Agile Software Development – Do we really calculate the costs? A multivocal literature review.

Aidan Fogarty[1], Aaron Edgeworth[1], Oishin Smith[1], Michael Dowling[1], Murat Yilmaz[1,2], Silvana Togneri MacMahon[1,2], Paul Clarke[1,2]

[1] School of Computing, Dublin City University, Ireland
[2] Lero – the Science Foundation Ireland Research Centre for Software

{aidan.fogarty23,aaron.edgeworth3,oishin.smith25,michael.dowling35}@mail.dcu.ie, murat.yilmaz@dcu.ie, silvana.macmahon@dcu.ie, paul.m.clarke@dcu.ie

Abstract. Agile software development methods, in their various different forms, have become the basis for most software projects in today's world. The methodology is present in almost all organisations today. However, despite the popularity, failure rates in software projects remain high. This paper identifies why agile methodologies have become so successful. In addition, the paper discusses certain factors that may often be overlooked in organisations that have adopted agile methods, such as rework, maintainability, adoption, turnover rates and the potential costs associated with each. The research carried out was a multivocal literature review (MLR). Multiple white and grey literature which was deemed to be relevant was selected. 32 contributions from white literature were selected for use in the review as well as 8 from grey literature sources. We find that while agile has many advantages, organisations may overlook the potential downsides of using an agile methodology. If not managed or implemented correctly, organisations risk taking on more hidden and expensive costs, for example in relation to rework. It is important that organisations are sufficiently trained in agile methods in order to succeed.

Keywords: Agile Software Development, Rework, Agile Costs, Agile Success.

1. Introduction

Many agile software development practices may predate the agile manifesto, but nevertheless, since its inception in 2001, agile software development (or for short, agile) has grown in popularity [43]. Many common metrics used to measure the success of a project, such as Business Value Delivered, Customer Satisfaction and Earned Value, show an improvement when agile systems are implemented [1]. From a business standpoint these are positive and in line with what the organization is seeking to improve. However, as agile has become more ubiquitous, some organisations risk overlooking the potential costs that agile may bring.

It is undeniable that agile has proven to be an effective way to manage software projects, but this may contribute to it simplified reputation as a "one size fits all" solution to software process, even though there are many situational factors that affect the development process [44], and a need to continually adapt it has been shown to be beneficial for business success [56]. More generally, it has been suggested that agile as a concept, for example in manufacturing, should not be considered as a one-size-fit-all solution [2]. Indeed, variation in software development situational contexts also affects the software process [51-54] and the inevitable change within given contexts presents as a constant challenge to software developers [55]. Though it has been shown that agile delivers an overall improvement in project success, precisely what defines "success" in software companies is subject to variation [45]. Often companies will adopt agile in the hope of it being a general solution, but they may not adapt their organisations to allow agile to work for them [3]. The goal of this research is to investigate whether agile methodologies incur certain hidden costs that may not be accounted for (or calculated). Agile projects should be implemented with a robust level of understanding and discipline, without which they run the risk of losing time, reducing quality and creating confusion for the development team [4]. Furthermore, the way in which agile affects the culture and traditions of a company is difficult to measure, but research has shown that this has been one of the most overlooked and difficult problems to correct [5]. The agile approach requires an excellent working understanding of its processes by everyone involved, including the customers. We suggest that companies that view this development process simply as a quick way to fix production or delivery problems face hidden dangers that could result in compounding these issues. To examine this important space, we investigate what it means for agile projects to be successful, and we seek to identify aspects of agile that can be troublesome and that companies may inadvertently overlook.

Section 2 of this paper details the related agile software development literature, with Section 3 examining the concept of success in agile software development, and Section 4 investigating the challenges associated with agile. Section 5 presents research limitations and Section 6 concludes the paper and identified possible future research directions.

2. Related Literature

Four researchers were responsible for conducting the review over a 7 week period as part of an undergraduate assignment based Dublin city University. A multivocal literature review (MLR) [46] was employed, and it involved the use of various white (such as academic papers) and grey (such as blogs, newspapers, websites) literature. Careful consideration was taken when choosing grey literature and it was almost always used in combination with white literature to corroborate a point. The team took part in weekly meetings where any problems and questions were discussed with the module lecturer.

2.1. Research Questions

The data sources and search strategies that were thoroughly examined determined the research questions in this paper. The aim of this paper is to provide an answer to the two following research questions:

- What is “success” for agile software projects?
- What are the factors that organisations may overlook when using or adopting agile?

These two research questions were chosen as we believe that they effectively target the goal of the paper. From these research questions, the team aimed to provide information on topics such as rework, agile adoption, turnover and maintainability.

2.2. Data Sources and Search Strategy

In the first meeting, key search strings were discussed, and the literature review was then broken down into several smaller steps. First was to determine the key strings which were used in determining relevant academic papers and grey literature. Key strings such as “Agile success”, “agile rework costs”, “questioning agile”, “rework agile” and “failure in agile” were used and careful examination of the different literature was performed before the relevant material was extracted. Google search was utilised to find grey literature. The academic papers were found through Google Scholar and the digital libraries of publications include ScienceDirect, IEEE and ACM which were used to find the academic papers used in this paper.

2.3. Inclusion/Exclusion Criteria

Prior to conducting the bulk of the research, a class was held on how to correctly carry out an

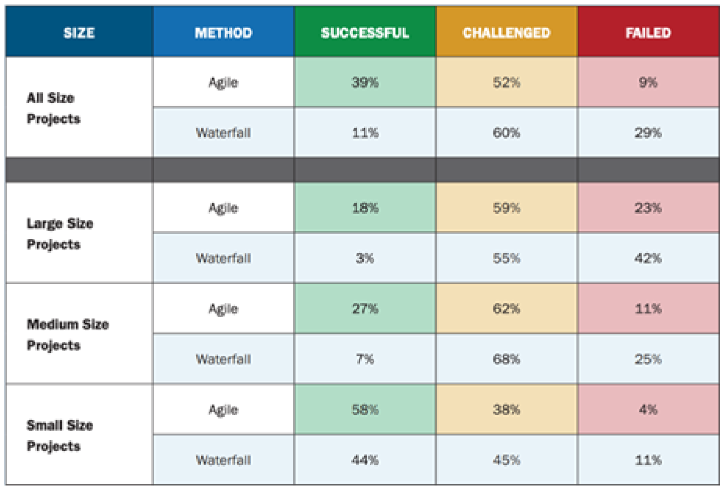
MLR. Criteria for including and excluding certain literature was discussed. The title and abstract of the returned papers were briefly analysed to determine relevancy. Papers were considered relevant if they provided an interesting viewpoint on one of the topics mentioned above as well as being available in English. For grey literature, further criteria needed to be accounted for such as the validity of the source or author. Using this process, just over 40 pieces of literature were deemed relevant. The result of our findings are discussed in the following sections.

3. Agile Success

Traditional software development is plan-driven in which work begins by documenting a complete set of requirements, followed by high-level design documents before the coding and testing has begun. The emphasis on documenting comprehensive set of requirements and a design up front have while beneficial to certain aspects of software projects, they can also manifest as "a source of major contention, rework, and delay at high-change levels" [6]. It is these challenges that caused agile methods to become so popular [7], possessing as they do the ability to enable organisations to be flexible in their treatment of requirements and focused on enabling businesses to rapidly respond to changes [8]. There are also claims that agile methods achieve success as the primary focus is delivering business value while reducing costs [9], and while some costs are certainly eliminated (such as documentation costs), the authors of this work suggest that perhaps there has been insufficient treatment of other less obvious costs that are not presently subject to measurement. Prior to agile, reported project success rates were far lower than today, for example the 1994 Standish Group Chaos report, stated that software project success rates were only 16.2%. However, from this it should not be concluded that agile software development has improved project success rates, as many other technological innovations have occurred in the intervening years, and indeed, measures of project success are also subject to change [46].

3.1. Defining Success in Agile

In order to understand why agile has become so successful, it is important to define success. Defining project success has been a topic of discussion for many researchers and practitioners. Often, the definition of success comes back to the triple constraint [7]. Using this approach, a project is considered successful when it is delivered on time, within budget and on target or according to requirements. This view of success may be seen as project management success. However, this may be an over-simplified view of success, and thus it is adapted to include other perspectives, for example customer satisfaction [10]. The 2015 Standish Group Chaos report defines success as being on time, within budget and producing a satisfactory result, marking a project that passes all three as successful, challenged if one of the three measures fails and lastly, failure if the project was cancelled [11] (the report gathered data from over 10,000 projects). The report examines difference in success between agile and waterfall methods, finding that in general agile is more successful (ref. Fig. 1). However, the difference is most pronounced in medium to large size projects.



SIZE	METHOD	SUCCESSFUL	CHALLENGED	FAILED
All Size Projects	Agile	39%	52%	9%
	Waterfall	11%	60%	29%
Large Size Projects	Agile	18%	59%	23%
	Waterfall	3%	55%	42%
Medium Size Projects	Agile	27%	62%	11%
	Waterfall	7%	68%	25%
Small Size Projects	Agile	58%	38%	4%
	Waterfall	44%	45%	11%

Fig. 1 - Chaos Report 2015: Success in Agile vs Waterfall [11]

While the measurements above for defining success are important, they do not take account of certain qualities of agile such as technical excellence, process improvement and sustainable development. The Art Of Agile Development outlines that success should be considered as a union of organisational success, technical success and personal success (ref. Figure 2). The book highlights that all three are necessary. Personal success is required to ensure employees remain motivated, technical success to ensure the software created is maintainable and organisational success to ensure that the project is delivering value [9].

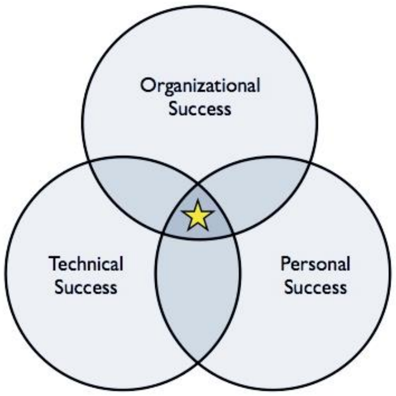


Fig. 2 – Success in Agile Software Development [9]

Agile, however, is not a silver bullet for software project success [9], with multiple grey literature sources discussing many underlying problems that have appeared in agile recently [12, 13, 14]. Kent McDonald discusses how organisations are moving to agile as they believe they will be at a disadvantage if not seen to do so. Ron Jefferies outlines that when agile is adopted poorly, developers can suffer with having less time to do the work, with increasing demand to build software faster. This can result in more defects, slower progress and developers leaving an organisation. A survey examining critical agile success factors found the following to be important: a correct delivery strategy, proper practice of agile software engineering techniques, a highly skilled team, a solid team management process, a team oriented environment and strong customer involvement [15]. Customer involvement may be particularly important, and that without it, agile projects may experience pressure to over-commit, loss of productivity, significant rework and business loss [16]. If we consider that customer collaboration, sometimes extending to the on-site presence of a customer, is highly encouraged in agile software development, we be alert to the impact of reduced customer collaboration in agile settings. Afterall, perhaps not all clients will be able to locate on-site – full time or even part time - during projects.

The 2018 State of Agile report notes that 97% of the respondents utilise agile software development methodologies, with Scrum and SAFe being the most popular agile implementations [1]. The report highlights the reasons for adopting agile methods as accelerating software delivery, managing changing priorities and increasing productivity. It is important to note that the report also mentions that 83% of the respondents were below a high level of competency with agile practices, which we suggest is cause for concern in relation to long term software project and system sustainability. A 2017 study, “An Agile Agenda”, examined 300 companies across the UK and US, finding that agile “has been adopted so enthusiastically that it is now being stretched beyond its limits” [19]. Of course, the rise of frameworks such as SAFe may work to reduce this impact, but they might be more heavyweight agile frameworks, requiring as they do an increases in process and not necessarily at a rapid delivery cadence. In this sense, scaled agile frameworks such as SAFe might, we suggest, could be classified as *semi-agile* frameworks, falling as they do between traditional lifecycle models such as the waterfall, and more recent agile themed approaches such as continuous software engineering [48].

4. Potential Challenges with Agile

In this section, we aim to outline potential challenges that may be overlooked when using agile methods.

4.1. Minimal Documentation

One of the primary values of the Agile manifesto is “working software over comprehensive documentation”, which encourages developers to focus more on delivering software rather than spending time on documentation. Often, the code is considered the documentation and as change is guaranteed in agile, spending time on documentation can be considered effort wasted [17]. Traditional methods such as Waterfall made extensive use of up front design and documentation. Agile, in contrast, aims to reduce the large up front design cost as this is considered one source of the high rework costs that may associated with traditional approaches [6]. Note that it is not just the initial creation of the big upfront design that is considered expensive, it is also the fact that the design must be revisited and modified as changes arise. Agile does not completely disregard documentation, it only emphasises that it is more important to apply knowledge rather than documenting it [18].

However, there are genuine concerns regarding the absence of appropriate levels of documentation on agile projects, especially as it has been observed that “44% of Agile projects that fail do so because of a lack of documentation” [19]. Due to agile methodologies focus on minimal documentation, it means that the primary source of knowledge within the methodology is tacit. This approach works in favour of agile methods if the team’s tacit knowledge is sufficient for the project life cycle, however, there is also a risk that the team may make critical mistakes due to the shortfalls in tacit knowledge [6]. In small agile teams or organisations, tacit knowledge may suffice, but in larger agile organisations, formal documentation is required for inter-team communication and coordination [20].

Another interesting point refers to one of the 12 principles of agile: Agile processes promote sustainable development. With relation to sustaining quality performance in software development, tacit knowledge transformation to organisational knowledge is essential. However, due to agile’s fast paced environment, it can be challenging to record tacit knowledge in documentation [18]. Knowledge sharing techniques have been used in agile approaches such as pair programming, pair rotation as well scrum daily stand-up meetings which include the entire team [21]. But the hidden value of these practices in enabling a cohesive overall agile strategy may not be appreciated across the board, for example some may consider pair programming too expensive, even though Kent Beck in prescribing XP [49] advocated that all practices must be implemented, and not just some a la carte selection.

Rudimentary and partial measures of success may only measure up until the project is handed over to the maintenance stage, and therefore, further studies have examined maintainability in agile environments. One study, involving 18 organisations, observed that agile may be effective in the short term as there is less focus on documentation and more on productivity, but long term there may be problems [22]. For example, the study results showed that 50% of software engineers lost track of projects they are working on, 66% also highlighting that the loss of key engineers would be a major issue, leading to increased costs and decreased productivity. This is a good example of the challenges raised by absences of formal knowledge reification (for example in the form of documentation describing the product), whereby developers that wishes to extend the design may be puzzled by the current implementation [17].

4.2. Rework

Rework can be described as redoing a process or activity that was initially implemented incorrectly, often due to changing requirements, and it can directly impact the performance and profits of an organisation, perhaps depleting 40% to 70% of a project's budget [23]. We suggest that there are genuine challenges regarding the classification of rework, especially concerning avoidable and unavoidable rework. However, literature shows that some rework is unavoidable in software development as software is an evolving process. Multiple research efforts have described the different types of rework in software development, which may be broken down into three categories [24]:

- Evolutionary: Usually rework that occurs in response to a change in user requirements (unavoidable).
- Retrospective: Rework that occurs when developers knowingly exclude needs required by users. Thus, requiring the developer to add the needs in the next version (avoidable).
- Corrective: Often the most common type of rework. It involves fixing defects that were added in previous versions of the code (sometimes avoidable).

It is important that organisations identify the root causes of the different types of rework, as it can help with improving productivity, developer morale and customer satisfaction [24]. Reasons for rework vary within different organisations, and we suggest that some interpretation may be required in classifying rework, and not all participants may agree on the classification reached for a given item. Common reasons for rework are often due to poor communication, ambiguous requirements, inadequate testing and a lack of documentation [6]. The main causes can also be represented using the "fish-bone" model as seen in Fig. 3 [23].

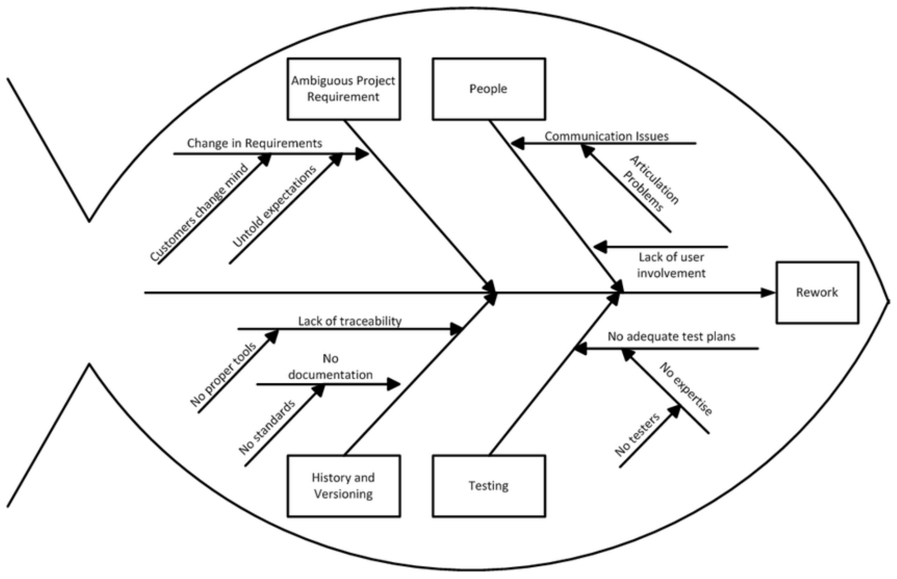


Fig. 3 – Fish-bone Model [23]

While some consider refactoring to be considered a form of rework [24], rework is considered by many as a negative process, whereas refactoring is positive [7]. Refactoring helps to improve code understanding, improve maintainability, easier to test and allows for easier defect detection [5]. Of course, with agile based development, rework is inevitable [24] as the fundamental principle is that of welcoming change at any time during the development lifecycle [25]. Agile principles encourage better collaboration with customers, technical excellence and releasing working software frequently, all of which target the causes of rework. Owing to the increased collaboration between users and developers in agile, the amount of rework can be reduced. A study which examined the socio-technical aspects of agile, showed rework was reduced because of frequent communication between team members [26]. Agile also promotes the use of an automated build and test system which alerts developers of errors early in the development lifecycle when new features are implemented [24]. A study which examined the effectiveness of agile methodologies versus a waterfall methodology found that the total amount of rework done was less in the agile approach (ref. Fig. 4) [27]. Our analysis suggests that most rework research focuses on minimizing the amount of rework in a given software project [23], and for an aspect of cost volatility as large as rework can potentially be, there was little concrete or substantial contemporary published material dedicated to measuring the cost of rework in agile settings.

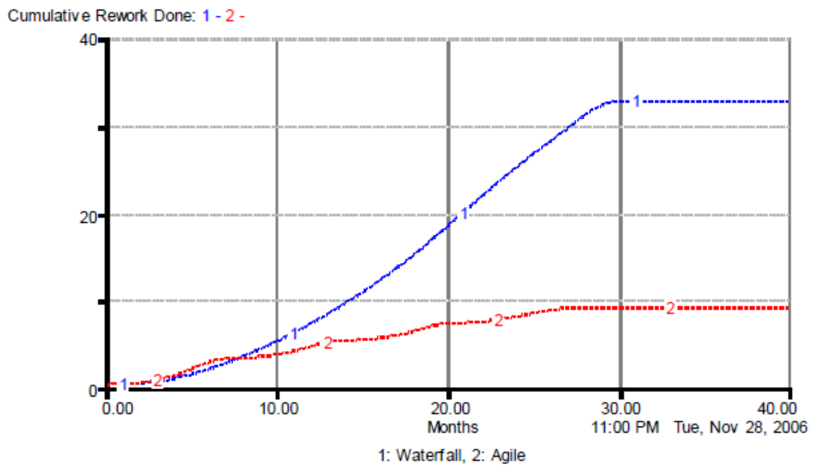


Fig. 3 – Rework in Agile Versus Waterfall [27]

It is accepted that 10% to 20% of total effort in agile is spent on rework [24], yet we are also told that it accounts for 40% to 70% of a software project's budget [23]. All types of rework should be collected and analysed if organisations wish to find the root causes. However, software developers may avoid noting rework as it can take away from perceptions of productivity and their organisations may tie rework to performance reviews. Organisations should encourage developers to speak about rework without the worry of it affecting performance reviews [24], perhaps rework reduction should be a performance parameter. It is also interesting to note the relationship of rework with the product backlog which is often used in agile methodologies such as scrum. One study focused on how product backlog changes are managed in Scrum projects [28]. In this study, it states that as a requirement changes, over 50% of practitioners write the updated requirement as a new requirement, instead of recording the change reason or motivation. This may lead to significant untracked rework and it appears that there is ripe potential for both research and measurement in this space.

4.3. Organisational Change

All software development is complex [49] and we can therefore expect that it requires detailed planning and analysis. It is therefore not surprising to discover that up to 34% of agile projects that fail are reported to do so because of a lack of planning for the project and its methods [29]. We therefore see that while agile may advocate more so-called *lightweight* developer-oriented planning, monitoring and control, this alternative to traditional project management-led planning does not necessarily yield improved project success. In the UK, over £30 billion is estimated to have been wasted on unsuccessful agile projects in 2017 [19]. These are the costs that companies hope to reduce when adopting agile but which may manifest due to its adoption.

One example of the process change that may be required relates to software architecture, with 68% of CIOs interviewed in one survey claiming that agile teams require more architects, with hiring and training of architects seen as necessary if a company wishes to scale-up using agile methods [19]. This can be considered to be a very interesting observation, as software architects have a broader view of the software system and its overall design, and an insistence of having higher numbers of architects in agile environments may reduce the impact of design decay. Perhaps, indeed, agile settings prefer or benefit from increased architectural knowledge among heretofore standard software engineers. Agile architects are expected to mentor and be directly involved with the development process and future planning as requirements change [30].

When transitioning to agile, changes in the organization's culture, structure and management can be expected. Agile is highly dependent on social interaction between team members, and a shift in organizational structure can impact the work culture of an organization. Culture has a powerful yet intangible effect on the development process, and the power shift from management to the developers which can cause discontent and disruption for a project and its members [31]. In the 9th Annual State of Agile Survey, 42% of respondents claimed one of the leading reasons for a failed agile project was company philosophy or culture at odds with core agile values, and 36% stated a lack of support for cultural transition [1]. Failed projects are a continuing problem in the IT industry, 66% of projects partially or totally fail [11], with the worldwide estimated cost of IT project failure reported to be 6.18 Trillion USD [32]. If a company is unable or unwilling to account for the cultural change that agile requires it will cost the company time, money and workers that become dissatisfied or unable to work in a conflicted system. Agile requires a shift from a "command-and-control" process to a "leadership-and-collaboration" process [31], and this might not be easily achieved in various settings.

Scrum is an agile project management approach that emphasises the interaction and cooperation of many different roles on a team, with the welcomed involvement of the client to the process. However, the various meetings required in agile, including those with client involvement and potentially other stakeholders also, can raise challenges related to differing priorities [33]. Face to face communication is regarded as the most effective form of communication and is encouraged as the primary form of communication [25]. This is typically facilitated by the daily stand-up meetings, however, as software development has become increasingly globalized in recent years, organisations have had to figure out how to accommodate teams with members that are living in different time zones. Success rates for projects with members that are located far apart drops significantly when compared to those that are co-located [34]. Large time zone differences, national and religious holidays, language barriers all can act as an impediment for agile practices [35]. If these practical realities are not properly accommodated, organisations could experience additional costs in terms of time, money and quality. The collaborative nature of agile methods means that it is at increased risk [5].

4.4. Job Satisfaction

Some research suggests that most of what we know about job satisfaction in agile is anecdotal and to be careful about the findings [36]. Job dissatisfaction can have a huge impact on staff turnover rates and stress, and high turnover rates are concerning as they can have a considerable economic effect on an organisation and its employees [36]. Where turnover is an issue, and the software development business is not immune to this phenomenon, training new replacement employees to an efficient standard is important. However training new employees can be costly especially in small software development companies [18].

Many different factors may affect job satisfaction in agile environments, with certain aspects shown to have a positive effect, including agile project management methods (such as daily stand-ups, retrospectives, iterative planning) and development practices (such as automated unit testing, pair programming, continuous integration and refactoring) [37]. Studies have proven that higher perceptions of job characteristics of a) feedback, b) task significance, c) skill variety, d) job autonomy, e) ability to complete a whole task are positively related to job satisfaction [36, 37, 40]. What can infer from these works that factors such as job autonomy, skill variety, task significance, feedback and the ability to complete a whole task are significant factors to support developer satisfaction [36, 37].

Comparing the different levels of stress that an employee may be experiencing depending on the methodology used may provide useful insights into how agile affects stress. A study performed by Mannaro et al, studied factors affecting satisfaction in software teams and specifically examined the relationship between stress and the software methodology adoption, finding that agile methods were associated with reduced stress levels [38, 39]. A similar study conducted in Switzerland performed an online survey querying managers and developers about the usage of development methods and practices such as agile, and examined possible associations with stress (ref. Fig.5) [39]. In this study, managers and developers were asked to rate how agile had influenced their stress at work. They responded on a scale from 1 (not very stressed) to 5 (very stressed), and findings suggest that managers seem to be less stressed in agile environments compared to the developers who tend towards a neutral observed stress effect. This is perhaps not surprising as managers have less direct responsibility.

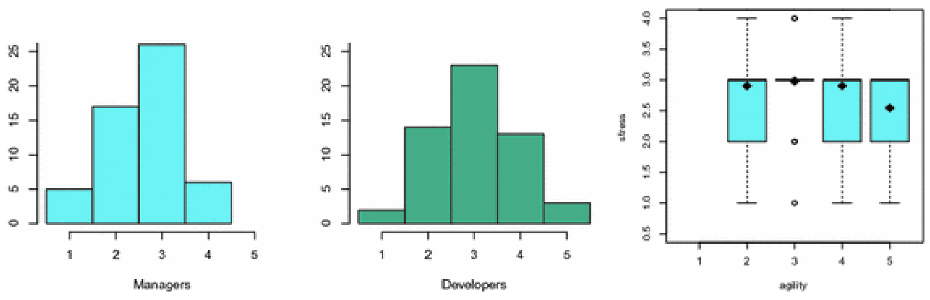


Fig. 5 – Levels of stress in agile, managers vs developers [39]

4.5. Knowledge and Employee Turnover in Agile

Knowledge loss is a serious issue for any development team, the smaller the team, the greater the impact of the staff turnover rates [18]. This impact can manifest in many ways, for example on team productivity and quality. There are many different forms of knowledge loss that can happen within a company, for example, knowledge hoarding, staff turnover and knowledge loss due to the pace of technology. Knowledge hoarding is where employees want to keep their knowledge hidden for inter-personal reasons or where it may be endemic in the organisational culture, however building a knowledge sharing environment is very important for any business. [18]

The pace at which technology changes is a good example of the challenge for knowledge sharing in software development companies. Developers may find it difficult to update their knowledge while trying to meet all their deadlines and because of that they will not get around to sharing knowledge with their colleagues [18]. Due to the agile methodologies emphasis on minimal documentation, employee turnover rates are perhaps especially concerning to a company [18, 25, 42]. If the original development team cannot be maintained, knowledge about the software will be lost and extra resources will go into training new employees [18, 25]. These costs might not be insignificant the knowledge intensive software development ecosystem where employee turnover has been a perennial challenge.

The opportunity to work on interesting projects, being able to influence decisions and the relationship with their users are statistical factors in satisfaction [36]. One study found that job dissatisfaction can lead to high voluntary staff turnover [36], with turnover costing about 20 percent of all manpower expenses [41]; other studies have estimated turnover costs to be as much as 70-200% [36]. Software development firms will understandably want to avoid high turnover rates and reports agile positively affects employee satisfaction, might be aligned with improved strategic outcomes including staff retention [36, 38]. It should be noted that calculating the true cost of replacing an experienced worked is a non-trivial exercise in its own right [41].

Agile adapts to employee turnover rates by introducing two knowledge sharing approaches, which create strong enthusiasm in software engineering [18]. Pair programming is the first one and it allows knowledge sharing between two developers. Pair rotation is the second knowledge sharing technique and it is used to break the ice between software development team members, however, the downsides to pair programming and pair rotation are that they double the manpower and cost on a given problem [18]. So while there are mechanisms in place to support knowledge sharing in agile settings, they are not free and perhaps not consistently adopted in industry, and the very concept of knowledge retention in software engineering is a broad and complex field [50].

5. Limitations of research

The field of agile software development is vast, it is employed around the world and is documented extensively across both academic and grey literature. To investigate the true costs of agile software development is to some extent an impossible task, especially given the extent of situational variation that can arise in software development [44]. In an effort to better understand the costs associated with agile software development, four primary researchers investigated different costs themes in agile software development over a seven week period. While this was sufficient to gain some insight into the phenomena of interest, it can be considered quite limited in terms of a full evaluation of all of the material that has been documented on these topics.

Furthermore, the four primary researchers were four final year undergraduate students in the B.Sc. in Computing Applications (Hons.) based in Dublin City University, Ireland. These researchers are not formally trained in research, and it was their first opportunity to conduct academic research. To offset this, the researchers received training on performing multivocal literature reviews, and were furthermore supported weekly on the process of performing robust academic research.

6. Conclusion & Future Work

Awareness of the strengths and successes of agile are spoken about at large in the software industry. When implemented into an organisation's process effectively, the benefits to customer and employee satisfaction can be substantial; with product improvement and cost reductions reported across both academic and grey literature. However, the ease at which agile can be effectively adopted can be overestimated, and the concept of “success” may be overstated and over simplified, giving a false impression that agile is the standard, simple fix to any project regardless of the business context. Agile is a way of thinking and a culture that must be clearly explained to all those who are involved and creating awareness of this point is massively important. Without mature familiarity with the agile process and all its supporting practices, both internal and external disruption can arise which can directly or indirectly affect the product and its clients. Those responsible for strategic decision making in software development firms should invest in understanding agile more deeply, the ultimate realisation from which may be that agile is more complex than some simple scaffolding provided in some agile method. It is in fact a cocktail of interrelated practices and techniques that drive an organisational culture and which can prove difficult to master.

Clear and direct communication are essential to an effective agile strategy, and issues with scaling and distantly located team members can be exacerbated if they are not carefully accommodated in the agile process. Often, management can look to agile to fix general issues with a project at the time of adoption, but as an organization has more success, they risk scaling without properly preparing their process.

There are a significant number of intangibles that are difficult to assess when it comes to examining project success in agile. With work culture playing a large role in the level of an employee's job satisfaction, the need for a culture shift can cause dissatisfaction for workers. Our research found that the agile paradigm shifts some of the power from management and grants it to the developers, and that this can raise developer job satisfaction. Although agile may increase perceived work pressure among developers, overall their engagement with their role can increase and stress levels can be reduced. Somehow, the increased pressure that comes with increased empowerment may in fact serve to reduce overall stress load. But our findings here are not conclusive and given the role of job satisfaction in employee retention, we recommend further attention to understanding the reasons for high staff attrition and if these may be related to agile adoption. Other reasons for this attrition clearly exist, for example the high demand on skilled software developers can drive wage inflation. But not every employee leaves a company for financial reasons, and with knowledge workers in particular, job satisfaction may be a key factor to consider. And a key concern in job satisfaction relates to the work processes, and therefore, this should be examined further through the lens of agile software development.

In relation to rework, our findings suggest that it is a complicated concept. Put simply, what one person considers to be rework, another person may classify as requirements discovery through an evolutionary feedback process. This is perhaps the most important finding from our research: it seems that there is very limited research into the costs of rework in agile settings. We also observe what might be considered a dangerous practice of introducing wholly new product backlog features when eliciting user feedback on already-implemented features. Left unchecked, this practice can disguise rework as new product backlog items, and it could be the source of cost hemorrhage in some settings. It may be that effective agile implementation should insist on linking new product backlog items to earlier backlog items to police against runaway rework costs. But as much of this appears unmeasured – or at least unreported – today, we cannot know the true position. This, we suggest, should be a major concern for software companies.

While our research brought forward no dedicated studies on the impact of minimal documentation in agile, multiple studies have suggested that a lack of documentation may cause issues. Agile may be beneficial for short term goals, but if ineffectively implemented, it may exacerbate long term maintainability costs. This point cannot be overemphasised and it plays back into the long-established guidance not to take an a la carte approach to agile software development practices. Supporting practices such as refactoring and pair programming are important ingredients in building a long-term sustainable product and team. Without an emphasis on software design as supported through refactoring, products can become very expensive to maintain, and without practices for training and knowledge sharing such as pair programming, companies can silently accumulate dependencies on transient workers , all the while perhaps also working against a truly agile philosophy. And finally, let us not forget that while the Agile Manifesto values working software over comprehensive documentation, and individuals and interactions over processes and tools, it also recognises that there is *value* in documentation and processes. This is perhaps the small print that some businesses miss.

Acknowledgements. This work was supported, in part, by Science Foundation Ireland grant 13/RC/2094 and co-funded under the European Regional Development Fund through the Southern & Eastern Regional Operational Programme to Lero - the Irish Software Research Centre (www.lero.ie)

References

1. Annual State of Agile Survey, www.stateofagile.com/ [accessed 10th June 2020]
2. Shewchuk, John P. "Agile manufacturing: one size does not fit all." Strategic management of the manufacturing value chain. Springer, Boston, MA, 1998. 143-150.
3. Agile budgeting: How much will it cost?, Agilest.org [accessed 10th June 2020]
4. Stoica, M., Marinela, M., and Bogdan, G-M. "Software Development: Agile vs. Traditional." Informatica Economica 17.4 (2013).
5. Korkala, Mikko, and Frank Maurer. "Waste identification as the means for improving communication in globally distributed agile software development." Journal of Systems and Software Vol. 95 (2014): 122-140.
6. Boehm, Barry. "Get ready for agile methods, with care." Computer, 35(1) (2002): 64-69.
7. Serrador, Pedro, and Jeffrey K. Pinto. "Does Agile work?—A quantitative analysis of agile project success." International Journal of Project Management 33(5) (2015): 1040-1051.
8. Agile Software Development Ecosystems, Jim Highsmith, Addison Wesley, Boston. 2002.
9. Shore, James. The Art of Agile Development: Pragmatic guide to agile software development. O'Reilly Media, Inc., CA, US. 2007.
10. Van Der Westhuizen, Danie, and Edmond P. Fitzgerald. "Defining and measuring project success." Proceedings of the European Conference on IS Management, Leadership and Governance 2005. Academic Conferences Limited, 2005.
11. Standish Group. 2015 Chaos Report.
12. Agile Q&A: Why do Organizations Adopt Agile? Kent McDonald. www.agilealliance.org/why-do-organizations-adopt-agile/ [accessed 10th June 2020]
13. Developers Should Abandon Agile, Ron Jefferies. https://ronjeffries.com/articles/018-01ff/abandon-1/ [accessed 10th June 2020]
14. Questioning Agile Dogma, Dave Nicolette, www.leadingagile.com/2019/02/questioning-agile-dogma/ [accessed 10th June 2020]
15. Chow, Tsun, and Dac-Buu Cao. "A survey study of critical success factors in agile software projects." Journal of systems and software 81(6) (2008): 961-971.
16. Hoda, Rashina, James Noble, and Stuart Marshall. "The impact of inadequate customer collaboration on self-organizing Agile teams." Information and Software Technology 53(5) (2011): 521-534.
17. Knippers, Daniël. "Agile Software Development and Maintainability." 15th Twente Student Conf. (2011)
18. Ersoy, I. Burak, and Ahmed M. Mahdy. "Agile knowledge sharing." International Journal of Software Engineering (IJSE) 6(1) (2015): 1-15.
19. An Agile Agenda, 6Point6 Technology Services, https://6point6.co.uk/insights/an-agile-agenda/ April 2017. [accessed 10th June 2020]
20. Dikert, Kim, Maria Paasivaara, and Casper Lassenius. "Challenges and success factors for large-scale agile transformations: A systematic literature review." Journal of Systems and Software Vol. 119 (2016): 87-108.
21. Chau, Thomas, and Frank Maurer. "Knowledge sharing in agile software teams." Logic versus approximation. Springer, Berlin, Heidelberg, 2004, pp.173-183.
22. Kajko-Mattsson, Mira. "Problems in agile trenches." Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement. 2008.
23. Ramdoo, Vimla, and G. Huzooree. "Strategies to reduce rework in software development on an organisation in mauritius." International Journal of Software Engineering & Applications

6(5) (2015): 9-20.

24. R. E. Fairley and M. J. Willshire, "Iterative rework: the good, the bad, and the ugly," in Computer, vol. 38, no. 9, pp. 34-41, Sept. 2005.
25. The Agile Manifesto, Agile Alliance, 2001. https://agilemanifesto.org/ [accessed 10th June 2020]
26. Inayat, Irum, Sabrina Marczak, and Siti Salwah Salim. "Studying relevant socio-technical aspects of requirements-driven collaboration in agile teams." 2013 3rd International Workshop on Empirical Requirements Engineering (EmpiRE). IEEE, 2013.
27. Chichakly, Karim. "Modeling Agile Development: When Is It Effective?." Proc Int'l Conf. of the System Dynamics Society 2007.
28. Alsalemi, Ahmed Mubark, and Eng-Thiam Yeoh. "A survey on product backlog change management and requirement traceability in agile (Scrum)." 2015 9th Malaysian Software Engineering Conference (MySEC). IEEE, 2015.
29. Agile development, an 'IT fad' that risks iterative failure, Cliff Saran, May 2017. https://www.computerweekly.com/news/450418205/Agile-development-an-IT-fad-that-risks-iterative-failure [accessed 10th June 2020]
30. The role of the Agile Architect, Andrew Johnston. https://www.agilearchitect.org//agile/role.htm [accessed 10th June 2020]
31. Nerur, Sridhar, RadhaKanta Mahapatra, and George Mangalaraj. "Challenges of migrating to agile methodologies." Communications of the ACM 48(5) (2005): 72-78.
32. Worldwide cost of IT failure, Michael Krigsman, December 2009.
33. Coram, Michael, and Shawn Bohner. "The impact of agile methods on software project management." 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'05). IEEE, 2005.
34. Geographically Distributed Agile Teams, PMI Disciplined Agile. https://www.pmi.org/disciplined-agile/agility-at-scale/tactical-agility-at-scale/geographically-distributed-agile-teams [accessed 10th June 2020]
35. Kajko-Mattsson, Mira, Gayane Azizyan, and Miganoush Katrin Magarian. "Classes of distributed agile development problems." 2010 Agile Conference. IEEE, 2010.
36. Melnik, Grigori, and Frank Maurer. "Comparative analysis of job satisfaction in agile and non-agile software development teams." International conference on extreme programming and agile processes in software engineering. Springer, Berlin, Heidelberg, 2006.
37. Tripp, John F., Cindy Riemenschneider, and Jason B. Thatcher. "Job satisfaction in agile development teams: Agile development as work redesign." Journal of the Association for Information Systems 17(4) (2016): 267.
38. Meier, Andreas, et al. "Stress in agile software development: Practices and outcomes." International Conference on Agile Software Development. Springer, Cham, 2018.
39. Mannaro K., Melis M., Marchesi M. (2004) Empirical Analysis on the Satisfaction of IT Employees Comparing XP Practices with Other Software Development Methodologies. In: Eckstein J., Baumeister H. (eds) Extreme Programming and Agile Processes in Software Engineering. XP 2004. Lecture Notes in Computer Science, vol 3092. Springer, Berlin, Heidelberg.
40. Tessem, Bjørnar, and Frank Maurer. "Job satisfaction and motivation in a large agile team." International Conference on Extreme Programming and Agile Processes in Software Engineering. Springer, Berlin, Heidelberg, 2007.
41. DeMarco, Tom, and Tim Lister. Peopleware: productive projects and teams. Addison-Wesley,

2013, pp 17, 118.

42. Documentation in Agile: How Much and When to Write It? InfoQ, January 2014. https://www.infoq.com/news/2014/01/documentation-agile-how-much/ [accessed 10th June 2020]
43. Clarke, P., O'Connor, R.V., Yilmaz, M.: In Search of the Origins and Enduring Impact of Agile Software Development. ACM proceedings of the International Conference of Software and System Processes (ICSSP 2018), Gothenburg, Sweden. 26-27 May 2018. pp.142-146.
44. Clarke, P., O'Connor, R.V.: The situational factors that affect the software development process: Towards a comprehensive reference framework, Information and Software Technology, Vol. 54(5), May 2012, pp.433-447.
45. Clarke, P., O'Connor, R.V.: The Meaning of Success for Software SMEs: An Holistic Scorecard Approach, In: Proceedings of the 18th European System and Software Process Improvement and Innovation Conference (EuroSPI 2011), CCIS Vol. 172, pp.72-83. Heidelberg, Germany: Springer-Verlag.
46. Vahid Garousi, Michael Felderer, Mika V. Mäntylä, "Guidelines for including grey literature and conducting multivocal literature reviews in software engineering", Elsevier Journal of Information and Software Technology, volume 106, pp. 101-121, February 2019
47. O'Connor, R.V., Elger, P., Clarke, P.: Continuous Software Engineering - A Microservices Architecture Perspective. Journal of Software: Evolution and Process, 29(11), 2017, pp.1-12.
48. Beck, K. Extreme Programming Explained: Embrace Change. Addison Wesley. 2000.
49. Clarke, P., O'Connor, R.V., Leavy, B.: A Complexity Theory viewpoint on the Software Development Process and Situational Context. In: proceedings of the International Conference on Software and Systems Process (ICSSP), pp. 86-90 (2016)
50. Rashid, M., Clarke, P., O'Connor, R.V. A Systematic Examination of Knowledge Loss in Open Source Software Projects. International Journal of Information Management (IJIM), Volume 46, June 2019, pp.104-123.
51. O'Connor, R.V., Elger, P., Clarke, P.: Exploring the impact of situational context: A case study of a software development process for a microservices architecture. In: proceedings of the International Conference on Software and Systems Process (ICSSP), Co-Located with the International Conference on Software Engineering (ICSE), pp. 6-10, DOI:10.1145/2904354.2904368 (2016)
52. Clarke, P., O'Connor, R.V., Solan, D., Elger, P., Yilmaz, M., Ennis, A., Gerrity, M., McGrath, S., Treanor, R.: Exploring Software Process Variation Arising from Differences in Situational Context. In: Proceedings of the 24th European and Asian Conference on Systems, Software and Services Process Improvement (EuroSPI 2017), pp.29-42, 5-8 September 2017, Ostrava, Czech Republic.
53. Giray, G., Yilmaz, M., O'Connor, R.V., Clarke, P.: The Impact of Situational Context on Software Process: A Case Study of a Small-sized Company in the Online Advertising Domain. In: Proceedings of the 25th European and Asian Conference on Systems, Software and Services Process Improvement (EuroSPI 2018), pp.28-39, 5-7 September 2018, Bilbao, Spain.
54. Marks, G., O'Connor, R.V., Clarke, P.: The impact of situational context on the software development process - A case study of a highly innovative start-up organization. In: Proceedings of the 17th International SPICE Conference (SPICE 2017), pp.455-466; 4-5 October 2017, Palma de Mallorca, Spain.
55. Clarke, P., O'Connor, R.V.: Changing situational contexts present a constant challenge to software developers. In: Proceedings of the 22nd European and Asian Conference on Systems, Software and Services Process Improvement (EuroSPI 2015), CCIS (Vol. 543), pp. 100-111,

30 September - 02 October 2015, Ankara, Turkey.

56. Clarke, P., O'Connor, R.V., Leavy, B., Yilmaz, M.: Exploring the Relationship between Software Process Adaptive Capability and Organisational Performance. IEEE Transactions on Software Engineering, 41(12), pp.1169-1183, doi: 10.1109/TSE.2015.2467388 (2015)