SCHOOL OF COMPUTER SCIENCE AND STATISTICS

How do software development teams view agile transformation?

M.Sc. Management of Information Systems

by Andrzej Holubowicz

University of Dublin, Trinity College

Supervisor: Dr. Frank Bannister

Submitted to the University of Dublin, Trinity College, September 2015

Table of Contents

List of figures

List of tables

Declaration

I declare that the work described in this dissertation is, except where otherwise stated, entirely my own work, and has not been submitted as an exercise for a degree at this or any other university. I further declare that this research has been carried out in full compliance with the ethical research requirements of the School of Computer Science and Statistics.

Permission to lend and/or copy

I agree that the School of Computer Science and Statistics, Trinity College may lend or copy this dissertation upon request.

Acknowledgements

I would like to thank my supervisor Dr. Frank Bannister for all his guidance, support and patience over the months that it took to do this research.

I am indebted to the eleven anonymous individuals who agreed to be interviewed for this research. This dissertation would not have been possible without their help, and I am eternally grateful.

Last, but not the least, I would like to thank my beloved wife. I would not be here without your love and your support. Thank you.

Abstract

The purpose of this research was to explore the issues and challenges that might arise in agile software development processes during transition from traditional development methods to agile. It also provides insights for management to help organizations avoid and overcome barriers in adopting Agile. A qualitative research method design was used to capture the knowledge of software development practitioners in its natural settings.

Data was collected through semistructured face-to-face interviews. In the first round of data analysis, a simplified open coding technique was used to identify possible concepts, along with their properties and dimensions. The simplified open coding technique is a form of content analysis where the data is read and categorized into concepts. In the second round, the codes were reviewed, and the concepts were organized by recurring themes. These themes were used later as a basis for creating a set of stable and common categories.

The research presented 11 main findings, which are:
Finding 1: How software development teams understand waterfall & agile?
Finding 2: Waterfall could be suitable where failure will be catastrophic
Finding 3: Upfront design does not reduce errors or inflexibility
Finding 4: The ability to adapt to change is a key to competitive advantage
Finding 5: Agile leads to greater customer satisfaction
Finding 6: The day of monolithic documentation is over
Finding 7: Technical teams adopt agile faster than management, but before management buys-in transformation cannot be completed
Finding 8: Death march to delivery / burnout issue has to be dealt with
Finding 9: The QA approach in agile is still not perfect
Finding 10: Managing vendors is easier in waterfall
Finding 11: Coaching and training is essential for successful transformation

I. INTRODUCTION

In this chapter, an introduction to the background of the study will be provided. The circumstances under which the opportunity to study the problems will be explained. The chapter will then discuss the broad nature of the problem and establish the benefits of undertaking this study to understand this problem better.

Research Background

In the 1990s, close to 80% of all software development projects ran over budget, were behind schedule or did not satisfy the requirements specified (Guntamukkala et al. 2006). Many of these projects required a large amount of time to completion, in many cases, making the original requirements invalid or outdated. Coping with changing requirements and business environments was a major problem for the software development industry. Since the early-mid 2000's, numerous modern methodologies such as eXtreme Programming (XP) (Beck & Andres 2004) and Scrum (Schwaber & Beedle 2002) have come to the fore within the software industry to help software development efforts to cope with change, become leaner in their processes and perform better.

A subsequent report in 2006 which followed up on the previous report in 1994 showed that close to 35% of the projects were categorised as successful in 2006 (Rubinstein 2007). Also, only 19% were considered outright failures in 2006, compared with 31% in 1994.As one can imagine, having a 20% 'outright failure' rate may be considered high. However, having a 35% success rate would be considered abysmal for most business scenarios.

Traditional development approaches were around for decades, they were very structured and rigid with emphasis on getting it right the first time. This kind of approach tends to fail in the current rapidly changing market environment and calls for changes in the approach to development. The methodology change poses a series of problems for organisations as they have to adapt to radically different ways of conducting their business. It is human nature to resist the change and multiple issues arise from the transformation point of view.

This research takes a close look at the processes and individual viewpoints on the changing environment in order to inform managers of the arising problems and give them the ability to either avoid or remedy them.

Scope of the study

The success or failure of agile transformation in the long run depends on the position of those who will be participating in the process. Software development teams can either embrace or reject systematic changes and it is unlikely that a transformation can be successful without support from the development teams themselves.

This study investigates attitudes towards adopting agile, to probe this face to face interviews were conducted with eleven individuals working in a variety of roles within the software development industry. The participants were selected based on their experience and roles within software projects in an attempt to get good representations from various functional groups.

Research Beneficiaries

This research will be of interest to several different groups. Firstly managers in software development organisations who are either planning or undergoing agile transformation, as it aims to look into processes that are taking place from the “boots on the ground” perspective. Software development managers should be able to use the research to advise their vision of how agile fits into their requirements, aligning the IT function more closely with the overall business goals. Agile practitioners may find the research useful in understanding how IT can play both a supportive and transformative role in the agile transformation.

Research questions / objectives

The primary research question posed in this study is:

- How do software development teams view agile transformations?

This question is broad and gives rise to a number of secondary questions, as follows:

- Are there factors that are slowing down the adoption of agile?
- What are the barriers for agile adoption?
- Are there aspects of agile that software development teams are unhappy about?
- Do software development team members understand agile / scrum topics?

- How to convince people to adopt agile?
- What are the implications for managers?

Dissertation roadmap

Chapter I introduces this research, and discusses the research question and subquestions.
Chapter II reviews the relevant literature in the field of software development, covering academic and selected commercial information sources.
Chapter III discusses the approaches taken for this research and the reasoning behind them.
Chapter IV contains an analysis of the data gathered, along with selected quotations from research participants.
Chapter V contains conclusions as well as some proposals for further research.

II. LITERATURE REVIEW

In this chapter, the existing literature in the field of software development processes will be reviewed. Particular attention is given to the software development methodologies being used in this case study, i.e. the waterfall model and the agile development model. Once the theoretical context of these methodologies is examined, an examination of the transformation studies is carried out. A number of areas are covered in this chapter:

- What is software development?
- Characteristics of waterfall model
- Characteristics of agile / scrum
- Previous studies in the area of agile transformations

Software development

A software development process defines how software development organizations manage delivery of software products, this includes but is not limited to writing and maintaining the source code. Development methodology is “a recommended collection of phases, procedures, rules, techniques, tools documentation, management and training used to develop a system” (Avison & Fitzgerald 2006). In a broader sense software development can be extended to include elements of research, new development, reuse, redevelopment and any other activities that result in software product (Associates 2012). There are two primary approaches to managing this process, the waterfall, known as the traditional model and more recent agile approach (Morris 2001). Most approaches to the software development can be generalised to include the mixture of the following development stages:

- Problem definition
- Research
- Requirements gathering
- Planning / developing solutions
- Implementation (writing code)
- Testing
- Release
- Maintenance & bug fixing

Software development methodologies

Software Development Methodology (SDM) did not emerge until 1960 (Elliott 2004). Software development lifecycle (SDLC) could be considered one of the oldest formalised method of it systems development. The main idea behind SDLC was:

"to pursue the development of information systems in a very deliberate, structured and methodical way, requiring each stage of the life cycle from inception of the idea to delivery of the final system, to be carried out rigidly and sequentially" (Elliott 2004, p.87)

Software development methodologies require extensive planning, strict process and rigorous reuse (Boehm & Turner 2003). This paved the way to so called plan-driven methods, alternatively know as traditional methods, the main constrain of those methods is the assumption that development teams should have a good understanding of all requirements upfront and that the requirements are not going to change much (Hickey & Davis 2004; Lu & Tseng 2009). Similar classification is offered by (Dingsøyr et al. 2012) who labels waterfall, spiral and the Rational Unified Process (RUP) as heavyweight methods which provide rigid and staged software development lifecycles. Further evolution of the above methodologies paved the way to incremental development models (McConnell 1996) such as evolutionary prototyping (Gilb & Finzi 1988) and the adaptive development model (Wong 1984).

The following is a brief timeline of software development methods:

1970s

Structured programming since 1969

Cap Gemini SDM, originally from PANDATA

1980s

Structured systems analysis and design method (SSADM) from 1980 onwards

Information Requirement Analysis/Soft systems methodology

1990s

Object-oriented programming (OOP) developed in the early 1960s, and became a dominant programming approach during the mid-1990s

Rapid application development (RAD), since 1991

Dynamic systems development method (DSDM), since 1994

Scrum, since 1995

Team software process, since 1998

Rational Unified Process (RUP), maintained by IBM since 1998

Extreme programming, since 1999

2000s

Agile Unified Process (AUP) maintained since 2005 by Scott Ambler

Waterfall

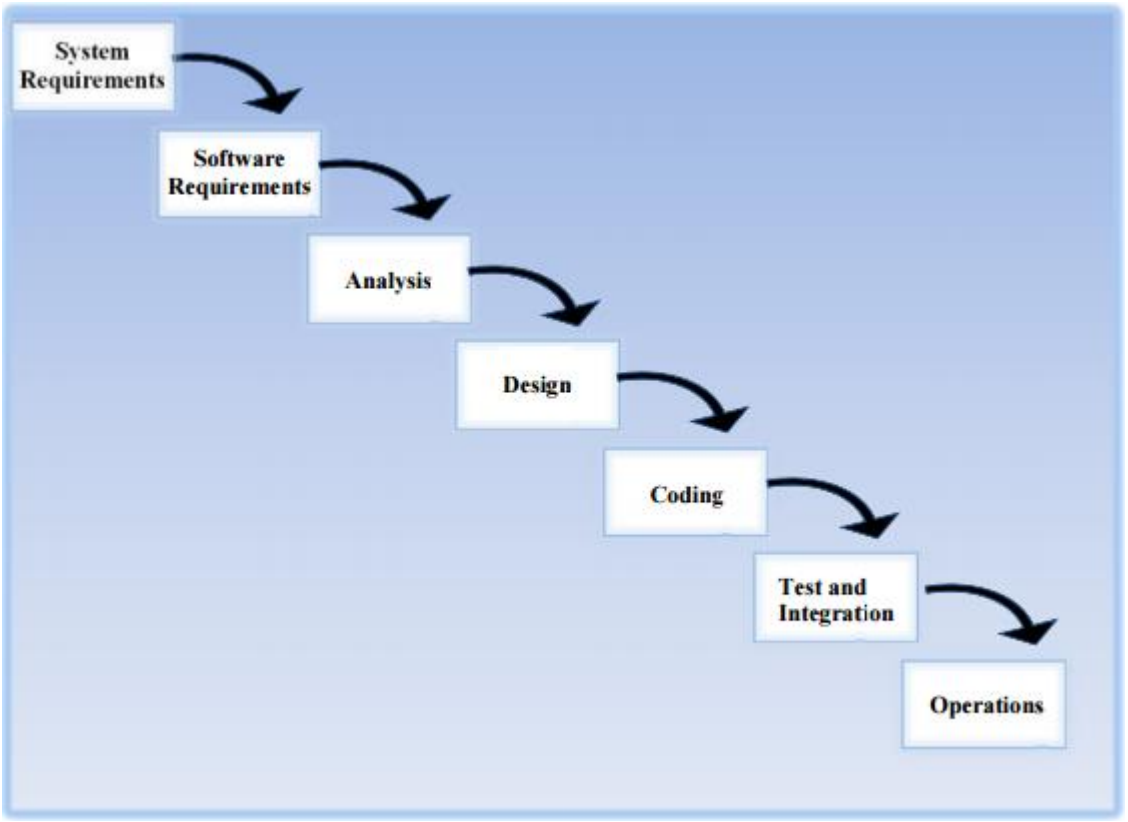

Figure 1 basic representation of waterfall method (Palmquist et al. 2013)

The waterfall development method was first presented by W.W. Royce in 1970 it was created to overcome issues with large software projects (Guntamukkala et al. 2006). Waterfall approach is a sequential design process and can be visualised as flowing steadily downwards through the development stages. It originates in the manufacturing and construction industries which are highly structured and any change to the pre-approved plan is costly and undesirable (Benington 1987).

Over the course of many years of evolution of traditional software development methods, waterfall was widely adopted in large scale projects, especially in the public sector and the military (Palmquist et al. 2013). This was mainly due to the fact that waterfall was considered simple, methodical, well organised, predictable and of high assurance (Boehm & Turner 2003; Fruhling & Vreede 2006). However traditional development methods have a number

of problems such as lack of ability to adapt to changing business requirements, tendency to go over budget, delays and delivering less functionality than originally anticipated (Boehm & Turner 2003; Schach 2004; Sommerville 2007; Watson et al. 1997). Conventional methods also failed to ensure improvements in productivity, sustainability and simplicity (Brooks 1975). Boehm and Turner (2003) and Jones (2008), observed that during the average software development lifecycle business requirements often change north of 25%. Furthermore Williams and Cockburn (2003) argue that traditional development methods were not originally designed to cope with changing requirements in the middle of a software development project. They also mention that the ability to adapt to changing conditions is often a critical factor in a success or failure of a software end product.

Johnson (2001), talks about interesting results in the research by Standish Group in 1994 on a group of over 8,000 projects with 365 respondents representing mainstream software development firms, showing that only 16% of projects leveraging traditional methods are completed on time and on budget as well as with all functionality agreed at the beginning of the project. At the same time just over 50% of the projects finished over budget, over time and not delivering on the entire scope agreed in advance, moreover 31% of projects were cancelled.

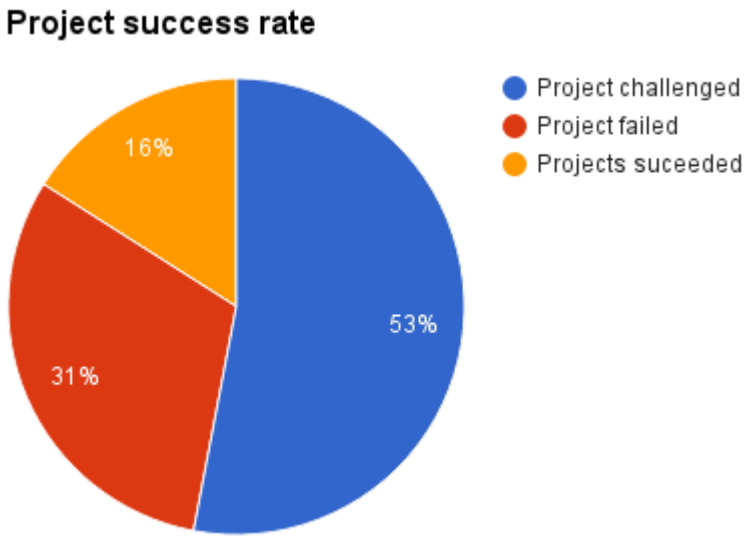

Figure 2 Success rates of projects on traditional software development methodologies (Standish Group Report, 1994)

Taylor (2000), in another study of over 1000 IT projects in the UK also reports that scope management in waterfall projects was the single biggest factor for failure. According to Leffingwell (2007), four key assumptions of waterfall that turned out to be false are:

1. Requirements can be defined upfront
2. Change during the development process is small
3. System integration can be predicted in advance
4. Software innovation can be done on a predictable schedule

Rapid nature of technological progress meant that organisations had to search for more lower risk development methods. Georgiev & Stefanova list them as external risks, cost risks, schedule risks, technology risks, operational risks - which all call for more flexible approach to SD (Georgiev & Stefanova 2014). Waterfall by its nature was rigid and lacked customer feedback which led to the birth of new methodologies such as rapid prototyping (McConnell 1996), the V model (Sommerville 2010) and the spiral model (Boehm 1988). The next section explains the characteristics of agile methodology which has been developed in order to address the shortcomings of waterfall method (Highsmith & Cockburn 2001).

Agile / Scrum

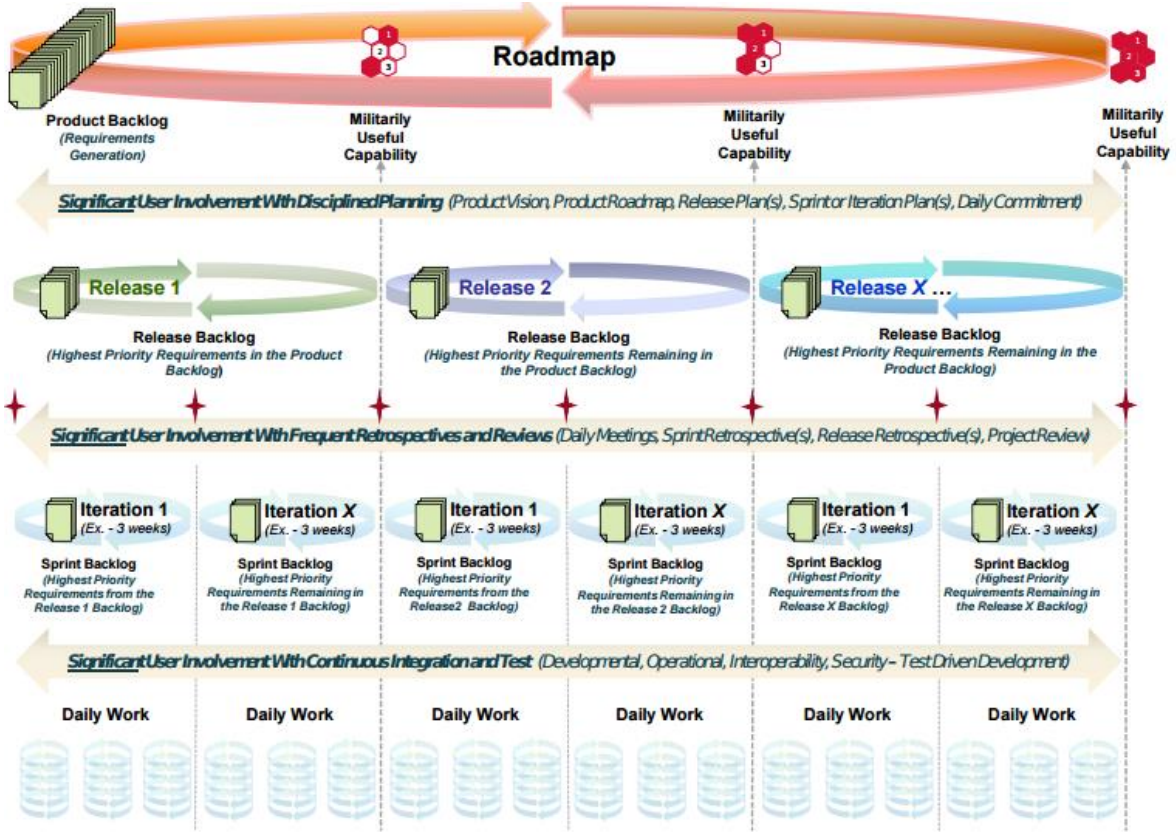

Figure 3 Agile Lifecycle (Palmquist et al. 2013)

In contrast to waterfall model, agile software development promotes adaptive planning, evolutionary development and encourages flexible response to change, as laid out in agile manifesto which defines these concepts (Beck 2001). Agile challenges the traditional methods that were around for decades and provides new values - lightweight documentation, fast delivery, customer satisfaction and high quality (Highsmith 2001)

In a nutshell agile methodology can be summarised as iterative, incremental development which embraces change in the entire systems development lifecycle. Minimum planning, lightweight and speedy iterations, best practices, client-centric, collaborative and frequent software releases (Highsmith & Cockburn 2001; Cockburn & Williams 2003).

No.	Agile principle
1.	Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2.	Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3.	Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4.	Business people and developers must work together daily throughout the project.
5.	Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6.	The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7.	Working software is the primary measure of progress.
8.	Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9.	Continuous attention to technical excellence and good design enhances agility.
10.	Simplicity—the art of maximizing the amount of work not done—is essential.
11.	The best architectures, requirements and designs emerge from self-organizing teams.

12.	At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

Table 1 12 agile principles (Beck 2001)

Scrum is an agile framework which was defined by Jeff Sutherland (2004). However, Sutherland acknowledged that the term and many of the ideas were borrowed from a study by (Takeuchi & Nonaka 1986) where they compared high-performing, cross-functional teams to the scrum formation used by rugby teams. The Scrum framework is suitable for teams of 5-9 people, and can be scaled up for operations and divisions containing hundreds of individuals. It is being used in organisations such as Google and Microsoft and Intel (Schwaber & Beedle 2002). The agility in Scrum comes from the fact that the iterations of incremental functionality that is delivered are done in 2-4 week time intervals. During this 2-4 week period of incremental development, known as a sprint, the Scrum team meet each day to discuss the progress being made. This meeting is known as the daily scrum. The aim of the Scrum team at the end of each sprint is to deliver a 'potentially shippable product increment'. One of the main conditions in the Scrum framework is that once a sprint has begun, no changes can be made to the requirements being developed during that sprint. However, any changes to other requirements which are not being developed during the sprint, such as new requirements or changed priorities, can be freely and openly managed in the product backlog (Schwaber 2004).

Cohn (2009) however warns that transitioning to agile from other methodologies is hard because of how radically different it is and how it requires change in organization on a global scale. He also argues that scrum is radically different for software development teams and goes against the training they received so far - the approach where a problem has to be completely analysed before any coding can begin. Engineers have to forget that approach and learn that full solution is not always required in agile environment.

There is a magnitude of differences between agile development and waterfall. Boehm (2003), for instance believes that the primary objective behind using agile method is instant value, where waterfall is used in a situation of high assurance. He also believes that agile is better suited when there is a lot of ambiguity around requirements, especially in the initial phase of the project, whereas traditional methods could work well when requirements are clearly defined up front. He goes on and compares the two further, see table 2 below.

Project Characteristic	**Agile Discriminator**	**Waterfall discriminator**

Primary objective	Instant business value	High Assurance
Requirements	Ambiguity, great degree of change, uncertainty	Known in advance, stable
Size	Small teams, small projects	Large teams, large projects
Architecture	Designed for current requirements	Designed for current and future requirements
Planning and Control	Internal plans, qualitative control	Strict control, quantitative control
Customers	Dedicated, knowledgeable, collaborating	As needed, focused on contract provision
Developers	Agile, experts, collocated, swarming	Plan oriented
Refactoring	Inexpensive	Expensive
Risks	Unknown Risk, Major impact	Well understood, minor impact

Table 2 Discriminators between Agile and Waterfall (Boehm & Turner 2003)

Agile methods have potential to increase customer satisfaction, lower bug rates, shorter development cycles and ability to adapt to changing business requirements (Parnas 2006; Boehm & Turner 2003). Furthermore agile could provide increased quality, shorter time to market and overall greater customer satisfaction (Parrish et al. 2004). Miller and Larson (2005) argue that agile improves close collaboration between users and developers on the project and that because of short development cycles is well suited to cope with change. Johnson (2002), talks about a Shine Technologies study of 131 software development professionals and companies which use agile method and finds out that 93% respondents mention better efficiency, 49% mention reduction in costs, 88% mention quality improvements and 83% mentioned overall much better customer satisfaction.

Agile methodology also has its critics, some argue that method can become inefficient in certain types of projects and this is especially probable in large organisations (Larman & Vodde n.d.). While Barlow et al (2011) claims that many organizations find agile too extreme and adopt an approach somewhere in between the traditional approach and agile. An example of such an approach would be Dynamic Systems Development Method (DSDM) that aims to combine day to day elements of agile and plan driven approach to management - without having to sacrifice any of agile principles. One of its main critics is Philippe Kruchten (2011) who claims:

“The agile movement is in some ways a bit like a teenager: very self-conscious, checking constantly its appearance in a mirror, accepting few criticisms, only interested in being with its peers, rejecting en bloc all wisdom from the past, just because it is from the past, adopting fads and new jargon, at times cocky and arrogant.”

He lists around 20 “elephants in the room” that agile has to deal with, including things such as; commercial interest in censoring failures, anarchism, elitism and that it is constantly trying to make traditional management look bad.

Previous studies on agile transformations

Williams (2012) investigates how agile manifesto principles are being valued by agile development teams. Agile principles being: Individuals & interactions over processes and tools, working software over comprehensive documentation, responding to change over following a plan. Williams concludes that “agile manifesto creators nailed the principles that transformed software development over 12 years”. This study however was funded by scrum alliance - an organisation which is promoting the use of agile, which might be a case for a vested interest of the funding body in the research result. Furthermore, Laanti (2011) - conducts a study to collect evidence on the success of agile transformation in a large scale software organisation - Nokia. The study reveals that most respondents agree about benefits flowing from agile transformation. The main identified benefits are: higher satisfaction, feeling of effectiveness, increased quality and transparency, increased autonomy and earlier detection of defects.

Kautz (2014) attempts to measure impact of SCRUM methodology on productivity using 7 productivity indicators:

1. Number of interruptions,
2. Endless Development cycles,
3. Repetition of mistakes,
4. Meeting deadlines,
5. Bug fixing time,
6. Number of uninterrupted development hours,
7. Employee performance.

Kautz used the evidence gathered to demonstrate the positive impact of scrum on productivity as measured by the factors above. Kautz argues that open organizational culture is critical to Scrum's productivity and that a change in culture would have impact on productivity indicators (further research needed).

Mar and Szalvay (2011; 2006), classify following stages in the introduction of agile:

Figure 4 Mar an Szalvay (2011; 2006) agile transformation stages

To understand how software development teams view agile transformations it is important to understand at which stage the transformation has been observed / experienced. First phase proposed in this model is exploration in the initial phase, agile transformation tends to be a frightening experience especially to the software development teams who are new to the concept and who are used to working in waterfall style development. In the coordination phase, wider organizational consensus is achieved and more than one team is rolling out agile in a slightly more controlled manner. This is then followed by process definition phase, at this point of transformation, teams within the organization learn how to collaborate with each other, until that point most collaboration and work was happening at the team level. Best practices are being introduced and it is arguably one of the most important aspects of agile transformation that such best practices in return create a good environment for collaboration within the organization. Finally just before full transformation occurs it is essential to achieve strategic alignment which is getting the organisational buy-in from all other layers of business and making sure that they are compatible with the new way of doing things.

Gandomani (2015) points to inadequate or dysfunctional training as single most critical success factor for organisations transforming to agile style development methodology. He claims that the primary cause of the training issues are:

"the consequences of the transformation process and the heuristic and ad-hoc treatments as the strategy used by agile teams to cope with this challenge".

The study uses grounded theory to arrive at this conclusion. Gandomani in another study in 2014 (Gandomani et al. 2014) investigates other critical success factors in agile transformations as: Training, Coaching and Mentoring, Management Buy-in, Team Buy-in,

Right people selection and empowerment, Continuous meetings and negotiations, agile champions, incentive factors - where training, coaching and mentoring is indicated by the author to be most important.

Paasivaara (2014) investigates the use of Communities of Practice (groups of experts sharing a common interest on topic and willing to deepen their knowledge) approach as part of transformation from a traditional plan-driven software development to lean and agile. Author names COP’s are main factor in the success of adoption of agile in Ericsson case study.

III. METHODOLOGY AND FIELDWORK

This chapter describes the research methodology approach, why and how it was selected, and how it was performed. It will also explain how the research was orchestrated, and the data collected.

Research methodology

Methodology is the standardised, academic analysis of the methods applied to a field of study. It comprises the theoretical analysis of the body of methods and principles associated with a branch of knowledge (Ishak & Alias 2005).

Information Science (IS) research methodologies can be broadly summarised into two sectors: quantitative and qualitative (Lee & Hubona 2009; Myers & Avison 2002). There are various research strategies, Saunders (2009) argues that four of them are key: positivism, realism, interpretivism and pragmatism.

	Positivism	**Realism**	**Interpretivism**	**Pragmatism**
Axiology	Perfectly objective.	Generally objective, some bias.	Subjective interpretation.	Both objective and subjective viewpoints are taken.
Most common Data Sources	Qualitative	Either	Qualitative	Both
Epistemology	Results only based on 100% confirmed facts.	Ideally based on facts.	Results might be based on subjective viewpoint.	Results derived from either observed facts or subjective viewpoint.

Table 3 Research philosophies (Saunders et al. 2009)

Yu (2001) argues that it is a common error to equate positivism with numbers and interpretivism with qualitative data, suggesting Saunders could be oversimplifying in his classification. Axiology was the main driving force in the process of selecting methodology for this particular study.

Realism was the next considered methodology, but although every attempt was made to ensure no bias in this study - some residue risk has been identified as questioned participants could have been potentially influenced by bias in interviews which is a problem when the interviewer for whatever reason influences the types of response the respondent gives. Moreover because of the ethics regulations participants would have been made aware of researcher employment which could have generated bias this way. Realism and interpretivism therefore have been ruled out.

Research strategy

In previous chapters of this study, many secondary research sources have been identified. These sources will be used to empower conclusions, however further primary data could expand the body of science as well. The following two methods for gathering primary data were considered:

Online Survey

According to Shaughnessy et al. (2014) survey research can be used to assess thoughts, opinions and feelings. Moreover this kind of research can be either specific or have wide goals. More recently, online surveys are a great way to gather larger number of responses in a short period of time. This carries specific cost advantage over other methods of data mining (Wright 2005).

However data gathered in this fashion has a number of limitations:

- not 100% certain data = demographic selected
- might not represent views, random mouse clicks
- participants could submit more than one response
- there is no option to probe for more in-depth responses

This kind of research was assessed as useful but quite limited, especially because previous studies as identified in literature review chapter had already used online surveying on a large scale. It was decided that face-to-face interviews would bring more value to the body of science.

Interviews (semi structured)

Research interviews are often categorised into three broad types in relation to the level of formality and structure as follows: Structured interviews; Semi- structured interviews; unstructured interviews (Saunders et al. 2009). The key benefit of an interview to this particular study is the ability to gather the data and probe for further details on answers as appropriate, ""adapt the questions as necessary, clarify doubts, and ensure that the responses are properly understood, by repeating or rephrasing the questions" (Sekaran & Bougie 2010). However, interviews also have their limitations:

- gaining access to individuals might be limited / difficult
- data is prone to be biased
- participants might not have time to consider their answer fully

In a bigger study it might be beneficial to use both surveys and interviews simultaneously. However it was established that because of the availability of the secondary research it was more interesting to focus on interviews to try and extract more tailored data.

Ethical Requirements

Ethical approval for this study was sought from the Ethics Committee at Trinity College Dublin in March 2015. The target audience for the research was those working in software development teams within software development orientated organisations, all of whom were to be over eighteen years of age. Participants were asked to sign a consent form before interviews could take place.

Conflict of interest

Potential conflict of interest is that the researcher is employed by major consulting firm and works in agile software development environment. Mitigation of this conflict is proposed in the following measures:

- At the start of every interview employment and position of the researcher will be clearly communicated, research was being performed on a private basis with the knowledge but not the support of that organisation.
- Research participants who might have dealt with the researcher through his current (but not past) employment will not be invited to participate.
- Researcher shall refrain from using facilities (including but not limited to office space, laptop, phone, contact database, etc) in the completion of this research.

Questionnaire

Research participants were asked if they were willing to allow an audio recording of their interview that would be destroyed at the conclusion of the study, and all have agreed.

A mixture of open and closed questions were prepared, covering four specific topic areas:

- Questions for individuals making use of agile software development
- Questions for individuals not making use of agile software development
- Questions about agile principles; early and continuous delivery of software, welcoming changing requirements, frequent delivery, business and engineering working together daily, motivated individuals environment, focus on face-to-face conversations, working software as primary measure of progress, agile processes as drivers for sustainable development, technical excellence, simplicity, self-organising teams, and continuous improvement.
- Demographic questions

Timeline for gathered data

Interview 1	Senior Business Analyst	Male	18/05/2015
Interview 2	Senior Business Analyst	Male	19/05/2015
Interview 3	Senior Engineer	Male	21/05/2015
Interview 4	Senior Engineer	Female	22/05/2015
Interview 5	Project Manager	Male	25/05/2015
Interview 6	Lead Engineer	Male	26/05/2015
Interview 7	Project Manager	Female	28/05/2015
Interview 8	Enterprise Architect	Male	29/05/2015
Interview 9	CEO	Male	30/05/2015
Interview 10	Director of Programme Management	Male	02/06/2015
Interview 11	Product Manager	Female	03/06/2015

Table 4 Interview schedule

Analysis of gathered data

Interview data can be very rich and full of interesting facts and observations. However, it does not lend itself to analysis via traditional methods. Responses that could be categorised have been presented in a chart form. Where this was not possible, statements have been grouped into topic areas and quoted anonymously as promised to the various participants. Each interview has been transcribed and main points then were grouped in using colour coded sticky notes. Additional perspectives for the gathered data have been added from academic literature and from commercial publications.

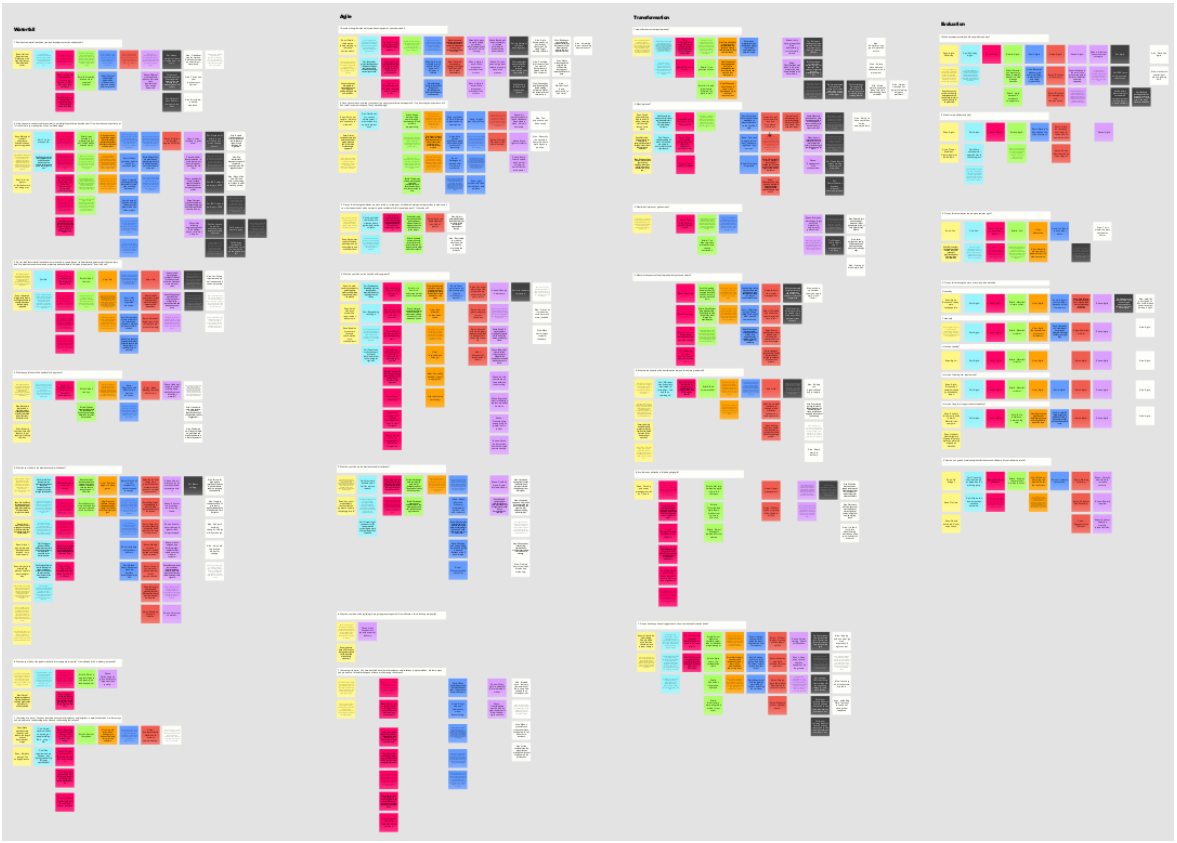

Figure 5 mural.ly software has been used to colour code and group responses in to topical groups

IV. FINDINGS AND ANALYSIS

A total of 11 software development professionals were interviewed for this study. Out of these all have experienced both waterfall and agile methodologies in their professional careers. Majority went through transformation in one or more of their companies and have extensive experience developing in both methodologies.

Finding 1: How software development teams understand waterfall & agile?

Waterfall

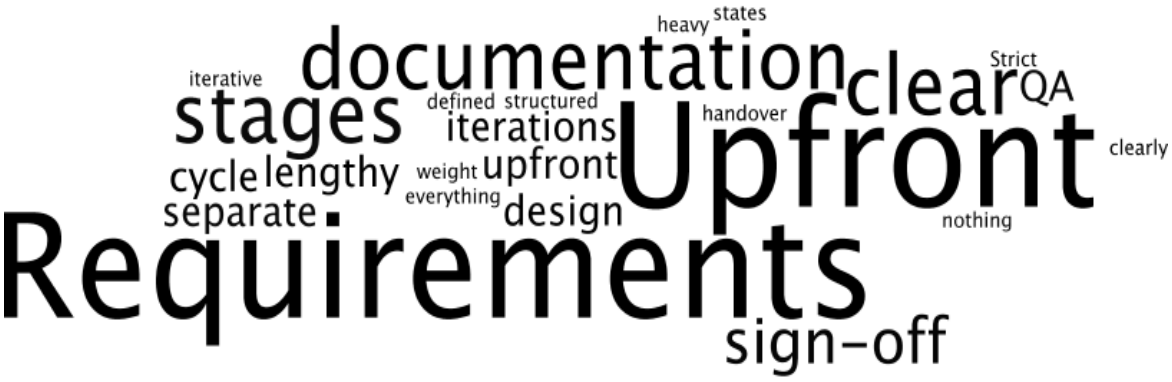

Figure 6 Tag cloud based on most common responses to the understanding of waterfall question

The general theme across interviews when asked about understanding of waterfall was that it is a series of sequential steps with most work being done upfront:

"Upfront requirements gathering, strict sign off process, clearly defined cycles",

"Extensive requirements definition phase, documentation and nothing happens until each phase is completed and signed off",

"Huge effort upfront",

"multiple handover states, clearly defined phases",

"clearly defined stages, requirements, design, development QA",

"Series of sequential processes from requirements capture to the implementation of the system".

Some interviewees noted that waterfall was never meant to be done in lengthy iterations and that Royce in his paper actually warned about dangers of making waterfall not iterative.

"One misconception about waterfall is that you do one iteration instead of doing it multiple times, as per Royce's original paper",

This is a true observation as Royce in his original paper on waterfall suggested that only high level of iterations can result in a successful waterfall project.

"*but in reality from my own experience it is very sequential and iterations are not very often"*.

Royce himself explains the dangers of such approach. He argued that the way it should be done is more "repeat / recycle", if something wasn't working it was supposed to be changed, it appears that Royce's paper wasn't viewed as a whole, so in many cases the implementation of waterfall is doomed to failure.

Agile

Figure 7 Tag cloud based on most common responses to the understanding of agile question

First common theme amongst interviewees on agile is that it is collaborative. Respondents give various definitions which broadly classify agile teams as consisting of specialist working together with the common goal in mind. In other words specialists from different field coming together and pushing the project forward:

“smaller collaborative teams working in iterative environments”,

“lightweight software development”,

“swarming issues knocking down blockers”.

Another common view is that in agile world work is chopped down into smaller more manageable pieces. Requirements are represented in user stories which represent small pieces of functionality which can be independently released - ultimately leading to incremental business value which is covered later in this chapter.

“incremental software development in timeboxed short cycles so that you get some working software at the end of each cycle”.

Most people also associate agile methodology with process, the focus on certain principles, people over processes, software over documentation, customer satisfaction, responsiveness to change, working software. One respondent suggests that people might mistakenly understand that agile is process-less and argues that it is an even stricter process than waterfall.

“agile given its name, people perceive as developing without a process - which is wrong. Agile is as strict of a process as waterfall if not stricter”.

Agile also introduces a wide range of programming practices which greatly improve how software is developed, agile is not only about the process, it is also about good practices. Unit tests, embedded QA, refactoring all the time, dealing with technical debt - none of which were existent in waterfall world.

Another characteristic given by the interviewees is about how in agile methodology you do not have to be certain upfront, you learn as you go. This creates a change friendly environment. Being comfortable with ambiguity has been mentioned several times.

“agile is just enough information to proceed. Get a feel how things are working and figure out your priorities”.

All respondents mention one or another form of prioritisation in agile and how it affects day-to-day projects. Stories form project backlog and as complexity emerges new stories can be written and backlog can be re-prioritised. The measure of success in agile is often that the work that was not essential was not done, as it helps free up resources and utilise them elsewhere.

"agile is backlog of known features, prioritised according to business value and estimated"

"way of minimising work in progress to a minimum and focusing on the most important features to be done".

Most respondents talk about the concept of working software, which is one of the main principles of agile development, it is the concept that software should be built to fulfil burning business need and this should be demonstrated to the business as soon as possible.

"working software as a measure of success"

"agile development is a philosophy whereby you are trying to get working software".

Finally the last two common points that emerged were customer satisfaction and ability to accurately plan. The concept that the entire team works to deliver on the simple fact that happy customer is the key, really gives perspective to the development team, they no longer have to work on abstract tasks but tasks that have tangible benefits to the customer:

"agile to me is taking a much shorter time to figure out if the customer is satisfied",

"complete transparency amongst the team that lends itself then to other stakeholders"

Customer satisfaction also extends into business value, another core principle of agile methodology, not doing things just for the sake of doing them. Everything in business value has a dollar sign attached. Finally from managerial point of view, agile is being praised for its ability to report but also predict progress.

"you can have a high level release plan very early in the process"

Respondents were also asked to compare agile in waterfall, with the below results it has to be pointed out that the sample size of the studied group is rather small and there could be a high degree of bias towards agile due to the nature of the study.

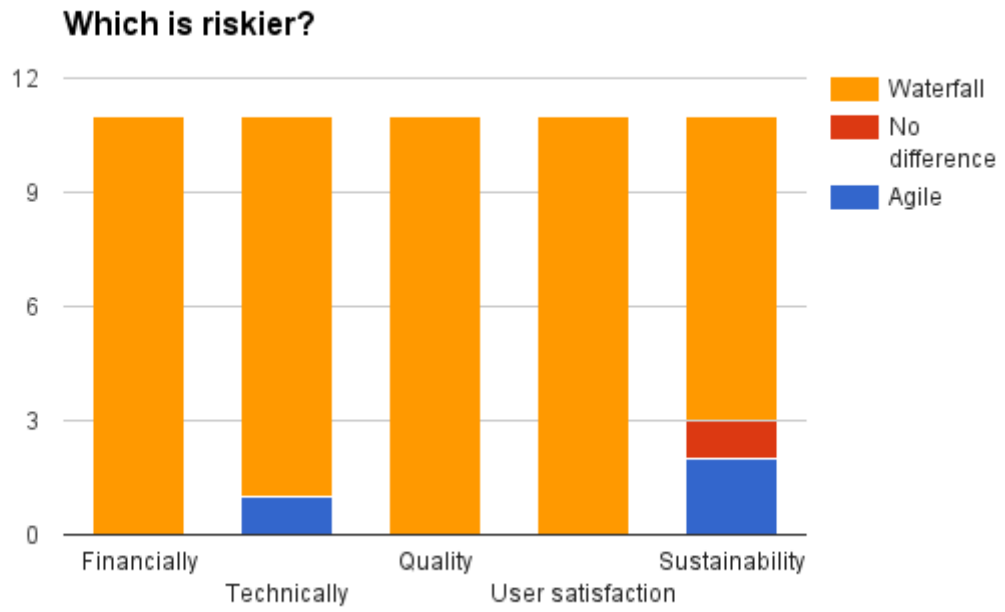

Figure 8 which methodology is riskier?

Most respondents unanimously claim that agile is less risky from financial, quality and user satisfaction point of view because of the feedback loops it provides. In typical waterfall the bulk of the costs lies in the design phase and there are less options to back out of the project or make changes. Therefore agile poses less risk due to the inherent checkpoints. The same goes for user satisfaction in which the customer gets to see the results of work much earlier in the process. When it comes to quality, one of the main reasons for it being less risky than waterfall is its insistence on good practices.

"Depends on the contract. Agile in long run. Not a single startup would use waterfall - that tells us something."

From the technical point of view respondents also point at waterfall as the much riskier of the two. It is argued that waterfall gives the illusion of less risk because traditional project managers prefer waterfall, however it is much easier to control delivery in agile. One respondent, an experienced technical lead, claims that waterfall could be less risky than

agile when it comes to code maintenance because developers have more time between iterations to improve the codebase.

“From technical point of view, one part of risk is that project performs as expected. Waterfall could have a slight edge.”

Finally long term sustainability gives a little more varied answers, two respondents point to agile as a little bit riskier in the long run and one said that there is absolutely no difference. The rationale behind this is that agile performs best in green field type projects and that when it comes to long term support and managing existing systems waterfall could not be such a bad option because of the strict change control mechanism.

“In large organizations development teams and support teams are not the same.”

“Possibly the only situation in which Waterfall can compete. However with the right mix of people in place agile can work just as well as waterfall.”

Finding 2: Waterfall could be suitable where failure will be catastrophic

Respondents were asked to assess in what case waterfall methodology would be suitable to a given project. The most common conception was that waterfall would be optimal for highly regulated or catastrophic failure types of projects. Examples are given in the air transportation industry or medical devices, any project where a failure would be catastrophic and where a massive amount of QA is needed. Banking and finance is also mentioned due to the highly regulated environment with compliance stakeholders - who expect everything to be detailed and delivered as per documentation, central bank issuing guidelines type of initiative that needs to be rolled out.

.

"it is in line with how auditors would sign off on a given project. Once you have heavy documentation, years from now auditors could have access to that information", *etc"*,

"banking / highly regulated environment but not because it is better but because they are more comfortable with it"

This kind of project environment attributes the biggest chance of success for a waterfall project. If the organization has enormous amounts of time to spent in the design phase - it could work well in waterfall. Whenever requirements are given and cannot be changed, where change is simply prohibited in a regulated industry such banking, where you have to deliver regulatory functionality on a given date. Where the delivery expectations are absolutely cast in stone, for example in stock broking- where the project team only get access to a trader for a very short period of time upfront - not allowing for agile model to work.

"it could work well in a organization with very heavy control and governance mechanism - you will have a lot of throwaway work - but in the end it will deliver".

Some respondents make a link between manufacturing and waterfall and how in the past when software development was done on a low level it was easier to work on a waterfall project, another interviewee points out that:

"*Today if you want to build a screen you put a widget in the environment and add labels and you are done. When these environments were not around, someone would actually have to draw a screen, draw a line etc. to show boundaries of the screen"*.

On the other hand most interviewees attribute the best chance of success to product based projects as best candidates for adopting agile, projects where you are constantly improving that product. Agile is also believed to be more suitable to greenfield projects, where a brand new piece of software is being developed as opposed to an existing / legacy system. Facebook and Spotify model are being given as an example of the kind of project that works well with agile.

"look at facebook / netflix model - they change things, get feedback and if people don't like a particular feature, they remove it."

In general agile is recommended in a situation where requirements are uncertain throughout the duration of the project as methodology better suited to dealing with ambiguity and at abstract level.

Some respondents go further and argue that agile would be suitable to most projects without exceptions, there are cases when rollout of agile might not be needed but best practices should be still ported over:

"I don't think i could think of a situation where agile would not be suitable"

"you don't necessarily have to follow agile to the letter every time - as long as you can adapt what is best about it".

Finding 3: Upfront design does not reduce errors or inflexibility

One of the classic arguments for using waterfall development method is that the time spent at the beginning of the project in the design phase provides savings in latter stages of development by a factor of 50 to 200 (Benington 1987). It is common for companies to spend 20-40% of project time in the design / development phase (oxagile.com n.d.). However it appears that in practice, many of the interviewees claim that it is nearly impossible to get things right at the beginning and that a lot of the time, upfront designing is a wasted effort.

"First of all the idea that you can capture everything on the outset is mad (..), no matter how well things are understood in advance or are perceived to be understood, it is only when the first line of code is written that people can see what really is going on".

From an engineering point of view, an argument is made that discussing requirements or design solutions without getting into the code is not going to work. Technologies that are being worked on are so new that there is simply no way to understand the complexity upfront. A Senior systems architect, points out that in his experience it would not be unusual to spend a couple of months in the design phase before any code was written. The development team would be handed set of sequence diagram and told to build exactly this way. Another interviewee adds that:

"not involving engineering team in the design phase costs a lot in long run".

This view is also represented when interviewees describe their professional experience in waterfall, many point out that it is not possible for people involved in the design phase to fully understand requirements upfront. Sometimes they could not envisage full details in advance - which often causes teams to miss-deliver features. In such cases McConnell suggest that it might be more beneficial to change the design rather than persist in the design that does not take account of changing conditions (McConnell 2004)

Another argument against spending too much time in the design phase comes from a senior development engineer who states:

“Technologies are changing constantly, by the time you are using new piece of technology you are not able to fully understand it. Architect is not able to foresee everything in advance - it is impossible”.

This argument is discussed further latter on in this thesis in the resistance to change chapter. Furthermore multiple interviewees argue that waterfall origins are in manufacturing and that it could work in a simple process, but is not suitable for software development as we know it today:

“It is a daft idea. It comes from the factory setting from other industries where development / production phase isn't as complex as software development.(...)You do a blueprint and then have the manual on how to assemble. In construction industry it could work, because you can hire bricklayers carpenters, and so on. But not in software development which is highly complex creative / abstract process”

Upfront design also leads to the conundrum whereby spending so much time in the design phase makes it more difficult to change that plan. Projects start with big specifications, when it comes to development if you run into problems and, for instance, there is disagreement with the design or assumptions, there is no way to challenge it. Because so much time and resources went into the design phase it is seen as infeasible to change things at the time of development.

“You could see big mistakes being made but there was absolutely no possibility of correcting them“.
Respondents also note that the upfront design leads to another problem where requirements / solutions become too big to manage.

“requirements get too big and everyone knows it is not going to be delivered”

This makes waterfall projects very difficult to manage. This view is further supported by a couple of software engineers who note, that often when development is completed and presented to the customer it turns out that it is not what they originally wanted - this is both de-motivating and means there is inherent redevelopment.

“You finish development of one feature, but it is not what user wanted so then you have to change it again - it burns time”.

Moreover, during the course of the project, requirements that are growing beyond the point of control mean that it is difficult for the project team to figure out what it is that they are mean to develop. There is a huge loss of understanding of what the base requirements were originally. This is in contrast to agile development which focuses on the amount of work not done as a success factor, waterfall has the tendency to over deliver because of unmanageably big requirements:

“(...) features were delivered but were not required / used by the user”.

Finding 4: The ability to adapt to change is a key to competitive advantage

"It is not the strongest or the most intelligent who will survive but those who can best manage change. " Charles Darwin

Adaptation of the organism is the changing of the structure of function of life in a new environment, permanently changed living conditions or external stress. The effectiveness of adaptation determines the adjustment. Adaptation is an essential, inherited or non-inherited trait and is subject to the development of all living organisms. Similarly to the biological definition of adaptation, the ability to embrace change in software development industry is of the utmost importance because of rapidly changing landscape. It is no longer a case that being big means being successful.

Change is unwelcome in Waterfall

"The limitations of waterfall are way too glaringly obvious. The idea that someone can imagine every possible outcome 6 months before a project start without any interaction with the team - is just ridiculous."

Upfront design in waterfall ultimately leads to another issue, organisations who spend all of that time in advance planning and preparing minute details of the project find it very difficult to accept changes at a later stage. This behaviour has been called out multiple times in the interviews. It is claimed that waterfall does not facilitate change and change becomes costly the further into the project you get. Change is always questioned as something that should have been thought through in advance.

"Change isn't welcome because you would have to explain to stakeholders what we thought was going to work 6 months ago is not going to and we have to start again - good luck with saying that".

This resistance to change is represented across multiple layers - delivery managers protect original plans as they feel there is no time to change the course without delaying the project. This attitude is then passed over to the delivery team who is working on solution that there is little faith in, which further reduces morale. When asked about suitability of waterfall to a certain project, one respondent noted:

“Not suitable at all, as you can't predict things in software as easily as in other industries. Software industry changes at a great peace and waterfall is not able to cope with that change”.

Amongst the interviewed group difficulties in managing change occurred in most of the waterfall projects. Commonly when development was finished business users wanted to do things differently. Most respondents pointed out an inability to cope with change as single biggest problem with waterfall.

“dangerous if anything changes - things always change”

“You can’t cope with change”

“ it is not responsive, cannot respond to change”.

Lengthy iterations

Iteration length appears to be one of the hottest topics of the study, all respondents point to lengthy iterations as one of the main issues with waterfall, while some rightfully note that Royce himself mentioned that long iterations are going to make waterfall infeasible. Therefore an argument can be drafted that implementation of the waterfall across the board is flawed.

“you simply can't be waiting years or even months for feedback”

“waterfall means long time until product delivery”

“very slow to market”

Long iterations, combined with requirements that are too big to manage, artificially create landscapes where a very strict sign off process is required, this leads to overly complicated documentation, covered later on in this chapter. Lengthy iterations also have a negative impact on team morale.

"Sense of hey, we have to do so many things - but it is 6 months from now, so we don't have to worry now"

"*6 months to a year iterations. And by the time you received the spec you knew it was impossible to finish everything that was in the spec by the deadline. Loads of pressure".*

One of the proposed approaches for addressing the issues of lengthy iterations includes Royce's final model which talks about returning to the previous cycle once issues have been discovered or going back to the design phase all together (Royce 1970). Most of the respondents gave similar answers when asked how this could be improved in waterfall, some of them noted:

"waterfall could work well if chunks of work are partitioned and project is delivered in a form of increments"

"6 month iteration work quite well, but as soon as the iteration are longer it could become a big nightmare"

Manageability

This area does not to seem to be covered extensively in the literature, but came up a lot in the course of interviews. Many respondents point out that in waterfall it is hard to estimate big pieces of work in any meaningful way which ultimately leads to a very poor ability to plan and manage.

"waterfall deadlines - a lot of the time are wishful thinking"

Managers however seem to prefer waterfall because it is familiar in the traditional project management definition. However it gives a false sense of security as often these project plans are far from reality. Another group of respondents claims that from a project manager perspective it might be perceived as easier to manage, because of the familiar deliverables such as milestones. However this is at odds with the experiences of the people who work on the ground:

"a lot of project milestones were not being met"

"normally the deadline was not met and dev team needed another 6 months to release software. Always stretching over",

"much of the time it felt like fiction, readjusting your plans to what was happening".

Interviewees with management background also see the positives of waterfall and claim it is easier to communicate with other business functions such as finance. Some but not all respondents see management in waterfall as much more structured in a traditional project management way, with clearly defined plan, milestones. They also argue that shielding a project from change might be a good thing as it is easier to keep the scope of it under control.

"When people see things in iterations they tend to ask for changes, 'could you do it this way or that way' to ask for changes. 'could you do it this way or that way'"

Not all managers speak of waterfall favourably, one in particular made a comment that the bigger the team, the less it worked. And the greater the chance that there will be budget / time overruns and decrease in customer satisfaction.

Agile thrives in changing environment

On one side of the spectrum we have a rigid waterfall methodology that does not welcome chang,e on the other hand we have agile, which embraces the change, allowing organizations to adapt to current conditions. Change is where agile methodology really thrives, as it is structured so that it provides feedback loops, if something is not working it is simply changed without spending time or resources on trying to get it working for no reason.

"change seems to be treated as learning process, iteration in early and you actually present your code much faster in the process".

At the current pace of technological change the risk is far too great not to be able to cope with change. Therefore the ability to adapt is critical to drive customer satisfaction.

Finding 5: Agile leads to greater customer satisfaction

“1000 times better customer satisfaction.”

A key advantage of agile mentioned by many correspondents was its ability to elicit what customers really want. At the start of most major IT projects, few customers are clear on their exact requirement; their view of their requirements changes as the system develops and, for example, they see new possibilities that they might not have heretofore been able to visualise. The following are two typical comments:

“*Most customers don't know what they want”.*

“*[Often a] customer doesn't know what he / she wants but at least they start seeing things earlier and they have a chance to argue whether they really want it or not. Agile is the only methodology factoring that in. It can't work in waterfall project”*

Respondents see in agile a major advantage over waterfall in delivering a final product better matched to the customers’ actual needs. This structure allows the team to advise the customer on the direction and address most burning issues right out of the traps. As one person put it, it gives developer the:

“…*ability to listen and respond to customer needs. Because at the end of the day software development is all about delivering towards requirements of users”*.

Agile gets customers much more involved in the development process, it provides plenty of opportunities to exchange feedback with them and this is quite the opposite scenario to the submarine effect in waterfall discussed later in this section. One senior programmer summed up the benefits in two words:

“happier customers”

Agile does an excellent job of switching the focus of everyone on the project to drive business value. Because everything has a dollar sign attached, it is relatively easy to decide which feature to pursue, because it is dictated by the business value, and it is always the case of achieving best value for money. This type of focus also helps build better relationships with the customer as they can see that they have a say in what is getting delivered:

"agile lets the customers see the impact of their prioritisation, lets them see they are in the driver's seat".

Sub-marine effect

The submarine effect as described by a senior programme manager, but also refered to by other interviewees, is the notion that from the time when initial requirement has been acquired from the client to the time they see the software they requested - the customer is not engaged. This is probably one of the biggest flaws that this study has discovered in waterfall.

"I will give you my requirements and you will be back to me in 6 months".

The submarine effect is a manifestation of how resistant to change waterfall really is and how distant the understanding of a product by a project team is to what the customer had in mind, it is very hard to pivot and adapt for change.

"*the disappointment because no matter how well you described requirements it is not in 100% what the client had in mind"*,

"(...) by the time, things were presented to the customer it was generally not what they asked for".

They might want to change their requirements towards the latter stages of the process which triggers redesign and modernisation activities which ultimately end up increasing the cost of the project. Parnas' argument seems to be holding true with interviewees saying:

"Sense of isolation from the customer, that grew with the size of the project"

Many respondents mention that even if the team truly believed that they developed what customer had written down at the beginning of the process, they were still unhappy - because things have changed over the course of the project.

"when customers are unhappy they look to other suppliers and business is lost".

One of the interviewees, brought up extreme example of how bad submarine effect can get, where a project was running on a single iteration for over 2 years, software was written that nobody knew how to test. Respondent left the company when the project was not completed and found out that the customer whom the software was originally intended for, sued the company because they did not deliver anything.

Short iterations / working software

Another way which has been proven to drive customer satisfaction is short iterations. They provide a way of constantly checking that the project is on track from the customer point of view, if the iterations are frequent enough at the end the work is being showcased to stakeholders. This practice minimises risk and allows the team to adapt to change rapidly (Beck & Andres 2004). In practice respondents note:

"It is possible to have a loop of feedback done within a week - which is invaluable"

Short iterations enable much more focused conversation with stakeholders, it is much easier to get customer feedback. Agile goes further and each iteration should deliver working software artefact, every 2 - 3 weeks, a physical working deliverable which is what the customer ultimately wants. The customer gets what he wants faster, but from the software team point of view:

"splitting things into small manageable chunks makes it much easier to stay focused"

There are also voices from experienced engineers who say that short iterations have a potential to be disruptive. From the code point of view it can weaken code base, as everything is done on a 2 week radar rather than on a 6 month radar.

"What you can end-up doing is a quick fix on top a quick fix. Because of the 2 week iteration"

Finding 6: The day of monolithic documentation is over

Too much documentation is involved/created

A common pro argument for waterfall is that it puts emphasis on documentation and source code and that in other methodologies knowledge can be lost overtime (Technologies 2012). Furthermore literature suggests that waterfall has clearly defined stages and therefore is easy to understand and follow (Hughey 2009). In reality however, the documentation suffers from over-documentation which is mentioned by many respondents but particularly accurately described by one of them:

"writing documentation for the sake of writing documentation. Same person reading it will be writing it. Documentation gets easily out of date".

Documentation artefacts are getting too big and hard to keep up to date, a lot of paperwork is involved and documentation becomes cumbersome.

"As a BA you are given word document template which when populated could have couple of hundred pages that nobody ever reads. Repetitive. Once completed, already out of date"

Often there is a case of documenting something without discussing it with the team and just handing it over, as if everyone would arrive at the same understanding, this is an artificial approach to documentation - a word document describing how interface should flow. Maintaining documentation becomes a task in itself consuming valuable project resources, clearly balance is broken between what is considered necessary and what is considered excessive.

"multiple version of documents, quickly getting confusing and outdated",

"big cumbersome documents, very hard to maintain consistency in a 100 page document"

There were also couple of positive points being made about documentation in waterfall. These are captured in the following quote:

"perhaps documentation could be considered a benefit - everything in waterfall is very well documented."

Agile's leaner approach to documentation works

Scott Ambler argues that the documentations should be "Just Barely Good Enough" (JBGE) and that overly comprehensive documentation normally leads to waste and that developers rarely have faith in detailed documentation because it usually not in sync with how things were coded - at the same time documentation that is too little detailed can lead to issues with knowledge transfer, communication and learning patterns (Ambler 2007). Agile encourages lightweight documentation in one of its main principles, "working software over comprehensive documentation" (Beck 2001). This viewpoint is shared by the interviewees who describe the approach to documentation in agile as:

"documenting enough to get going, less is better"

"scribble something on the board and start working"

There are however extreme cases where people coming from waterfall into agile have a misconception that no documentation is required. This is utterly false, because, as Ambler argues, this creates issues down the line when it comes to onboarding new team members, knowledge transfers or simple communication.

Finding 7: Technical teams adapt to agile faster than management, but before management buy-in the transformation cannot be completed

Strategic Alignment

"One of the most critical things for transformation is to get buy-in not only from the team but also from the leaders."

"You can't be agile when your hands are agile and legs are not."

This is probably one of the single most significant findings on transformation in this study. The transformation seems to be progressing very quickly at the team level, however getting the organizational buy-in can be tricky and all respondents quote examples where it did not work well. All of the interviewees have experienced problems with strategic alignment of agile in their organisations.

"there were challenges, because the commercial side of the business was still very much waterfall focused. We decided to roll it out at the engineering level first and made a fatal mistake of not getting the senior management bu-in first. So what ended up happening - we were developing in agile style, but sales team were saying we absolutely have to follow what is in the contract, I don't care if we deliver earlier - i just want to get paid.You are allowing the customer to change their mind too much, scope creep is not welcome if we are not charging for it".

In any transformation project there is always a lot of emphasis on the development team themselves, but the senior management layer, as well as non technical business units, are often left out completely. The change that agile introduces is very significant and without buy-in from the top to the bottom it is doomed to failure. For example getting people in finance to budget based on the number of scrum teams vs traditional approach takes a lot of convincing. Getting buy-in is important because different functional groups within the organisation have to understand that they are getting a small piece of useful functionality frequently rather than big deliverables once or twice a year. Many respondents also note that getting the strategic alignment right, greatly accelerates roll out of agile within the organization.

How to achieve strategic alignment?

Strategic alignment or buy-in at a senior level is all about communication. Introducing agile causes a lot of disruption which not everybody is comfortable with. The easiest way to get people buy-in is to understand what their fears and requirements are. When the buy-in is achieved there is less fighting and more continuous improvement. It is critically important to sell the concept not only to the technical teams but also to any other impacted functions of the business, the key is to make them realise the benefits that come with agile:

"Making sure the business understands that we are trying to build working software and expand on it - businesses tend to say 'no, it has to have this set of features'."

Working in isolation and blame game

One of the reasons why development teams adopt agile faster is something mentioned by most of respondents in the study; tensions in the waterfall projects and what has been named as the 'blame game'. Waterfall is being seen as blame heavy.

"Document was bad, therefore development was bad",

"Artificial environment with scaffolding - blame game as natural defence mechanism vs having a conversation about the issue"

Because of waterfall's structure and clear stages, there is a tendency for project members to blame one another for not fulfilling their part of work properly, therefore making it more difficult to work together. For example it would be common for QA to blame engineers for poor implementation, engineers to blame business analysts for unclear / incomplete requirements, business analyst blaming architecture for gaps in design and so on and so forth.

"sometimes it was a case that neither BA's nor Engineers fully understood the requirement - which ultimately caused a lot of pressure because you would have heated conversation between the 2 groups".

This has been called out on multiple occasions and linked to general dissatisfaction when working on a waterfall project. Waterfall implies that all thinking was done in advance, therefore it should not be questioned in other phases of the development. As long as developers do everything as per specification they cannot be touched, this creates a vicious

circle of putting more detailed requirements in and making them even more complex. This behaviour naturally has a tendency to de-motivate people, by isolating them into abstract tasks rather than keeping them focused on the end goal - which should be delivering customer satisfaction.

"Sense of working on complete abstract tasks that have no meaning, was a lot more prevalent on the bigger projects than on smaller projects. On a smaller project you would typically sit next to requirements guy and you might overhear his discussion with the customer".

Another example of how pathological this could get was from a business analyst with years of experience in waterfall.

"*Writing emails - if I don't hear from you by a given date, I would have assumed sign off. You have to cover your back to have something to show"*.

Fencing off stages in waterfall projects is creating an environment which is limiting communication, remediation has been proposed to include techniques for improving human to human interaction such as daily standups and introducing more feedback loops, both with the client and within the team.

Autonomous teams perform

Agile concept of autonomous teams has its roots in the business management model and was pioneered by Eric Trist after the end of World War II, the concept states that the working group should be able to decide for itself how the work should be done and split amongst the members of the team (Roth 2011). This concept is praised a lot by the software professionals as something that works very well in agile.

"Members can bring their own experience to the table and have a say in design decisions"

"people can make their own intelligent decisions, rather than waiting for sign offs"

Involving the entire team much earlier, helping to harvest all the brain power; this approach is also being portrayed as a remedy to blame game in waterfall, team spirit, focus on delivering as a team vs witch hunt in waterfall.

"sense of pride in the team, people would feel dishonoured if their build broke"

Having the team deciding on what to work on and how to do it also has a positive impact on the overall software design, it is much easier to only gather the bare minimum information to get the project off the ground. Engineers input in the design process broadens horizons. This approach also improves communication within the team, something that waterfall is clearly struggling with.

Agile is making job more enjoyable, therefore software teams adopt it faster

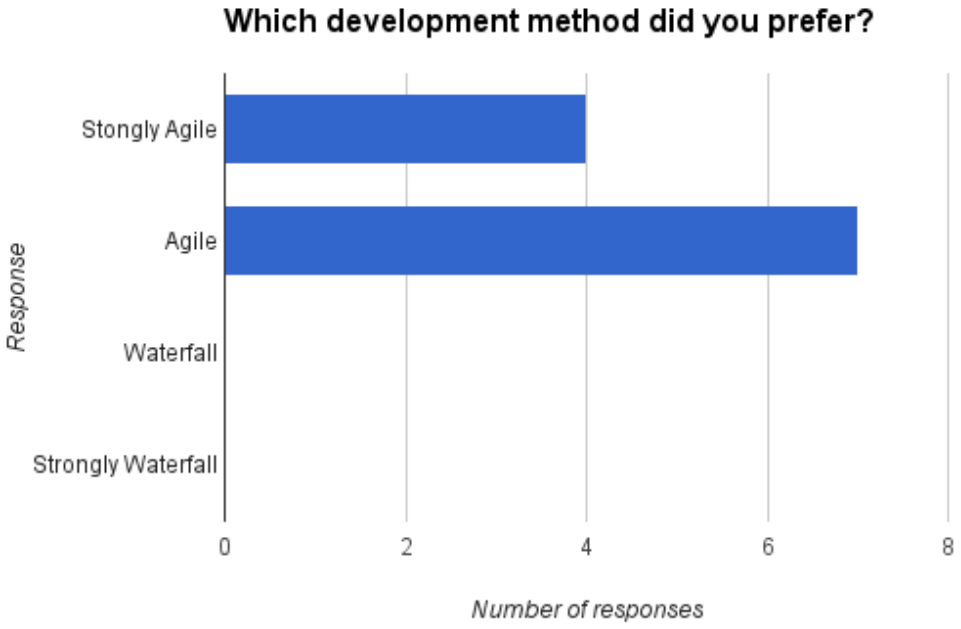

Figure 9 which development method did you prefer?

"If you take well organised waterfall team and well organised agile team - agile wins every time because it is iterative and you don't have to know everything on day one. Just enough to get going"

"Because it makes my job more interesting, better sense of belonging, productivity"

"Because of transparency, communication and team spirit"

"Everyone in the roles in the process are able to perform role better and more enjoyably"

“Happy team. Good design. Motivated individuals”

In general across interviews agile methodology has been attributed to making work in software development far more enjoyable and increasing motivation. An interesting point is made by a lead engineer:

“IT problems are so abstract that if you tell your brother and sister what you did today - you can't really do it. What agile provides it takes you away from the abstract. It says your job is to satisfy customer and he or she has this thing that he or she wants. Which you can talk about to the family”.

Continuous reporting tools and always working towards business value also contributes to feel good factor in the software team, as it allows team members to feel better about themselves because of the reporting tools - they can see that they are getting paid to deliver something to a customer who is happy. Shorter iteration times also have impact on how much more teams are prepared to cope with the technology, a great example of this has been brought up by one of the interviewed engineers:

“sometimes IT can be a scary job, if you have a release every year - if something goes wrong on sunday at 2.00 at night - you have few hours to fix it until directors get involved. - agile reduces this factor as you are releasing much more often, it becomes less scary.”

Finding 8: Death march to delivery / burnout issue has to be dealt with

"In waterfall people might get bored and leave, in agile people might get burned down and leave."

Probably one of the most significant findings on agile in this study, death march to delivery is a concept where development teams are continuously developing features without any sort of down time. Constant pressure to deliver is not necessarily intrinsic to agile, but it easy for business to just throw new tasks at the agile team and the team works flat-out to deliver and then there is another thing being thrown at them. This finding makes 7th agile principle sound heroic and really unrealistic:

"Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely." (Beck 2001)

Death march to delivery and programmer burnout, will eventually affect morale. In waterfall on 6 months release cycles developers experience increased pressure to deliver only at the end of each iteration, in agile delivery happens every sprint.

"agile = spinning wheel of delivery and you really need to find some downtime to avoid burn-out"

Respondents had some ideas on how to potentially remedy this issue, suggesting that every 4th or 5th sprint should be consolidation sprint. Often times the business doesn't want to do that because business doesn't want to do this as they choose feature over consolidation. Also a suggestion has been made to follow the footpath of the likes of Google, who gives their programmers up to 20% of their time to work on their own projects within the realm of Google products:

"all developers have internal task lists of things that are bugging them and they rarely have time to work on them, any initiative that would give them some time to tackle them would be very beneficial"

In terms of long term sustainability the issue of death march to delivery or developer burn out has to be addressed, especially because of the huge deficit of software development

professionals on the market. It is probably easier for an organisation to address the issue rather than having to deal with high staff turnover.

Finding 9: The QA approach in agile is still not perfect

QA approach in Waterfall

The interviewed group had similar experiences with quality testing in waterfall, which was generally pushed out as a very last stage of the development cycle. However it has to be noted that in theory there should be various quality tests at different cycles, according to the experience of the sample group this was not the case.

"testing was a significant issue - we generally tend to leave QA as a very last step - and that was often overlooked in order to stick with the deadline",

"testing phase - testing everything at the very end of implementation. At that point you either jump to new development work or worse you wait for testing to finish"

"nobody is testing anything up until very late in the process. You are wasting time by not finding bugs from that time"

A significant number of interviewees mentioned overlooking the QA phase in their experience or removing QA altogether as a time saving measure and that it was a common practice to produce a substandard quality code based on the judgement call. Waterfall provides a distinct period of time for QA activities, however delivering on a given date is often becoming more important than quality. Another issue with the QA cycle in waterfall points back to lengthy phases of a project, pieces of software might pass QA, but these pieces of software might not be what the user actually wanted.

"specifications / tests are written in isolation and at large time intervals, which means the person who written test scripts might not be working in the organisation anymore"

It would not be uncommon for projects to have a QA phase 6 months after development is completed which dissolves understanding of functionality over time.

Approach to QA in Agile

Specialised tools and techniques, such as continuous integration, unit testing, pair programing, test driven development and code refactoring are commonly used in agile projects (Larman 2004). This enables agile teams to really focus on quality at the time of writing the code rather than trying to patch a working system at a later stage. QA members are also considered developers and each team member delivers towards the same goal,

embedding QA in the team fully and not having them separated. However the issue with QA is not necessarily a weakness of agile or waterfall method as such. Both could have the same sloppy approach irrespective of the methodology.

Finding 10: Managing vendors is easier in waterfall

A couple of interviewees touched on an interesting subject; waterfall suitability when working with offshore vendors. From one side it has been mentioned that the biggest problem with the offshore teams would be that they only deliver what is specified in the documentation. However at the same time a senior programme manager argued that it is easier to work when vendors are involved, especially to control the parameters of the engagement. Further discussion revealed that in waterfall projects the emphasis was on the vendor to provide resources and know how to deliver on a project, this however is not the case in the agile world. There is also a perceived benefit from the business desk point of view*:*

"A lot of the clients who want to limit the risk, they would much rather have these phases in a fixed price contract".

Similarly vendor management is one of the few areas that was identified as troublesome with agile methodology, where a company is delivering for a fixed price contract and they want to have set scope, set budget and fixed deadlines. Respondents at managerial level criticise agile for dissolving the line between client and vendor:

"you stopped having client vendor relationship and really just started taking bodies from that vendor. You provide engineers, we provide scrum teams and you are not providing any expertise to us beyond providing team members"

This is a departure from tradition, usually a company was not only buying bodies to join teams but was also buying technology and know-how. A vendor is engaged to fulfil a certain task, for example a delivery of a system and was held responsible throughout the process. An argument is made, that in agile there is much more emphasis on the client to have the know how and that it could work well for a big company with plenty of in house expertise but could be very troublesome for a company that does not necessarily specialise in software development. Finally in this model where client is the brain powerhouse behind the project, interesting observation is made by senior program manager:

"although offshore teams appear cheaper because you are paying a third of the rate, but when you factor in all of the middle management layers that they are charging for - costs become much closer".

Finding 11: Coaching and training is essential for successful transformation

"Assuming you have the buy-in, training is critical. Making sure you have appropriate skills amongst your delivery team."

"All just to do with training. People within each function should receive proper training."

Most interviewees attribute good coaching and training as one of the most critical factors in the success of agile transformation, especially during the initial phases of experimenting and coordination but also later on when trying to get wider organizational buy-in,

"how successful agile is depends on how strong it has been championed in a given organisation and amount of training that has been rolled out".

The amount of training and coaching required is driven by a multitude of factors, for example one of the respondents notes that there were projects where people were very resistant to change, because they required more coaching. There are different training needs during the agile transformation, some of the more basic ones include training new team members who were not exposed to agile before, the most important factor is rolling out training across the board and making sure new team members are trained throughout. Not only from the delivery team point of view, but also at a later stage interviewees found that transformation went much smoother when lead by experienced coach.

"Transformation must be lead by agile champions in order to adopt agile correctly"

"Make a successful case study, use champions, evangelists to sell benefits to the business".

Companies undergoing transformation should consider formal training especially for the local change leaders such as scrum-masters. When agile is adopted, formal training is critical, scrum masters play an important role - they need to be able to coach the team through the transformation.

"scrum master is very important because we are meant to be delivering value at the end of each sprint and if we can see we are not making progress and stories are piling up - it is not good. Good scrum masters can remediate this".

It is also important that transforming teams receive a similar amount of support from the organization, it is very important that each team gets the same level of attention when transforming so that they don't go off track. It has been discovered that during the course of transformation many times focus is only given to the pilot group and not much focus is given to other teams.

It has been identified, that having a transformational leader, evangelist, coach is very important and contributes greatly to overall success. Most respondents stated that having an agile coach for couple of months on the ground really helped to introduce the ins and out of a process and gave teams much more confidence in what was being introduced. This could be a help from the outside in a form of consultancy or coaches could be “grown” in house, by providing necessary training to for example scrum masters.

“we hired an agile consultant to coach the management team and they really helped us rollout agile company wide”

V. CONCLUSION AND FUTURE WORK

This chapter makes a number of conclusions based on this research study. It also proposes a number of areas for future research, and summarises the results of this research.

Conclusion

Adding to the available knowledge

The study is believed to be the first attempt to study the attitudes of senior IT staff towards waterfall to agile transformation topics by means of face to face interviews. The carefully selected research participants and detailed questioning has provided quality data that will be of interest to the intended target audience.

The limited scope of this study means that the data from the interviews cannot be considered representative of all software development professionals from various backgrounds. However collected data is sufficient to answer main research questions and has generated enough insights to make assumptions on secondary questions.

Quotations collected and reproduced in Chapter IV, provide an insight into the thinking of modern software development team members and will be of particular interest to anyone studying this subject in the future. The study also opened many avenues for further research.

Primary question: How do software development teams view agile transformation?

"younger CIO's coming from college will be expecting things to run in agile fashion."

Based on the data collected in this study, the answer to this question appears to be that views are strongly positive. The notable exception is that the managers tend to transform and capitalize on the benefits of agile much slower than technical members.

Agile methodology brings an abundance of practices and tools that include areas such as design, modelling, programming, testing, project management and quality assurance (Abbas et al. 2010). From a team point of view agile arrived with tools to track its own progress, from manager's point of view the ability to show / report on progress and monitor

every step of the way. Agile also arrived with best practices for programming, starting with the notion of extreme programming - which introduced the concept of best practices, notably code ownership, unit testing, test driven development, concept of build automation and continuous integration. All of these practices add a lot of structure to developing software solutions, therefore agile is being perceived as not a silver bullet but a good step in the right direction.

“Scrum being the most successful framework for organizing your project, *one of the main benefits is the drive towards continuous improvement, not to do what doesn't work but focus on what works*, *“focusing on infrastructure and tools.”*

Secondary questions

- Are there factors that are slowing down the adoption of agile?
- What are the barriers for agile adoption?

From all collected responses, lack of training or bad training is mentioned most times as slowing down factor for adoption of agile. This is at a team level as well as at the company level.

“Learn - as you go project managers didn't fully embrace agile, this meant that agile artifacts were not delivered sprints were not being finished, etc.”

The second most commonly mentioned factor was lack of strategic alignment. Alternatively know as managerial / leadership buy-in. Respondents mentioned that although teams were very quick to capitalise on the agile benefits a lot of the time managers and other business functions were transitioning very slowly or not transitioning at all which ultimately lead to problems and tensions downstream.

“Ultimately you do deliver better, working software, faster which can adapt quicker to market change but from a business process point of view it can make other business stakeholders uncomfortable.”

Finally, continuous pressure to deliver is found to affect transformation efforts, especially when unrealistic expectations are set by the leaders and not enough time is given to go through turbulent initial stages of the transformation.

“when you get to your work in large organisation with impending deadlines and when push come to shove - it becomes - look how do we do it? and in most cases people will turn to the process they know best.”

- Are there aspects of agile that software development teams are unhappy about?

Although the majority of respondents are very happy with new development method, it has to be noted that due to the limitation of the study this could not be a representative result and further study will be required to verify those results. However death march to delivery seems to be coming up in most interviews as something that should be addressed.

- Do software development team members understand agile / scrum topics?

Based on the data collected from participants in this study, the answer to this question appears to be a guarded “yes”.

.

- How to convince people to agile?

Based on the sample in the study technical teams do not have to be convinced to agile, often methodology is being piloted by technical teams first and then rolled out on an organizational level. However managers of various interconnected business functions often require convincing and show great amounts of resistance to change. The possible solution to this is to build an argument on a successful case study / pilot. Also understanding the fears and requirements of that particular group could provide good starting point in dismantling the resistance to change.

- What are the implications for managers?

The implications for managers found in this study could be split into two groups. Issues associated directly with the transformation, such as traditional change management aspects, as well as issues that are simply associated with agile and are not transformation specific. The main transformation related implication is achieving strategic alignment across all business functions which can fundamentally block all efforts if not achieved, examples of which are given in chapter IV. From implications not transformation specific, three can be mentioned. (1) Approach to quality assurance is still not perfect and is overlooked behind false assumption that agile has QA organised - which appears not to be true. (2) Area of vendor management appears to be much more challenging than it used to be in waterfall.

(3) Finally once successfully transformed organisations are struggling at scaling agile up for company needs. Agile in scale is very complex process and could be defeating the original simplicity argument, management of such scaled structures becomes a true challenge.

Opportunities for further research

A larger sample

The relatively small sample of this research is definitely a limitation for discovering many interesting view points. Similar study could be run to cover a more representative group of software development professionals with various experience levels and ideally various agile adaptation levels.

How to deal with death march to delivery?

One of the most interesting findings in this study is that software development teams might experience burnout because of the constant pressure to deliver. There is little information available in the literature on this, yet in times of enormous deficit of technical staff it might be cheaper to keep professionals than let them go because of the burnout issue. This can be researched form the managerial point of view - what techniques could be used so that delivery is continuous but burnout is not an issue.

How to scale agile operations?

There are some agile at scale frameworks including Scaled Agile Framework (SAFE) but there is little research evaluating their effectiveness. It would be interesting to evaluate some of those scalable frameworks and fuel decisionmakers, so that an appropriate framework can be used without having to spend time and resources figuring it out on their own.

“Agile at scale. There are frameworks SAFE and couple of others that we looked into. But it is difficult. General perception that nobody does it well”

“How do you organise teams and make them more effective in microservices environment?”

How to manage vendors in agile environment?

One of the main findings in the study is that vendor management was much easier in waterfall environment because of the ability to outsource entire stages in the production line. This is more difficult in agile and what tends to happen is that vendors simply provide “bodies” that can be “thrown” into scrum units. This poses series of challenges for client - vendor relationships - a suitable model for vendor management in agile is needed.

VI. REFERENCES:

Abbas, N., Gravell, A.M. & Wills, G.B., 2010. Using Factor Analysis to Generate Clusters of Agile Practices (A Guide for Agile Process Improvement). In *Agile Conference (AGILE), 2010*. pp. 11–20.

Ambler, S.W., 2007. Agile/lean documentation: Strategies for agile software development. *Retrieved June*, 20, p.2007.

Associates, D., 2012. *New Product Development Glossary*, John Wiley & Sons, Inc.

Avison, D.E. & Fitzgerald, G., 2006. *Information Systems Development: Methodologies, Techniques and Tools*, McGraw-Hill.

Barlow, J.B. et al., 2011. Overview and Guidance on Agile Development in Large Organizations. *Communications of the Association for Information Systems*, 29(1), p.2.

Beck, K., 2001. Principles behind the Agile Manifesto. *Agile Manifesto*. Available at: http://www.agilemanifesto.org/principles.html [Accessed February 8, 2015].

Beck, K. & Andres, C., 2004. *Extreme Programming Explained: Embrace Change*, Addison-Wesley Professional.

Benington, H.D., 1987. Production of large computer programs. In *ICSE*. pp. 299–310.

Boehm, B. & Turner, R., 2003. Using risk to balance agile and plan-driven methods. *Computer*, 36(6), pp.57–66.

Boehm, B.W., 1988. A spiral model of software development and enhancement. *Computer*, 21(5), pp.61–72.

Brooks, F.P., 1975. *The mythical man-month*, Addison-Wesley Reading, MA.

Cho, J., Jones, S. & Olsen, D., 2008. An exploratory study on factors influencing major selection. *Issues in Information Systems*, 9, pp.168–175.

Cockburn, A. & Williams, L., 2003. Agile software development: it's about feedback and change. *Computer*, 36(6), pp.0039–0043.

Cohn, M., 2009. *Succeeding with Agile: Software Development Using Scrum* 1 edition., Addison-Wesley Professional.

Dingsøyr, T. et al., 2012. A decade of agile methodologies: Towards explaining agile software development. *The Journal of systems and software*, 85(6), pp.1213–1221.

Elliott, G., 2004. *Global business information technology: an integrated systems approach*, Pearson Education.

Fruhling, A. & Vreede, G.-J.D.E., 2006. Field Experiences with eXtreme Programming: Developing an Emergency Response System. *Journal of Management Information Systems*, 22(4), pp.39–68.

Gandomani, T. et al., 2015. The impact of inadequate and dysfunctional training on Agile transformation process: A Grounded Theory study. *Information and Software Technology*, 57(0), pp.295–309.

Gandomani, T.J. et al., 2014. Exploring Facilitators of Transition and Adoption to Agile Methods: A Grounded Theory Study. *Journal of Software Maintenance and Evolution: Research and Practice*, 9(7), pp.1666–1678.

Georgiev, V. & Stefanova, K., 2014. Software Development Methodologies for Reducing Project Risks. *Economic Alternatives*, (2).

Gilb, T. & Finzi, S., 1988. *Principles of Software Engineering Management*, Addison-Wesley.

Guntamukkala, V., Joseph Wen, H. & Michael Tarn, J., 2006. An empirical study of selecting software development life cycle models. *Human Systems Management*, 25, pp.265–278.

Hickey, A.M. & Davis, A.M., 2004. A Unified Model of Requirements Elicitation. *Journal of Management Information Systems*, 20(4), pp.65–84.

Highsmith, J., 2001. History: The Agile Manifesto. *agilemanifesto.org*. Available at: http://agilemanifesto.org/history.html [Accessed February 8, 2015].

Highsmith, J. & Cockburn, A., 2001. Agile software development: the business of innovation. *Computer*, 34(9), pp.120–127.

Hughey, D., 2009. The Traditional Waterfall Approach. *University of Missouri – St. Louis.* Available at: http://www.umsl.edu/~hugheyd/is6840/waterfall.html [Accessed July 14, 2015].

Ishak, I.S. & Alias, R.A., 2005. DESIGNING A STRATEGIC INFORMATION SYSTEMS PLANNING METHODOLOGY FOR MALAYSIAN INSTITUTES OF HIGHER LEARNING (ISP-IPTA). *Issues in Information Systems*, VI(1), pp.325–331.

Johnson, J.H., 2001. Micro projects cause constant change. *The Standish Group International*. Available at: http://www.cin.br/~gmp/docs/papers/Micro%20Projects%20Cause%20Constant%20Changes.pdf.

Johnson, M., 2002. Agile methodologies: Survey results. *Victoria, Australia: Shine Technologies*. Available at: http://cf.agilealliance.org/articles/system/article/file/1121/file.pdf.

Kautz, K., Johanson, T.H. & Uldahl, A., 2014. The Perceived Impact of the Agile Development and Project Management Method Scrum on Information Systems and Software Development Productivity. *Australasian Journal of Information Systems*, 18(3). Available at: http://journal.acs.org.au/index.php/ajis/article/view/1095.

Kruchten, P., 2011. Agile's Teenage Crisis?

Laanti, M., Salo, O. & Abrahamsson, P., 2011. Agile methods rapidly replacing traditional methods at Nokia: A survey of opinions on agile transformation. *Information and Software Technology*, 53(3), pp.276–290.

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

Professional.

Larman, C. & Vodde, B., Top Ten Organizational Impediments to Large-Scale Agile Adoption. *InformIT*. Available at: http://www.informit.com/articles/article.aspx?p=1380615 [Accessed August 23, 2015].

Lee, A.S. & Hubona, G.S., 2009. A Scientific Basis for Rigor and Relevance in Information Systems Research. *MIS Quarterly*, 33(2).

Leffingwell, D., 2007. *Scaling Software Agility: Best Practices for Large Enterprises*, Pearson Education.

Lu, M.-S. & Tseng, L.-K., 2009. An integrated object-oriented approach for design and analysis of an agile manufacturing control system. *International Journal of Advanced Manufacturing Technology*, 48(9-12), pp.1107–1122.

Mar, K., 2006. An Enterprise Strategy for Introducing Agile: Part 1 - The Path to an Agile Enterprise. *Kane Mar*. Available at: http://kanemar.com/2006/02/20/an-enterprise-strategy-for-introducing-agile-part-1-the-path-to-an-agile-enterprise/ [Accessed August 3, 2015].

McConnell, S., 2004. *Code Complete* 2nd edition., Microsoft Press.

McConnell, S., 1996. Rapid Development: Taming Wild Software Schedules. Available at: http://books.google.ie/books?id=qM4Yzf8K9hwC&redir_esc=y.

Miller, K.W. & Larson, D.K., 2005. Agile software development: human values and culture. *IEEE Technology and Society Magazine*, 24(4), pp.36–42.

Morris, J., 2001. *Software industry accounting*, John Wiley & Sons.

Myers, M.D. & Avison, D., 2002. *Qualitative research in information systems*, Sage.

oxagile.com, Waterfall Software Development Model. *oxagile.com*. Available at: http://www.oxagile.com/company/blog/the-waterfall-model/ [Accessed July 14, 2015].

Paasivaara, M. & Lassenius, C., 2014. Communities of practice in a large distributed agile software development organization – Case Ericsson. *Information and Software Technology*, 56(12), pp.1556–1577.

Palmquist, M.S. et al., 2013. *Parallel Worlds: Agile and Waterfall Differences and Similarities*, DTIC Document.

Parnas, D., 2006. Agile methods and GSD: The wrong solution to an old but real problem. *Communications of the ACM*, 49(10), pp.29–29.

Parrish, A. et al., 2004. A field study of developer pairs: productivity impacts and implications. *Software, IEEE*, 21(5), pp.76–79.

Roth, W.F., 2011. *The Roots and Future of Management Theory*, CreateSpace Independent Publishing Platform.

Royce, W., 1970. Managing the Development of Large Software Systems. *Proceedings of IEEE WESCON*, 26(August), pp.1–9.

Rubinstein, D., 2007. Standish group report: There's less development chaos today. *Software Development Times*, 1.

Saunders, M., Lewis, P. & Thornhill, A., 2009. *Research Methods for Business Students*, Financial Times Prentice Hall.

Schach, S.R., 2004. *An introduction to object-oriented systems analysis and design with UML and the unified process*, McGraw-Hill/Irwin.

Schwaber, K., 2004. *Agile project management with Scrum*, Microsoft Press.

Schwaber, K. & Beedle, M., 2002. Agile Software Development with Scrum. *Pearson International Edition*. Available at: http://sutlib2.sut.ac.th/sut_contents/H129174.pdf.

Sekaran, U. & Bougie, R., 2010. *Research Methods for Business: A Skill Building Approach*, Wiley.

Shaughnessy, J., Zechmeister, E. & Zechmeister, J., 2014. *Research Methods in Psychology* 10 edition., McGraw-Hill Humanities/Social Sciences/Languages.

Sommerville, I., 2007. *Software Engineering*, Addison-Wesley.

Sommerville, I., 2010. *Software Engineering (9th Edition)* 9 edition., Addison-Wesley.

Sutherland, J., 2004. Agile development: Lessons learned from the first scrum. *Cutter Agile Project Management Advisory Service: Executive Update*, 5(20), pp.1–4.

Szalvay, V., James, M. & Mar, K., 2011. Agile Transformation Strategy. *Collab*.

Takeuchi, H. & Nonaka, I., 1986. The new new product development game. *Harvard business review*, 64(1), pp.137–146.

Taylor, A., 2000. IT projects: sink or swim. *The Computer Bulletin*, 42(1), pp.24–26.

Technologies, A., 2012. *Tutorial: The Software Development Life Cycle (SDLC)*,

Watson, R.T. et al., 1997. Key Issues in Information Systems Management: An International Perspective. *Journal of Management Information Systems*, 13(4), pp.91–115.

Williams, L., 2012. What agile teams think of agile principles. *Communications of the ACM*, 55(4), pp.71–76.

Wong, C., 1984. A Successful Software Development. *IEEE Transactions on Software Engineering*, SE-10(6), pp.714–727.

Wright, K.B., 2005. Researching Internet-Based Populations: Advantages and Disadvantages of Online Survey Research, Online Questionnaire Authoring Software Packages, and Web Survey Services. *Journal of computer-mediated communication: JCMC*, 10(3), pp.00–00.

Yu, C.H., 2001. Misconceived Relationships between Logical Positivism and Quantitative Research: An Analysis in the Framework of Ian Hacking.