can be arrived at by extensive, up-front planning [3].
- It also assumes that the processes are predictable and can be optimized and made repeatable [3].
- It is also based on the assumption that processes can be adequately measured and that sources of variations can be identified and controlled during the development life cycle [3].
- In summary, the traditional software development life cycle is highly process-centric [3].

Based on the above understanding of systems development, organizations adopt a management style that is [3]:

Command-and-control-based, with a set hierarchy. Therefore these are predominantly mechanistic organizations geared for high performance in a stable environment [3].

Characterized by high formalization and standardization. People with different specializations are assigned roles for producing defined outcomes. In addition to this, they also produce a significant amount of documentation that explains the software and its technical and design specifications [3].

Notable in that though customers play an important role, their participation is at maximum only during the specification and implementation stages [3].

From the customers' perspective, there are pros and cons to traditional approaches. The definite advantage is its scalability. Very large projects and mission-critical projects need a strong plan and close supervision, and they need provision for stability. The shortcomings are based on the assumption that customer requirements are well understood at the beginning and don't change much, which is rarely the case. Another fundamental problem is that processes take too long, such that when end users see the system, many things - including user requirements - have changed drastically [3].

To summarize, there are two broad themes that emerge in traditional systems development, around which the problems are centered [3]:

The development life cycle and its processes: The way processes are understood and designed and managed (as though they are well defined, predictable, repeatable processes) is highly process-centric. It uses hard-systems thinking, engineering-based in its approach. The focus is on achieving stability [3].

The management style: The way people are organized, managed, and controlled is through "high command and control," a formalized and standardized appropach with limited customer interaction [3].

Summary of Some Traditional methodology as follows:

Plan-Driven Methodologies: Traditional plan-driven models for software design, also known as Software Development Lifecycles (SDLCs) break up projects up into phases. A typical example might be [4]:

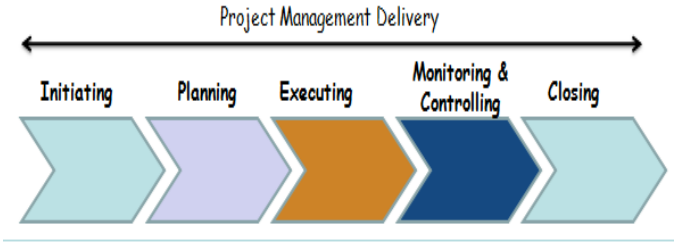

Figure 1. Traditional project management delivery [4].

- **Requirements**: Assessing scope of the system requirements and the overall project [4].
- **Architecture and Design**: Developing an understanding of the solution from a technical perspective, creating a high-level design of modular components and their interactions, and setting standards for how common technical issues should be resolved [4].
- **Development**: Producing code in an environment specific to the culture of the project. Tasks are assigned according to individual skills, and development continues until goals or milestones are reached [4].
- **Testing, Delivery and Feedback**: Testing of individual component should be ongoing, with application-level testing towards the end of the project — ideally, involving customers to confirm that requirements have been met, or to identify changes that must be made [4] .

Plan-driven methodologies work well for small, well-defined projects with a limited scope of work and few variables. Due to the IT industry's rapid growth and progression, methodologies have had to evolve to get expansive projects with multi-million dollar budgets completed on time and on budget [4].

c. Some traditional methodologies include:

1) Waterfall: The waterfall approach emphasizes a structured progression between defined phases. Each phase consists on a definite set of activities and deliverables that must be

accomplished before the following phase can begin. The phases are always named differently but the basic idea is that the first phase tries to capture What the system will do, its system and software requirements, the second phase determines How it will be designed. The third stage is where the developers start writing the code, the fourth phase is the Testing of the system and the final phase is focused on Implementation tasks such as training and heavy documentation. However, in engineering practice, the term waterfall is used as a generic name to all sequential software engineering methodology.[1]

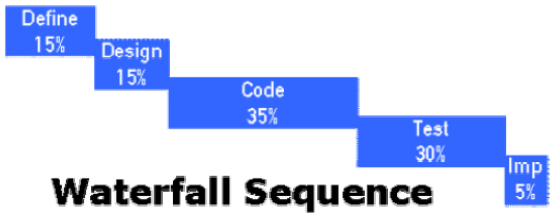

Figure 2. A Comparison between Agile and Traditional Software Development Methodologies M. A. Awad [1]

2) Rational Unified Process (RUP): All efforts, including modeling, is organized into workflows in the Unified Process (UP) and is performed in an iterative and incremental manner. The lifecycle of the UP is presented in Figure 3. Some of the key features of the UP are as follows [1]:

- It uses a component based architecture which creates a system that is easily extensible, promotes software reuse and intuitively understandable. The component commonly being used to coordinate object oriented programming projects [1].
- Uses visually modeling software such as UML – which represent its code as a diagrammatic notation to allow less technically competent individuals who may have a better understanding of the problem to have a greater input [1].
- Manage requirements using use-cases and scenarios have been found to be very effective at both capturing functional requirements and help in keeping sight of the anticipated behaviors of the system [1].
- Design is iterative and incremental – this helps reduce project risk profile, allows greater customer feedback and help developers stay focused.
- Verifying software quality is very important in a software project. UP assists in planning quality control and assessment built into the entire process involving all member of the team [1].

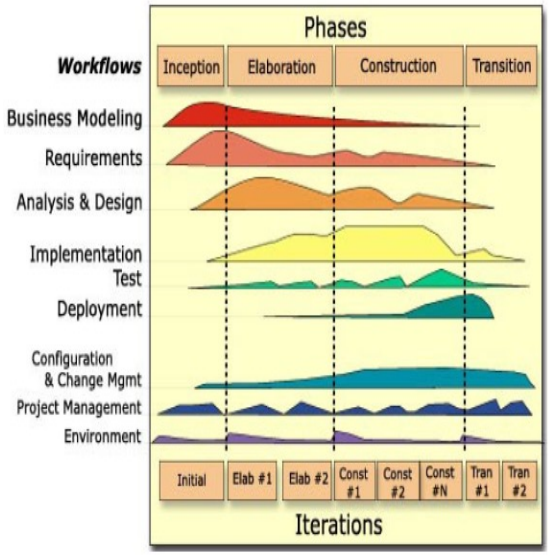

Figure 3. Development Methodologies M. A. Awad [1].

3) Rapid Application Development (RAD): Emphasizes user involvement and the building of prototypes to prove techniques and refine requirements. After the scope of the project is defined in the Requirements Planning phase, users interact with system analysts to develop prototypes in the User Design phase. Users then provide direct feedback during the Construction phase. Finally, in the Cutover phase the end users are trained to use the product as it is deployed.[1].

4) Spiral Model: Another heavyweight software development model is the spiral model, which combines elements of both design and prototyping-in-stages, in an effort to combine advantages of top-down and bottom-up concepts. The spiral model was defined by Barry Boehm, based on experience with various refinements of the waterfall model as applied to large software projects [1].

- Objective setting – Specific objectives for the project phase are identified [1].
- Risk assessment and reduction – Key risks are identified, analyzed and information is obtained to reduce these risks [1].
- Development and Validation – An appropriate model is chosen for the next phase of development.
- Planning – The project is reviewed and plans are drawn up for the next round of spiral [1].

5) ITIL: Information Technology Infrastructure Library (ITIL): A comprehensive set of processes and best practices for projects in the IT realm. ITIL is a how-to guide for IT projects such as data center design or large-scale institutional network hardware configuration.[4]

3. AGILE METHODES [3]

To address the challenges posed by the traditional methods, a set of lightweight methods called "Agile" were developed more than a decade ago. They include XP, Scrum, feature-driven development, and more. Agile methods, using various tactics, try to overcome the limitations of the dynamic nature of systems development projects [3].

Agile methods address the inherent problems of traditional systems development using two scientific concepts: One method is using "empirical process control." One can also say that this is based on "soft-systems thinking." The other method is seeing the systems development as "complex adaptive systems" and designing team structures and methods around that [3].

Here is how Agile addresses the shortcomings of the traditional software development life cycle [3]:

Agile methods adopt the empirical process control model as against the defined process control model. Empirical process control is meant for processes that aren't well defined and are unpredictable and unrepeatable. It implements control through frequent inspection and adaptation. Since software development processes are highly complex and variable in nature, empirical process control methods seem to be a best fit to deliver results [3].

According to complexity theory, a complex adaptive system (CAS) self-organizes and adapts to changes in the environment without any central rules governing its behavior. An Agile development system can be likened to a CAS, responding to complex and unpredictable changes in the requirements [3].

The development model changes from a linear life cycle model to an evolutionary delivery model. This is characterized by short iterative cycles, with periodic reflections and adaptations and continuous integration of code into the overall system under development [3].

Since the cycles are short and iterative, they provide the necessary flexibility and speed to adapt to changes in requirements through constant feedback from the stakeholders [3].

Agile methods also need constant collaboration with customers, using their input and feedback at various checkpoints during each iterative cycle [3].

a. Summary of Some Agile methodology are shown as follows:

1) **Scrum**: A generic process framework, not limited to software development. Cyclic and iterative, rather than phase-oriented - planning and implementation are concurrent, so while the team is busy building, they're also planning for the future [5].

Scrum is an iterative, incremental process for developing any product or managing any work. Scrum concentrates on how the team members should function in order to produce the system flexibility in a constantly changing environment. At the end of every iteration it produces a potential set of functionality. The term 'scrum' originated from a strategy in the game of rugby where it denotes "getting an out-of-play ball back into the game" with team work [1].

Scrum does not require or provide any specific software development methods/practices to be used. Instead, it requires certain management practices and tools in different phases of Scrum to avoid the chaos by unpredictability and complexity [1].

Key Scrum practices are discussed below and the Scrum process is shown as follows [1]:

- Product Backlog - This is the prioritized list of all features and changes that have yet to be made to the system desired by multiple actors, such as customers, marketing and sales and project team. The Product Owner is responsible for maintaining the Product Backlog [1].
- Sprints - Sprints are 30-days in length, it is the procedure of adapting to the changing environmental variables (requirements, time, resources, knowledge, technology etc) and must result in a potentially shippable increment of software. The working tools of the team are Sprint Planning Meetings, Sprint Backlog and Daily Scrum meetings [1].
- Sprint Planning meeting – Sprint planning meeting is first attended by the customers, users, management, Product owner and Scrum Team where a set of goals and functionality are decided on. Next the Scrum Master and the Scrum Team focus on how the product is implemented during the Sprint [1].
- Sprint Backlog – It is the list of features that is currently assigned to a particular Sprint. When all the features are completed a new iteration of the system is delivered [1].
- Daily Scrum – It is a daily meeting for approximately 15 minutes, which are organized to keep track of the progress of the Scrum Team and address any obstacles faced by the team [1].

2) **Extreme Programming (XP)**: XP focuses on tactical best-practices for building software rather than the best ways to get the overall project to the release on time and on budget. It prescribes a very specific set of software development practices, like pair programming, test-driven development, and continuous integration. As a result, agile software projects often use Scrum for project management while drawing tactical practices from XP [4].

3) **Feature Driven Development (FDD)**: Feature Driven Development (FDD) was used for the first time in the development of a large and complex banking application project in the late 90's. Unlike the other methodologies, the FDD approach does not cover the entire software development process but rather focuses on the design and building phases [1].

4) **Dynamic System Development Method**: The DSDM, Dynamic System Development Method, was developed in the United Kingdom in the mid-1990. The fundamental idea behind DSDM is to fix time and resources, and then adjust the amount of functionality accordingly rather than fixing the amount of functionality in a product, and then adjusting time and resources to reach that functionality. DSDM consists of five phases [1].

b. General Features and Comparison of Agile Methodologies [1]

Table 1 is shown the comparison of Agile Methodologies:

Method Name	Key Points	Special features	Identified weakness
ASD	Adaptive culture, collaboration, mission-driven component based iterative development	Organizations are seen as adaptive systems. Creating an emergent order out of a web of interconnected individuals	ASD is more about concepts and culture than the software practice
DSDM	Application of controls to RAD, use of timeboxing and empowered DSDM teams.	First truly agile software development method, use of prototyping, several user roles : "ambassador", "visionary" and "advisor"	While the method is available, only consortium members have access to white papers dealing with the actual use of the method
XP	Customer driven development, small teams, daily builds	Refactoring - the ongoing redesign of the system to improve its performance and responsiveness too change	While individual practices are suitable for many situations, overall view & management practices are given less attention
SCRUM	Independent, small, self-organizing development teams, 30-day release cycles.	Enforce a paradigm shift from the "defined and repeatable" to the "new product development view of Scrum"	While Scrum details in specific how to manage the 30-day release cycle, the integration and acceptance tests are not detailed
FDD	Five-step process, object-oriented component (i.e. feature) based development.	Method simplicity, design and implement the system by features, object modeling	FDD focuses only on design and implementation. Needs other supporting approaches.

Figure 4. Table of General Features and Comparison of Agile Methodologies [1]

4. DIFFERENCE IN AGILE AND HEAVYWEIGHT METHODOLOGY

A summary of the difference of agile and heavyweight methodologies is shown in table below [1].the difference is useful for comparing them.

Table 1. Difference in Agile and Heavyweight Methodologies [1].

	Agile Methods	Heavy Methods
Approach	Adaptive	Predictive
Success Measurement	Business Value	Conformation to plan
Project size	Small	Large
Management Style	Decentralized	Autocratic
Perspective to Change	Change Adaptability	Change Sustainability
Culture	Leadership-Collaboration	Command-Control
Documentation	Low	Heavy
Emphasis	People-Oriented	Process-Oriented
Cycles	Numerous	Limited
Domain	Unpredictable/Exploratory	Predictable
Upfront Planning	Minimal	Comprehensive
Return on Investment	Early in Project	End of Project
Team Size	Small/Creative	Large

5. HOW CAN WATERFALL AND AGILE WORK TOGETHER?[5]

While it's tempting for proponents of agile methodologies to claim they work best for every development project, that's simply not the case.While agile project management methodologies can generally be used for any development project and will often provide some powerful benefits, situations definitely arise when more traditional methods like Waterfall are the smarter way to go [5].

For example, large, enterprise-wide development efforts in which the user is being led through a standardized process are completed more efficiently using Waterfall methods. On the other hand, teams developing mobile applications – which must be highly flexible and quickly updated due to the nature of the ecosystem they're created for – will likely find agile methods more conducive to success [5].

However, in the real world of project management and development, many projects are not completely black or white. They actually benefit most from a hybrid approach that takes advantage of the strengths of both Agile and Waterfall methodologies without allowing them to get in each other's way [5].

a. Blending the Best of Both Worlds[5]

How can Waterfall and Agile work together to the benefit of the project [5]?First of all, it's important to understand where each method's strength lies[5].

Agile methods generally allow for faster iteration and more frequent releases with subsequent user feedback that can be worked into future development. Waterfall methods tend to greatly lessen the number and severity of errors that will affect the end user [5].

So, in many cases, the optimum project combination incorporates significant planning and QA input early in the development process to mitigate errors while introducing Agile processes

to the release schedule and user feedback opportunities, allowing for faster and more controlled improvements [5].

b. Pros and Cons[5]

Of course, blending these two methods into a hybrid project requires some level of compromise from both sides [5].

In contrast with a strictly Waterfall project, a hybrid project has to give up some level of certainty in exchange for the flexibility afforded by the Agile aspects of the development process. Similarly, in contrast with an agile project, a team working on a hybrid project may find their freedom limited by Waterfall's planning, budgeting and scheduling constraints [5].

Through adequate communication and effective cooperation between team members and diverse teams, however, the hybrid approach can often be the most effective means of completing complex projects with shifting requirements [5].

6. CONCLUSION

Although there are many benefits of using agile methodologies, but these methodologies cannot be fully used in all projects. Agile methodologies are adaptive with changes but they are used in Small Projects. Today Some Project do based on heavy methodologies yet, choosing a methodology is depended on type, scale of project and another factor.

REFERENCE

[1] M. A. Awad,"A Comparison between Agile and Traditional Software Development Methodologies", School of Computer Science and software Engineering, The University of Western Australia , 2005.
[2] Bhattacharjee, Vishwajyoti , July 06, 2012, http://EzineArticles.com/7162906.
[3] Vijaya Devi," Traditional and Agile Methods: An Interpretation", 23 January 2013, http://www.scrumalliance.org/community/articles/2013/january/traditional-and-agile-methods-an-interpretation.

[4] "Project Management & Agile Methodologies",September 17, 2012,https://www.cprime.com/2012/09/project-management-agile-methodologies.
[5] "Hybrid Projects: How Can Waterfall and Agile Work Together", May 13, 2014,https://www.cprime.com/2014/05/hybrid-projects-how-can-waterfall-and-agile-work-together.