El Adel Taoufik
IT 363
CRN 18093

Agile and waterfall

When it comes to larger scale systems, developing effective and working software has indeed been a challenge. Surprisingly, and according to the 2015 CHAOS report, more than 52% of all size projects either fail or are challenged (Standish Group 7). The causes of the failures are organizational and not technical. Taking adequate approaches and using the most effective methods while developing software are keywords for success. In this paper, I will conduct a comparison between agile and waterfall, by describing the principles of each methodology, stating their advantages and disadvantages, and propose few suggestions to address their challenges.

Regardless of what methodology is used to develop a system, there is a systematic cycle that the development process must take. This cycle is referred to as the systems development life cycle (SDLC) and is defined as the process of determining how an information system (IS) can support business needs, designing the system, building it, and delivering it to users (Dennis, et al. 5). The SDLC adheres to fundamental phases that are essential for developers, such as planning, analysis, design, and implementation. These phases proceed in a logical path, and each one

of them consists of a series of steps. Different projects may give more importance to particular parts of the SDLC or approach them in different ways. However, all system developments have elements of those four phases.

Waterfall is a traditional SDLC model. According to Wikipedia, “the traditional methodology or waterfall is a sequential development approach, in which development is seen as flowing steadily downwards through several phases”. It was proposed by Winston Royce in 1970 and it emphasizes a structured progression that is linear between defined phases. Each phase relies on the complacency of the previous phase and has to be documented with deliverables. The first phase is the requirement analysis where the system and software requirements are gathered through interviews, group workshops, or questionnaires. At this phase, the purpose, the users, and the environment of the system are all defined. The second phase is the design stage that establishes how the system will operate in terms of infrastructure; the user interface; the specific programs, databases, and files. Developing architectural and strategical designs determine the feasibility of the project. Next phase is the implementation, also known as the coding phase of software development. At this stage, ideas are converted into tangible source code using programming languages and tools. The designed models are tested to ensure the correct output is received. The fourth phase is the System Integration and Testing. At this stage, heavy and iterative testing gets performed. Each application is

tested and integrated with all other modules with different functionality before the entire system is tested. The final phase is the system deployment and maintenance. At this stage, the software is released at the client's end, then routine maintenance is carried out.

The main two advantages of the waterfall approach are that the phases are completed one at a time and are controlled with a timeline. Tasks are easier to manage due to the rigidity of the model. Minimal management is required because the end goal and deliverables get established and documented at the beginning. It works well for smaller projects where requirements are clearly understood. Waterfall model is most appropriate for short projects that have no ambiguous requirements and are well documented, clear, and fixed. It also helps when the technology is understood and is not dynamic.

The fact that any tangible output isn't produced until late in the waterfall cycle, causes a lack of flexibility in the model. Once a phase has completed, it gets extremely difficult, time and money wise, to accommodate unexpected changes or revisions. The lack of client/developers' interaction, and the emphasis on documentation that consumes a great deal of time from developers all manifest as disadvantages to the model. These drawbacks can hinder the success of projects and particularly the bigger and more complex ones.

The issues discussed above could be addressed by focusing more on working software rather than following plans or wasting too much time on documentations. More interactions with the customer before moving into the implementation phase would provide a clearer and updated picture of the requirements.

During the 80's, and as software was used more as a solution to more complex projects, a need for more flexible software development processes was arising. The low rate of project success and the lengthy time it took to produce working software using traditional methods have set up a ground to introduce new iterative models known as agile methods. Agile is both a philosophy and an umbrella term for a set of frameworks and approaches that share certain common characteristics. Lapham has defined agile as "An iterative and incremental (evolutionary) approach to software development which is per- formed in a highly collaborative manner by self-organizing teams within an effective governance framework with "just enough" ceremony that produces high quality software in a cost effective and timely manner which meets the changing needs of its stakeholders"(16).

Agile models include Scrum, Extreme Programming (XP), Feature-Driven Development (FDD), Dynamic System Development, Adaptive Software Development (ASD) and more. All these models share the same core practices and principles that value individuals and interactions over processes and tools, working

software over excessive documentation, welcoming change over sticking to scripts, customer collaboration over contract negotiation.

In 2001, a group of 17 software developers and methodologists met in Utah and developed what is known as The Agile manifesto. This declaration sums up the core values of agile in twelve principles. 1 Customer satisfaction by early and continuous delivery of useful software. Unlike the waterfall model, a prototype of the working software is continuously delivered to the customer for testing and feedback to not only ensure the fulfilment of the established requirement but to also accommodate any unexpected changes. 2 Frequent delivery (weeks rather than months). Delivering working features provides immediate value to the customer which contributes to the success of the project. 3 Close, daily cooperation between business representants and developers. This involves keeping minimal requirements and documentation and focusing more on daily interactions between developers and business people to keep all the members of the project on the same page. 4 Face-to-face conversation is the best form of communication. The purpose behind this principle is to unleash the power of face-to-face communication. Practices such as daily stand-up meeting, and co-located teams help overcome obstacles and improve productivity. 5 Welcome changing requirements, even late in development. The responsiveness to change allow the costumer to keep their competitive advantage in face of emerging threats or

opportunities. 6 Projects are built around motivated individuals, who should be trusted. The project is more likely to succeed if the developers are motivated and are provided with a productive environment with minimal micromanaging where everyone's leadership is a contributing factor to the success of the project. 7 Self-organized team. An agile team is not only responsible for writing code, they self-organize all aspects of software development, from meeting with the client to gather requirements to organizing their work. Agile methods empower people to allow them to produce at their best. 8 Sustainable development. Creativity and dedication require the team members to maintain a healthy work-life balance by setting a ceiling of 40 hours of work a week to avoid burnout or exhaustion. This enables developers to maintain a constant pace indefinitely. 9 Continuous attention to technical excellence and good design. The right skills and good design ensure the team can maintain the pace, constantly improve the product, and sustain change. 10 Simplicity. To develop just enough to get the job done at the time being. 11 Continuous improvement. This is achieved through retrospective meetings where progress is shared, impediments are discussed, and behavioral adjustments are set in order to improve the process. 12 Working software is the primary measure of progress. This principle has remedied some deficiencies of the waterfall model where the goal has deviated bureaucratically from focusing on working software to documentation and protocols. Just enough documentation, but essentially working

software is what creates value for the customer ("12 Agile Manifesto Principles Simply Explained").

As stated through the core principals of agile, the main advantage of this method is the adaptiveness to recurring changes. It is worth noting that the cost of addressing change has decreased drastically. Customer satisfaction is much higher and so is the project success rate.

Agile methods submerge few challenges that can surface as disadvantages. The minimization of documentation can hinder new team members to get up to speed. Agile demands more time and energy from everyone because developers must interact constantly with each other and with customers. The practice of daily meetings or the interaction with the client depend on the availability of each party which sometimes is just unfeasible especially when the members are not located in the same physical space. The unpredictability aspect does not allow a clear figure of the cost of the project as a whole.

These issues can be addressed by taking advantage of new innovations and techniques. For instance, work management software can be a great asset so that anyone can have a visible access to the set goals versus progress, and for developers to address the challenge of documentation. Videoconferencing, webcams, and collaboration tools can be used as alternatives when circumstances hinder face-to-face communication. On another note, teams that consists of more

than 10 members are not as efficient and will be affected by the law of diminishing return. Therefore, forming teams between 3 to 9 persons would ensure efficiency of communication and best productivity.

To conclude, many methodologists suggest that organizations should adopt either agile or waterfall depending on the size of the project, the resources on hand, the level of expertise of the developers, and how clear are the requirements. If the requirements have potential for change which is often the case nowadays, then agile is the way to go. If the client has a documentation intensive policy and must have the best estimate of the costs before starting the project, then waterfall should be considered. However, and according to statistical evidence, agile has proven to be more efficient and more successful than waterfall; hence it is fair to say that agile is indeed a higher and more powerful method than the outdated waterfall.

References

- The Standish Group, www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf
- Dennis, Alan, et al. Systems Analysis and Design. 5th ed., John Wiley & Sons, 2012.

- Lapham, Mary Ann., Williams, Ray., Hammons, Charles (Bud)., Burton, Daniel., & Schenker, Alfred. 2010. Considerations for Using Agile in DoD Acquisition (Technical Report CMU/SEI-2010-TN-002). Pittsburgh: Software Engineering Institute, Carnegie Mellon University. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=9273
- "The 12 Agile Manifesto Principles Simply Explained." LinkedIn, www.linkedin.com/pulse/12-agile-manifesto-principles-simply-explained-jacob-aliet-ondiek.