In recent years many developers have championed the role of agile methods over waterfall methods in software development. Are they right? Form a clear and concise argument, and support your claims with evidence from literature.

Karthikeya Udupa K M (1393456)

January 11, 2014

Abstract

The quality of a software system is a clear reflection of the standards and the methodology followed during its lifecycle. With increasing use of software in our daily lives, making sure that the end-product solves the intended problem and works flawlessly has become more important then ever. Various development methodologies followed are being constantly analysed and researched into to optimise the overall development process, leading to a more reliable and efficient end product. The traditional *Waterfall model* has been in use for many decades now, but with increasing complexity of the software along with rapidly changing needs of the modern day, alternative development strategies are being looked into, one of which is *Agile methodology*. By dividing the project into much smaller development sprints, the methodology plans to providing the end user business value from the product at a much earlier stage of development along with the flexibly to accommodate any modifications in the requirement even after the development has commenced. The article not only tries to analyse the merits and demerits of both the systems in comparison with each other but also also tries to evaluate whether or not the moving towards agile is beneficial for an organisation and is the trend that is being presented about widespread adaptation entirely true.

The nature of software and the way its developed has evolved over the two decades, successful functioning of various critical systems in several industries is entirely governed by the software which is managing it. Because of this, software

development itself has become an integral part of organisations leading them to invest both time and money in improving the development strategies. In Software engineering, the focus has always been on creation and implementation of methodologies that guide effective software development (Davis, Bersoff, and Comer 1988). Finding techniques which can be followed repeatedly with predictable results on software projects of different magnitudes belonging to different domains is essential in making the process of software development as rewarding as possible, ultimately leading to timely delivery of projects which fulfils the needs of their end users completely with adherence to the provided time and monetary constraints.

The severe dependency on the features and services offered by these softwares in critical systems firmly assert the need for them to be very reliable. Apart from critical systems, software solutions also have a very pervasive yet pivotal role in various aspects of our day to day lives, hence their recent failure in highly critical situations such as airplane systems, satellite launches (Dowson 1997) and many other scenarios is indeed very concerning, in fact a recent research conducted by Standish Group (The Standish Group Report 1995) shows that 15% of the software being developed is never delivered and a mere 16.2% of the projected are completed on time and on budget. All this indicates that there is a drastic need to look into improvement of the various ways in which software can be made more effective and reliable. The quality and standards followed during the development process is one such area which has been indicated by several researchers as a critical aspect in which improvement can be made as there is a direct correlation between the quality of the developed software and the development process involved (Fuggetta 2000). The various studies and researches related to the methodological development of software is is known as software development process (Fuggetta 2000) which is formally defined as:

"A software process can be defined as the coherent set of policies, organisational structures, technologies, procedures, and artefacts that are needed to conceive, develop, deploy, and maintain a software product."

The foremost goals in the process is to identify a optimal software lifecycle model, which is a formal classification of the tasks involved throughout the lifetime of software into various stages. There are several activities that are involved in development of a software systems (Sommerville 1996), each organisation has its software process but usually they follow a tailored version of one of many generic generic model.

With the emergence of requirement for developing of complex and large scale software solution, the earlier software development process which consisting of just two phases, an analysis and then a coding phase (Royce 1970), proved to be a failure and it was concluded that several additional stages were required to create an effective development methodology. The initial life-cycle model was developed

after failure of various high-level projects due to the unsystematic approaches adopted by the developers, the model is now known as the *waterfall model or the traditional model* (Royce 1970). There were several further improvements made in the model. This model consisted of seven sequential and linear stages of various processes. The model commenced with specification phase where the functionality and operating constraints of the system and the software are specified and analysed in great depth, based on the information gathered the overall structure of the software is defined and designed along with identification of critical individual components. This is followed by the development which is done in a particular programming language where the components are distributed among several development teams. Once the development is complete, each module is then integrated together and tested as a complete system. Once it is approved, the created solution is then delivered to the customer followed by an ongoing process to alteration and maintenance of the software solution based on the user's needs (Sommerville 1996).

Although this model proved to be a considerable improvement and fairly systematic over the previously followed development strategy, the waterfall model however is considered flawed. The most evident one being the lack of any kind of feedback between the various stages. The actual end-user's or the business requirement which the software is supposed to meet is often discovered at a much later stage of the process and same is the case with the various implementation issues that might have risen during development. However, with the specifications being frozen at an early stage, leaves a little room for alteration of the software based on the newly acquired information from the customer who were not involved in the development phase initially (Laplante and Neill 2004). This contributes to the perpetuating failure with the delivering of the software that is actually required by the end user, and the inability to bridge the gap between the end users that are going to use the product in real world and the system analysts who were responsible for understanding the actual needs (McCracken and Jackson 1982). It has also been observed that due to the considerable amount of the monetary investment of the overall process being invested in the requirement and designing activities, a change coming to light at the later phases can result in being very expensive (Ji and Sedano 2011).

(Larman 2004) indicates that waterfall produced unreliable schedules and estimates which can be credited to the reason that this methodology can lead to avoidance of complex and high-risk components till the end of the estimated time and also does not take into account the various complication that is involved in integration process, which is encouraged to be done at a very later stage of development.

In the recent years, a much more practical approach has emerged and is being

adopted by developers quite rapidly known as *agile development*. The methodology has several possible approaches (scrum, extreme programming, agile modelling etc) but primarily focuses on fast development and pays attention to possessing alertness to changes occurring in the requirement. This is achieved by breaking the project into smaller sub-projects and conducting development over short time periods with subsequent releases (Ruparelia 2010). This allows the end-user to experience the software and provide feedback and help the developers capture small incremental changes and hence requirement repositioning can be done accordingly. This is also follows a prime directive for the agile development methodology, to provide business value to the customer at an earlier stage and more regularly then it can be provided by traditional methods where the business value is given once the development process is complete (Racheva, Daneva, Sikkel, Wieringa, and Herrmann 2010). Another advantage that the agile methodology provides is the creation of learning opportunity at various different timescales, from daily stand ups to sprint review meetings, allows the team to decide the future work plan for the project and prioritising tasks for the upcoming iteration (Cohn 2005).

Agile process certainly presents a very viable alternative when compared to traditional methodology as the present pace of requirement and change in the market and technology (MacCormack 2003), several experimental project implementation have even indicated that a comparison of both the methodology implementing the same project resulted in a 50% reduction of the required time (Cutter 2002). However they also have their shortfalls. Firstly, there is a requirement for continuous testing, which means additional efforts would have to be put into maintaining and management of integrated test environments for different platforms (The Standish Group Report 1995). Due to lack of focus in architectural design the process might lead to bad design decisions (Larman 2004). There has also been considerable concern about the issue of scalability of agile development (Petersen and Wohlin 2009; Sommerville 1996).

Even though it is widely assumed that agile is replacing the waterfall on a large scale, it is not entirely true. A recent survey conducted among 200 process practitioner who worked on thousands of projects over the duration of five years indicated that waterfall is still very predominant and used in at least a third of the projects (Laplante and Neill 2004). This can also be attributed to the lack of understanding or even the convince in following easier but improper technique which may result in failure (Laplante and Neill 2004). For companies that already follow waterfall, switching to agile would not be a simple process. The team would have to unlearn traditional methodology and then approach towards agile processes, given time the whole system can be adapted gradually (Sureshchandra and Shrinivasavadhani 2008).

Over the years, there has been several researches and experiments conducted

to analyse various software development strategies and methodologies that would improve the overall quality and reliability of software. The wide variation in the techniques being used and the ever changing trends of the consumer's requirement arises the question that can one methodology ever meet the need of software development of products of all and any nature and sizes? Probably, not. The systems and their requirements are bound to change with time, and certain needs can only be identified once the implementation has been initiated. The perspective where in software development is viewed as a process helps in identifying the challenges that must be met in order to establish proper practices in place to provide effective results. But the overall process is not restricted to the lifecycle, the various other aspect that govern the development such as organisational factors, technological limitations and economic restrictions are vital and must be paid close attention to. An ad-hoc approach would never be efficient in dealing with the problem effectively. No methodology is considered perfect or complete to cover various aspects of development in project of different scales. Each methodology has it positive and negative aspects and each problem has its own set of requirement and approach, so the organisation has to be very considerate in selecting the most optimal development methodology for their software keeping in mind their requirements and constraints.

References

(1995). The standish group report. The Standish Group Report.

Cohn, M. (2005). *Agile estimating and planning*. Pearson Education.

Cutter, J. H. (2002, October). What is agile software development? In *CROSSTALK The Journal of Defense Software Engineering*, pp. 4–9.

Davis, A. M., E. H. Bersoff, and E. R. Comer (1988). A strategy for comparing alternative software development life cycle models. *Software Engineering, IEEE Transactions on 14*(10), 1453–1461.

Dowson, M. (1997). The ariane 5 software failure. *ACM SIGSOFT Software Engineering Notes 22*(2), 84.

Fuggetta, A. (2000). Software process: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, pp. 25–34. ACM.

Ji, F. and T. Sedano (2011). Comparing extreme programming and waterfall project results. In *Software Engineering Education and Training (CSEE&T), 2011 24th IEEE-CS Conference on*, pp. 482–486. IEEE.

Laplante, P. A. and C. J. Neill (2004). The demise of the waterfall model is imminent. *Queue 1*(10), 10.

Larman, C. (2004). *Agile and iterative development: a manager's guide*. Addison-Wesley Professional.

MacCormack, A. (2003). Agile software development: Evidence from the field. In *Agile Development Conference*.

McCracken, D. D. and M. A. Jackson (1982). Life cycle concept considered harmful. *ACM SIGSOFT Software Engineering Notes 7*(2), 29–32.

Petersen, K. and C. Wohlin (2009). A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case. *Journal of Systems and Software 82*(9), 1479–1490.

Racheva, Z., M. Daneva, K. Sikkel, R. Wieringa, and A. Herrmann (2010). Do we know enough about requirements prioritization in agile projects: insights from a case study. In *Requirements Engineering Conference (RE), 2010 18th IEEE International*, pp. 147–156. IEEE.

Royce, W. W. (1970). Managing the development of large software systems. In *proceedings of IEEE WESCON*, Volume 26. Los Angeles.

Ruparelia, N. B. (2010). Software development lifecycle models. *ACM SIGSOFT Software Engineering Notes 35*(3), 8–13.

Sommerville, I. (1996). Software process models. *ACM Computing Surveys (CSUR) 28*(1), 269–271.

Sureshchandra, K. and J. Shrinivasavadhani (2008). Moving from waterfall to agile. In *Agile, 2008. AGILE'08. Conference*, pp. 97–101.