Exploring Aspects of Agile Software Development Risk – Results from a MLR

Aaron Nolan[1], Ben Strickland[1], Adam Quinn[1], Kyle Gallagher[1], Murat Yilmaz[2] [0000-0002-2446-3224] and Paul M. Clarke[1,3] [0000-0002-4487-627X]

[1] School of Computing, Dublin City University, Dublin, Ireland
{aaron.nolan67, ben.strickland2 ,adam.quinn73 ,kyle.mckenna22 }@mail.dcu.ie

[2] Department of Computer Engineering, Gazi University, Ankara, Turkey
my@gazi.edu.tr

[3] Lero, the Science Foundation Ireland Research Center for Software
paul.m.clarke@dcu.ie

Abstract. Agile software development methods are widely used by software organisations, focusing on short developmental life cycles and customer satisfaction through the iterative and incremental development of software products. Despite their popularity, these methods present risks that may be underappreciated. This paper examines certain risks attributed to agile software development, with a focus on the lack of documentation, scope creep, technical debt and job satisfaction. Through the application of a multivocal literature review, we find that agile software development can greatly benefit projects. However, when agile methods are implemented inappropriately or sub-optimally, projects risk over-spending, delayed or defective software, employee turnover, and overall decreased productivity. Understanding the risks associated with agile software development can help practitioners to achieve higher efficiency and success in their software development projects.

Keywords: Agile Software Development, Risk, Documentation, Technical Debt, Scope Creep, Job Satisfaction

1 Introduction

Traditional software development methods have been used widely throughout industry due to their positive impact on productivity and efficiency in software development [28]. Approaches such as the Waterfall involve a heavily structured approach, enabled by prioritising upfront design and with resistance to late changes and heavy use of documentation [33].

However, studies suggest that traditional software development may have only limited impact on productivity and can result in overspending, and the finished product risks not meeting customer expectations [28]. Agile software development was developed to mitigate the deficiencies relating to traditional software development methods. The Agile Manifesto [14] outlines a set of principles and methods that prioritise communication with customers, promote incremental and iterative development, and encourage requirements change throughout the development process.

These principles are achieved using Agile Methods such as Scrum and eXtreme Programming (XP), which have been shown to have many positive impacts on productivity along with shorter project duration [28] [14]. Agile software development initially catered for small-medium development teams but has increased in prevalence with pressure mounting for larger scale teams to adopt it. The increased complexity and scale of contemporary software development projects can introduce risks affecting the success of a company [30]. In this research paper, we aim to identify commonly reported risks associated with agile software development. We further discuss the impact of these risks on development projects and offer suggestions on how these risks might be mitigated. We defined the following research questions:

RQ1: Is a lack of documentation a risk for agile software development?
RQ2: In agile software development, what are the causes and effects of scope creep?
RQ3: What risk does technical debt pose in agile software development?
RQ4: Is job satisfaction be impacted by agile software development?

Section 2 of the paper discusses the research methodology, outlining the sources, queries, and inclusion/exclusion criteria. Section 3 presents an analysis of the selected agile development risks. Section 4 outlines the research limitations and possible future research directions. Section 5 presents the concluding observations.

2 Research Methodology

2.1 Methodology

This research adopted a Multivocal Literature Review (MLR), considering both white (peer-viewed, e.g. academic research papers) and grey (non-peer-reviewed, e.g. blogs) literature. Google, Google Scholar, and other search engines including IEEE, ScienceDirect, Springer and Pennsylvania State University were utilised.

2.2 Search Queries

Initial search strings included “Agile Software Development Challenges” and “Agile Software Development Risks”. Having investigated some of the returned sources, queries were further refined to increase relevance with the research questions. These queries were a combination of keywords such as “agile”, “technical debt”, “scope creep”, “job satisfaction” and “documentation”.

Having created a pool of research documents, we then filtered them through our exclusion and inclusion criteria, outlined in Section 2.3 below, and used a snowballing process to chase additional material of relevance to the research focus.

2.3 Inclusion and Exclusion Criteria

Inclusion Criteria
In the case of white literature, documents were grouped according to title, and were included if relevant to a research question (and in consideration of the academic

robustness of the source repository (e.g. Springer, IEEE)). In the case of grey literature, documents were included as a supplement to the white literature, for example where a specific salient consideration presented as under-evaluated in the white literature.

Exclusion Criteria
To promote accuracy and relevance in the incorporated literature, sources were deemed inappropriate if failing to satisfy the following criteria:

- Insightful views on the topic/theme researched
- Reliable and Identifiable source(s)
- (Sub-)Titles relevant to the topic/theme of interest.

2.4 Research Analysis

This subsection identifies the source and year of publication of incorporated research.

Table 1. The sources where research documents were reviewed and cited.

Source	Number reviewed	Number cited
IEEE	40	14
ScienceDirect	16	6
Springer	14	4
Pennsylvania State University	7	4
ACM	8	1
Lancaster University	1	1
University of Jyväskylä	1	1
ISO	1	1
DAU	1	1
Colorado State University	1	1
NCBI	1	1
OpenAccess	1	1

3 Analysis

3.1 Documentation in agile software development

In the words of Andrew Forward, documentation is defined as “an artefact whose purpose is to communicate information about the software system to which it belongs” [1]. This stresses the usage for communication amongst software engineers. Furthermore, Parnas defined software documentation as “a written description that has an official status or authority and may be used as evidence” [1] which provides precise information on the systems. An article published in 2013 stated “the main characteristics of agile development are short releases, flexibility, and minimal documentation” [2]. The idea of ‘minimal documentation’ is not to be confused with a ‘lack’ of documentation which will be further elaborated on below.

Lack of documentation may cause disruption to agile requirements engineering. In an agile requirements engineering (RE) paper published in 2012, some problems that arise from lack of documentation are presented, including a failure to thoroughly inspect requirements. This same research further explains that “exacerbated by agile RE … the lack of documentation makes it difficult to verify the system by inspections or walk-throughs” [3]. It furthermore asserts that “little documentation makes for quick implementation, but if the same requirements need to be changed the lack of requirements documentation could impede the evolution of the software” [3].

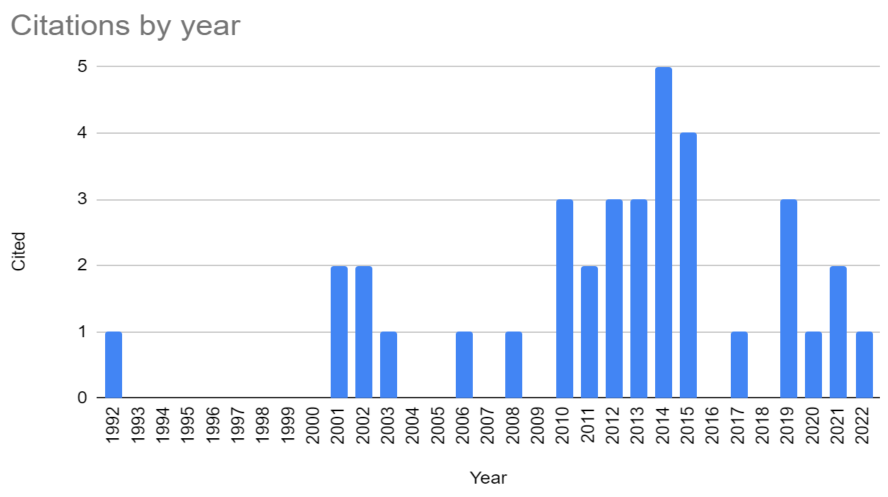
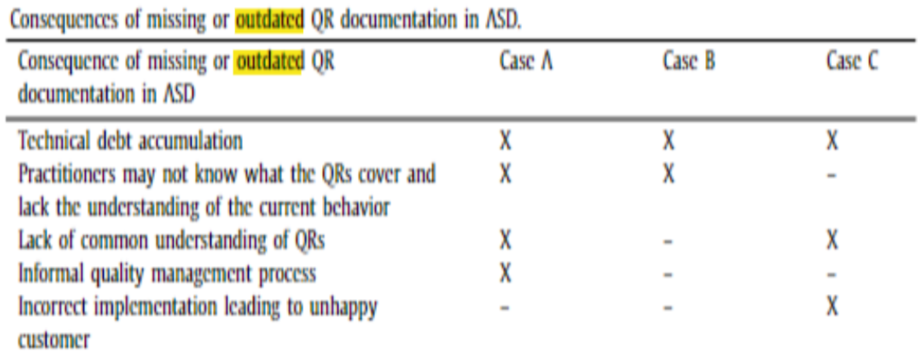


Fig. 1. The range of years that each cited research document was published in, along with the number of cited documents within each year.

Outdated Documents

“A document is outdated when it is not in sync with other parts of a system” [4] according to IEEE/ACM 41st International Conference on Software Engineering (ICSE) 2019. Outdated documentation can cause many problems for developers and makes it difficult for development and progress. In an article published in 2022 focused on optimal quality in agile software development, it is highlighted that “missing and outdated quality requirements (QR) documentation may lead to technical debt and a lack of common understanding regarding QRs” [5]. The point being that if documentation is outdated, it adds additional rework to an already time-pressured project (as well as a lack of knowledge on the outdated documentation). To further emphasise this point, it is stated that “missing and outdated QR documentation leads to incurring technical debt, a lack of common understanding of QRs, and incorrect implementations” [5]. Furthermore, the research identified five different consequences of outdated documentation and condensed them into a table as shown in Table 2 below.

Table 2. In the table, an X shows that responses from at least one participant or more participants of a case, are mapped to the theme on the corresponding row [5].



Consequences of missing or outdated QR documentation in ASD.

Consequence of missing or outdated QR documentation in ASD	Case A	Case B	Case C
Technical debt accumulation	X	X	X
Practitioners may not know what the QRs cover and lack the understanding of the current behavior	X	X	-
Lack of common understanding of QRs	X	-	X
Informal quality management process	X	-	-
Incorrect implementation leading to unhappy customer	-	-	X

Missing/incomplete Documents

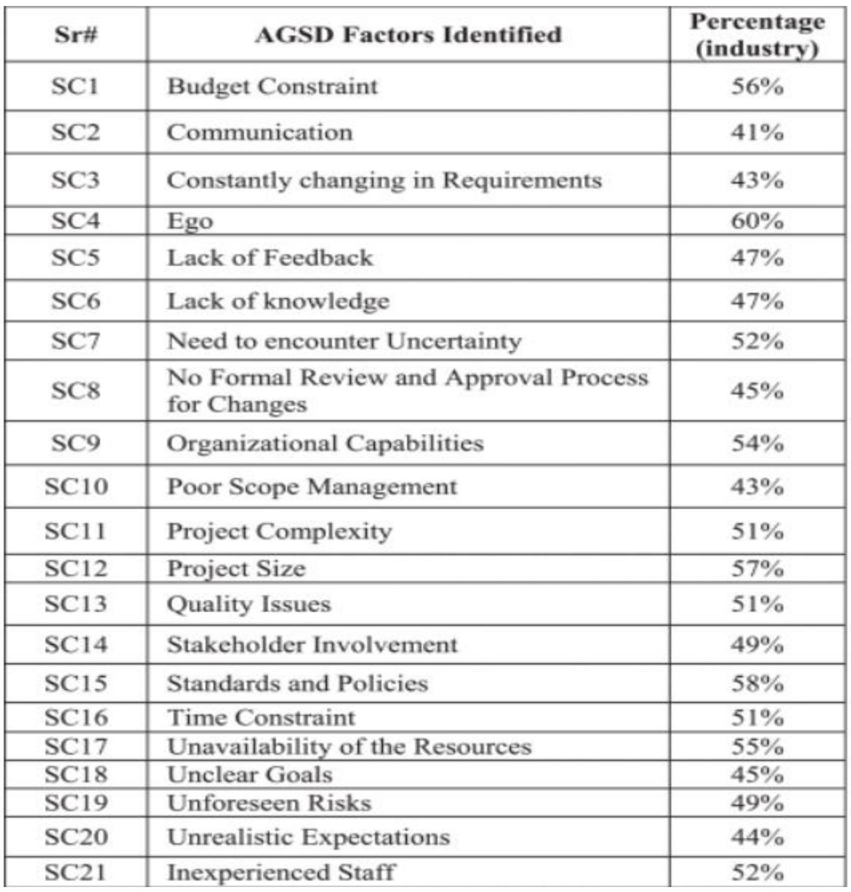
Documentation is incomplete if “it does not contain the information about the system or its modules needed by practitioners/users to perform their tasks” [4]. One reason for this could be that there is excessive documentation which makes it difficult to maintain a high standard for each document, resulting in loss or incomplete documentation. To emphasise this, an article published in 2017 noted that “in theory, agile software development generates only the minimally necessary software documentation” [6] which suggests that minimal documentation is practiced so as to avoid missing or incomplete documentation. Later, it also reports that a lack of documentation “may cause users and engineers to struggle to use or modify the software, and it can take several forms, including: software specifications, constraints, architecture, features, and rationales” [6] which shows the importance of maintaining a certain standard of documentation.

Lack of Document Quality

According to Ian Somerville, “much computer system documentation is badly written, difficult to understand, out of date or incomplete. Although the situation is improving, many organisations still do not pay enough attention to producing system documents which are well-written pieces of technical prose” [7]. The lack of quality can lead to many problems such as delayed completion times and problems with code. In a 2012 article looking at gamification of code quality in agile development, it is observed that “writing documentation is a form of well-behaving in software projects. A problem is, however, that “developers don’t like to do documentation, because it has no value for them (Selic, 2009)” [8]. It sems therefore that in some cases at least, “developers’ dislike for documenting leads to a lack of internal quality, which has become a pervasive problem in software projects” [8]. It is clear that developers need the motivation to correctly document their work in order to avoid a lack of quality in the documentation process. Perhaps this is also a question of professionalisation and maturity in software engineering.

3.2 Scope creep in agile software development

Scope creep can be considered a negative influence on software projects that occurs when “project managers compromise with [the] customer and accept new requirements which add changes to the scope of the project” [9]. The addition of these requirements is done while project development is still in progress, and this generally results in the cost, resources and time required to complete the project being increased from the budget initially agreed upon. In a systematic literature review published in July 2021, the primary factors causing scope creep within an agile software development team were identified. The diagram below highlights these factors, with the percentage referring to the frequency of occurrence within agile teams according to project managers surveyed as part of the SLR [10]. The three most frequent factors will be further detailed in Fig.2 below.



Sr#	AGSD Factors Identified	Percentage (industry)
SC1	Budget Constraint	56%
SC2	Communication	41%
SC3	Constantly changing in Requirements	43%
SC4	Ego	60%
SC5	Lack of Feedback	47%
SC6	Lack of knowledge	47%
SC7	Need to encounter Uncertainty	52%
SC8	No Formal Review and Approval Process for Changes	45%
SC9	Organizational Capabilities	54%
SC10	Poor Scope Management	43%
SC11	Project Complexity	51%
SC12	Project Size	57%
SC13	Quality Issues	51%
SC14	Stakeholder Involvement	49%
SC15	Standards and Policies	58%
SC16	Time Constraint	51%
SC17	Unavailability of the Resources	55%
SC18	Unclear Goals	45%
SC19	Unforeseen Risks	49%
SC20	Unrealistic Expectations	44%
SC21	Inexperienced Staff	52%

Fig. 2. Empirical Analysis of Identified Scope Creep Factors [10]

Ego

In this case, ego refers primarily to the behavior and personality of the project manager, who may have “inflated pride, ego, or confidence” in either themselves or their teams [10]. As a result of this, they may be more willing to accept new requirements and features that are beyond the capacity of the development team.

From an objective standpoint, the manager is expected to “be held responsible for project outcomes, yet they are expected to delegate decision making to the team” [11], yet in cases where they have an inflated ego, they may take the decision making process away from the team in relation to deciding whether new requirements can be accepted, and may neglect necessary steps prior to making these decisions, such as ensuring that the change or addition is “analyzed for resource, cost, and schedule impacts” [12].

In the context of an Agile team, emphasis is placed on frequent communication between team members, and the short length of iterations mean that sticking to deadlines and prioritising tasks in order to “identify what can be cut if something has to go” [12] are necessary actions. For scope creep to be avoided, it has been suggested that the manager must respect these obligations, avoid overconfidence and “have the strength, willingness, and communication skills” [12] to refuse new requirements when necessary. In the milieu of human and business interaction, the authors suggest that this is a non-trivial expectation. Furthermore, in software development more generally, the scope of individual roles in their practical implementation has been shown to vary considerably, for example, sometimes the *ScrumMaster* will act more like a traditional project manager (even though this is not advised in Scrum) [40]; therefore even though it is the Scrum Product Owner who theoretically represents the customer interests (and therefore prioritises feature implementation), in practice there can be no guarantee that this operates according to prescription in all settings.

Standards and Policies

This refers to a list of rules that define what practices are to be followed by both software development and project management teams. This is done to “achieve development process improvements that would otherwise be difficult to motivate and bring to fruition” [13] and ensure that development is as streamlined as possible. However, there are some difficulties with implementing these standards in an agile team.

For instance, one defining characteristic of the Agile Manifesto [14] is that “the most efficient and effective method of conveying information with and within a development team is face-to-face conversation” [14], also stating that the use of documentation should be deprioritised. Given that the standards followed by teams are generally defined by a third party (e.g. ISO) [15] and are so comprehensive, communication of these standards between team members becomes impossible without a method of storing them for later reference (i.e. documentation). This contradicts agile best practices. Given the necessity of quick and frequent communication when following this methodology, taking the time to research, apply and ensure a potentially large body of standards are followed takes time away from the tight schedule of agile increments, and in cases where following these standards are mandated as new requirements, or other additional requirements are required to be checked against these standards, this can potentially lead to scope creep.

Additionally, there are “almost no guidelines for incorporating into agile methodologies, processes that ensure their compliance with specified standards”, with the exception of a university research paper suggesting such guidelines [16]. This means that, in many cases, agile teams will have to infer their own methods for implementing standards, thus taking further time away from development.

Project Size

Agile development may be primarily suited to small to medium project sizes due to its prioritisation of “fast development and fast delivery” [10], though innovations such as SAFe [41] may aid agile in larger settings. The requirements of frequent, incremental deliverables and real-time communication become harder to achieve with larger projects, as communication (e.g. of requirements) must be done through more staff across multiple domains. The frequency of meetings required for this communication also increases as project size scales, with “cross-project sub-teams” [17] being required to coordinate work across teams, and an additional team to be established for “architecture and standards” [9] which ensures new features meet these criteria (see Standards and Policies for more details on this point).

Given that the acceptance of new requirements will mandate that these meetings be held and that they must be explained to necessary staff across project teams, increasing the project’s size will further extend these activities, and thus increase the likelihood of scope creep occurring. Additionally, studies have shown that “there exists an inverse relationship between the size of the project and the direct cost of scope creep” [18], which can be attributed to the increase in size causing “overconfidence in estimating realistic achievements” [18], primarily by project managers (see Ego for more details on this point).

3.3 **Technical debt in agile software development**

In 1992, Ward Cunningham outlined that the process of creating inefficient, unmaintainable and unexpandable code to quickly release an “acceptable” product to a customer will put the product “into debt”, adding that “a little debt speeds development so long as it is paid back promptly with a rewrite” [19], thus introducing the metaphor of Technical Debt. After two decades, in 2012, this metaphor was conveyed in a theoretical “Technical Debt Landscape” [20] as shown in Fig.3 below, defining that there are two categories of technical debt, visible and invisible. The visible technical debt consists of “new functionality to add and defects to fix” while the invisible technical debt consists of elements “visible only to software developers” [20]. Other technical debt frameworks have since been proposed [21].

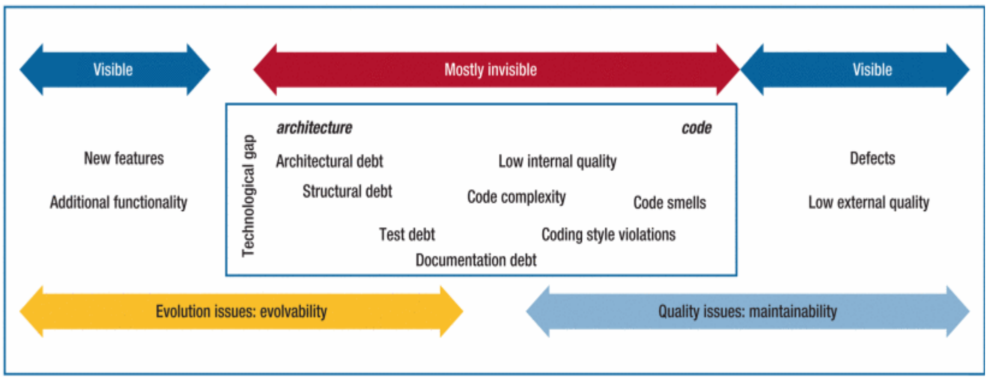


Fig. 3. Technical Debt Landscape. On the left, evolution or its challenges; on the right, quality issues, both internal and external. [20]

Agile software development is guided by a set of principles outlined in the Agile Manifesto [14]. These principles state that a software development team will "deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale" but that the "highest priority is to satisfy the customer through early and continuous delivery of valuable software" [14]. With the changing of requirements throughout the development process and the short time frame to get the product to the customer, sacrifices are made, creating both visible and invisible technical debt.

Invisible Technical Debt and Associated Cost

As shown in Fig.4, there are many forms of technical debt. Invisible technical debt, however, may be a high-risk factor in agile software development. In a study conducted on industry practitioners to determine the main causes of technical debt in agile software development [22], it was found that Architecture, Documentation (RQ1), Structure and Tests were the main causes of Technical debt.

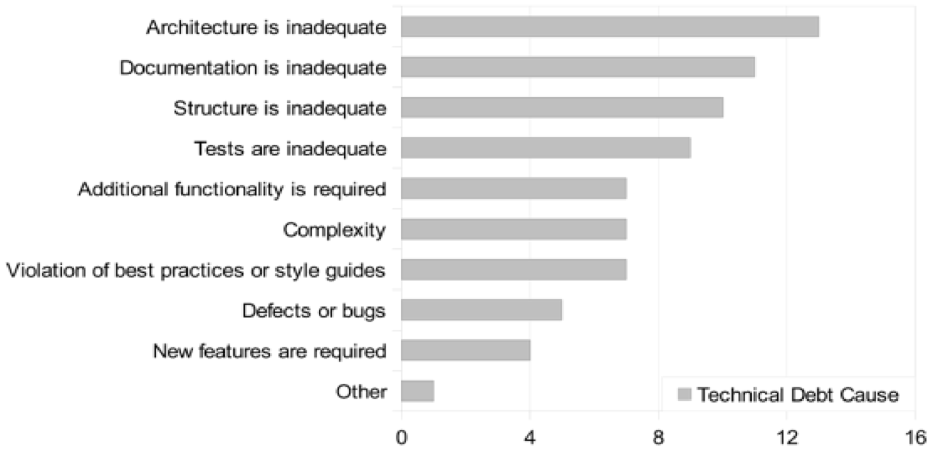


Fig. 4. Indicated causes for technical debt instances [22]

Fig.4 suggests that Architecture Debt is the main factor of Technical Debt in agile software development. Architectural Technical Debt (ATD) can be defined as “the result of sub-optimal upfront solutions, or solutions that become sub-optimal as technologies and patterns become superseded.” [21]. To identify the factors that create and cause ATD, a study in 2014 [23] investigated “what factors cause the accumulation of ATD” and concluded that alongside a list of other factors, documentation (RQ1), business factors (“evolution”, “unclear use cases”, “time pressure” and “feature prioritisation”) and parallel development all accumulate ATD. All of these factors are encouraged by the agile principles. In a 2015 study of ‘The Danger of Architectural Technical Debt’ across “7 sites at 5 large international software companies” [24], these same factors were investigated to determine their impact in the long term for an agile team. As shown in Fig.5 below, the “Interest” returned for having accumulated so much ATD creates a ‘phenomena/effect’ which triggers an ‘activity’. From this, it was proposed that “the real danger of some ATD items” are “contagious debt” and “vicious circles”.

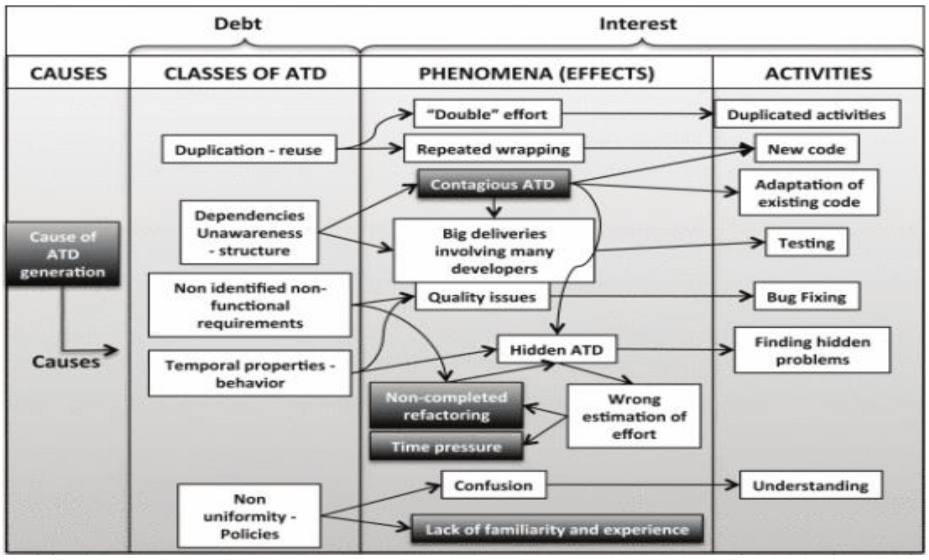

Debt
Interest
CAUSES
CLASSES OF ATD
PHENOMENA (EFFECTS)
ACTIVITIES
Cause of ATD generation
Causes
Duplication - reuse
Dependencies Unawareness - structure
Non identified non-functional requirements
Temporal properties - behavior
Non uniformity - Policies
"Double" effort
Repeated wrapping
Contagious ATD
Big deliveries involving many developers
Quality issues
Hidden ATD
Non-completed refactoring
Time pressure
Wrong estimation of effort
Confusion
Lack of familiarity and experience
Duplicated activities
New code
Adaptation of existing code
Testing
Bug Fixing
Finding hidden problems
Understanding

Fig. 5. The model shows the causes for ATD accumulation (black boxes), the classes of ATD (which represent the debt), the phenomena caused by the items and the final activities (which together represent the interest to be paid) [24].

Test Technical Debt (TTD) is the “second most studied” Technical Debt type [25]. TTD is the “lack of test scripts leading to the need to manually retest the system before every release, or insufficient test coverage regardless of whether tests are automated or manually run” [21], [26]. Even though testing is a critical component in software development and teams can be assigned solely for test automation, based on a study in 2013 [27], an “Agile Enterprise Team … whose only focus is to address technical debt (e.g., testing / test automation), mentioned that some debts remain in the backlog for extended periods if they are not impacting the development of new features". This research further states that "the priority is to develop new features and not to optimise existing ones", suggesting that the highest priority of agile software development promotes new feature delivery over technical debt management.

To show the impact and cost of technical debt in software development, Gartner estimated that the global technical debt bill will be "$500 billion in 2010 with the potential to double in five years' time" [21]. To summarise, technical debt is a risk factor in agile software development; although a modest amount of debt is sometimes warranted, if left unmanaged, it could be catastrophic for a software development business. It should also be noted that technical debt is nowadays routinely measured through use of tooling, for example, SonarCube [42].

3.4 **Agile software development job satisfaction**

The principles of agile software development introduce methods to overcome earlier development issues by focusing primarily on customer values, by delivering software frequently, welcoming changing requirements throughout the development life cycle, and continuous iteration of the product using a small, motivated team of developers. The main principles of agile software development state the "highest priority is to satisfy the customer through early and continuous delivery of software", to "welcome changing requirements, even late in development", and that "agile processes promote sustainable development" [14]. However, the collaborative and time-sensitive nature of agile software development methodologies can have negative impacts on the overall job satisfaction of the developers [28]. Pedrycs et al. outline communication, work sustainability, and work environment as the key elements to job satisfaction in software development [29].

Frustration with Agile practices

In a study conducted by Cho in 2008, it was found that core agile practices can lead to issues among developers. The need for constant communication with the customer was not always fulfilled, often only occurring when the project is complete. Unclear and undefined requirements set out by customers lead to "developers having a hard time figuring out what exactly the customer wants to include in their system" [28].

Lack of customer communication is also present in the stand-up practice, with a case study from 2010 into "Key Challenges in Agile Development" across 17 companies outlining customer involvement to be "highly passive", taking "more of an editor" role, and attending 28% of the stand up-meetings [30]. Further frustration with the stand-up practice stems from the inefficiency and perceived "waste of time" of planning and reviewing tasks, where the meetings "need to be adjusted based on the complexity of the project" being worked on [28].

It is noteworthy that when done correctly, agile practices do have a positive impact on the job satisfaction of agile team members over traditional software development [31]. We can infer that in contrast, poor implementation of common agile practices affects communication and work sustainability, which can lead to occupational stress and employee turnover [32].

Occupational Stress

A study conducted in 2018 suggests that stress can be linked directly to aspects of agile development [33]. Due to the high-intellectual nature of software development, and the need to "deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter time" [14], developer performance is directly affected by stress which leads to software defects, and further frustration among the development team [34]. The 2018 study into "Stress in Agile Software Development: Practices and Outcomes" found that stress is more common in new developers [34], with stress factors stemming from fear of providing unhelpful interactions and obsoleteness within the team and with clients. Stress has also been shown to impede knowledge sharing among experienced and inexperienced developers due to the time constraints outlined by the agile methodology, which prevents new team members from learning key skills, and decreases the productivity of experienced developers. [35] Research suggests stress has a negative overall impact on developer performance, work sustainability, and work environment, decreasing overall job satisfaction among teams, and increasing the risk of employee turnover [36].

Employee turnover

It is evident that certain agile practices can have a negative impact on the perceived job satisfaction of agile team members, through stress and frustration factors. The frustration and helplessness thus engendered is often a root cause of more systemic problems, such as developer turnover [36]. Turnover can be detrimental to an organisation, leading to large costs incurred for the company, with some studies estimating turnover costs to be between 70%-200% of an employee's annual salary [32]. These costs stem from costs relating to interviewing new candidates, training of new employees, and over time [36]. Training methods for new employees such as Pair Programming have been shown to be beneficial by building a sincere working environment, allowing for knowledge sharing, but can lead to "unequal participation and pair incompatibility" among developers [35]. Training can also decrease the productivity of developers "almost zero" to zero for existing team members, due to the need to prioritise training over development [35]. However, when implemented appropriately, companies can use agile software development to mitigate the risk of turnover, by stiving for higher job satisfaction through appraisal, continuous feedback, and improved communication between developers and customers [32].

4 Limitations and Future Work

The MLR undertaken in this research was limited to 6 weeks duration and was undertaken by 4 final year undergraduate computer science students. Both of these factors limit the effectiveness of the work. To mitigate this, the work was strategically designed, guided and reviewed by experienced academics, and detailed discussions were facilitated on a weekly basis.

A further significant limitation stems from the fact that although agile software development and its derivatives are widely used in practice, industry practitioners tend not to document or share their experiences widely. The result is that there is necessarily a significant gap in our information on applied agile software development. Future work might seek to better understand how some of the agile software development risks identified in this research are managed in practice in industry. Information related to this concern would be of value to the community in designing more robust software development practices and processes. It is furthermore the case that certain agile innovations that extend the original agile concepts may offer improved native treatment for some of the risks identified in this work, one example of which is the Scaled Agile Framework (SAFe) [41]. The scope of this work did not incorporate such perspectives, future work my seek to extend this research to achieve a broader understanding of agile risks in wider contexts. Additionally, sources such as the International Software Benchmarking Standards Group [39] could have been included to understand the benchmarks in current broad industrial practice.

This research indicates that there are relatively few studies examining agile software development risks (the observable tendency being to extol the virtues and not the weaknesses of agile development). It is not clear why this might be the case. Perhaps having invested in a development process, practitioners and academics alike are prone to bias and to seek out the positive outcomes. Future work might more thoroughly examine the risks associated with agile software development so that these can be comprehensively clarified for the broader software development community.

5 Conclusions

Since its introduction, agile software development has risen in popularity in industry. However, it may be the case that some elements of risk associated with agile software development are underappreciated. In RQ1, we examined if a lack of documentation can be a risk for agile software development. It was found that an absence of documentation or under-documentation can create problems for new project members, frustrating their efforts to learn a system that is unknown to them (and potentially leading to misunderstandings and errors). It is the view of the authors that certain principles may be helpful in reducing these risks while also taking advantage of the benefits of agile software development. For example, insisting on documenting towards the end of a project cycle when implementation decisions have been clarified provides for future system maintenance and evolution, while also avoiding a situation where documentation must be continuously updated as systems are innovated. Acquiring the discipline of storing electronic documents in easily accessible places will reduce the problem of missing or inaccessible documentation, it may also reduce the effect of wasteful documentation (documentation that is never used).

We suggest that treating-documentation-as-code might be a helpful concept – if documents are produced for software systems that are past or passing the aggressive innovation and discovery phase, the version control of minimal (yet instructive) documents in a version control system (e.g. GitHub), might be an attractive proposition. Of course, such documents might later need to be maintained and evolved to reflect changes in system implementation. Clearly, if an agile software project abandons all documentation, it may increase the risks for later engineers seeking to evolve or maintain a system. The role of high-quality code is also of significant importance – code that is well-written and easily read might be a potent form of documentation. Furthermore, tooling that can read code and present architectural and design views can be potentially very helpful (perhaps reducing the required volume of separate documentation).

In examining RQ2, we conclude that there are a variety of factors that can give rise to scope creep in agile projects. Human factors, such as the project manager's personal strength and experience (as well as their respect for and communication with the team) weigh heavily on scope creep. An inexperienced manager, or one with a sub-optimal software development understanding, can be a facilitator for inadvertent and unwise scope creep. Equally, team members can make these same errors of judgement. This human element reminds us that software development is a human-intensive business and is therefore necessarily critically dependent on a team's personalities and capabilities (both their hard skills and soft skills). Our research suggests that differentiating scope creep from attractive new feature innovation must be one of the most difficult challenges facing software development teams. And perhaps in some instances, there is no easy way of foretelling if proposed features (or extensions to existing features) will be justified at some future point. It is perhaps in this space that the magic of experience and human talent can reduce the worst excesses of unbridled scope creep.

Perhaps firms could explicitly evaluate the utilisation of new feature implementation and from this establish if features were genuinely required. Whatever the case, scope creep can be damaging – every line of code deployed to production systems can potentially be a source of a defect, and it may require maintenance for an extended period. We suggest therefore, that agile software development teams as hard questions of proposed user stories. The most important questions might be: "Is this user story really needed?", "Who will use this user story?" and "Are we happy that implementing this user story represents a coherent and sensible decision for our system?"

When examining the risk posed by technical debt in agile software development settings (RQ3), our research suggests that this debt may be exasperated by a constant prioritisation of product release over sustainable development. In agile software development, invisible technical debt may become a significant concern and risk to development. Invisible technical debt is composed of various factors, primarily architectural and test debt. Architectural debt is the result of sub-optimal upfront solutions, or solutions that became sub-optimal as technologies and patterns were superseded. It can accumulate to the point of contagious debt. Test debt is the lack of test scripts leading to the need to manually retest the system before every release, or insufficient test coverage regardless of whether tests are automated or manually run. Our research suggests that in agile, there tends to be a dedicated team creating test automation, however, in the absence of robust automated testing and in an environment of constant impending deadlines, technical and test debt can grow. When this occurs, system and project viability risks also grow.

Our final research question (RQ4) examines job satisfaction in agile development settings, finding that communication, work environment, and work sustainability are among the primary factors affecting job satisfaction. Although not an intention of agile method creators, some agile practices may negatively affect job satisfaction and employee turnover. Increased customer interaction might frustrate those developers who prefer coding over communication, yet it may become essential due to the decreased focus on explicit requirement definition in agile software development. Frequently recurring delivery deadlines might increase stress levels for some development professionals, as might the uncertainty associated with limited requirements definition. Higher stress levels might decrease productivity over time, and lead to increased incidence of errors. It might also lead to higher employee turnover. In the case of these two latter, points, they could arise in any software development setting, not just those with a agile software development approach.

No single software process can be perfectly suited to all software settings [37], there is too much contextual variation [38] to be accommodated. For most of the past 20 years, agile development practices have likely held sway for the mainstream of non-safety critical software engineering. Indeed, the original concept of agile software development itself may be becoming historic at this point, given the subsequent innovations in automation, cloud infrastructure, and other aligned technologies. Much has changed since agile was first introduced, indeed the absence of larger agile frameworks (such as SAFe [41]) in this research is a significant limitation. Nevertheless, the general paradigm influenced by the Agile Manifesto [14] does exhibit some risks, and it is for this reason that this research has been undertaken. A chain of events arises from the reduced focus on detailed documented requirements: there is limited ability to design systems prior to implementation, knowledge becomes increasingly tacit, and scope can become difficult to manage. So too can stress levels among employees rise, since delivery deadlines are frequent and successfully delivering depends significantly on high quality user engagement and communication (which itself requires time and investment). For all of these frailties, the community has clearly found ways to deal with them, as otherwise agile software development would not have risen to its position of prominence.

Acknowledgements. This research is supported in part by SFI, Science Foundation Ireland (https://www.sfi.ie/) grant No SFI 13/RC/2094 P2 to Lero - the Science Foundation Ireland Research Centre for Software.

6 References

1. Zhi, J., Garousi-Yusifoğlu, V., Sun, B., Garousi, G., Shahnewaz, S. and Ruhe, G., 2015. Cost, benefits and quality of software development documentation: A systematic mapping. Journal of Systems and Software, 99, pp.175-198.
2. Hadar, I., Sherman, S., Hadar, E. and Harrison, J.J., 2013, May. Less is more: Architecture documentation for agile development. In 2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE) (pp. 121-124). IEEE.
3. Haugset, B. and Stalhane, T., 2012, January. Automated acceptance testing as an agile requirements engineering practice. In 2012 45th Hawaii International Conference on System Sciences (pp. 5289-5298). IEEE.
4. Aghajani, E., Nagy, C., Vega-Márquez, O.L., Linares-Vásquez, M., Moreno, L., Bavota, G. and Lanza, M., 2019, May. Software documentation issues unveiled. In 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE) (pp. 1199-1210). IEEE.
5. Behutiye, W., Rodríguez, P., Oivo, M., Aaramaa, S., Partanen, J. and Abhervé, A., 2022. Towards optimal quality requirement documentation in agile software development: a multiple case study. Journal of Systems and Software, 183, p.111112.

6. Saito, S., Iimura, Y., Massey, A.K. and Antón, A.I., 2017, September. How much undocumented knowledge is there in agile software development?: Case study on industrial project using issue tracking system and version control system. In 2017 IEEE 25th International Requirements Engineering Conference (RE) (pp. 194-203). IEEE.
7. Sommerville, I., 2001. Software documentation. Software engineering, 2, pp.143-154.
8. Prause, C.R., Nonnen, J. and Vinkovits, M., 2012. A Field Experiment on Gamification of Code Quality in Agile Development. In PPIG (p. 17).
9. Madhuri, K.L., Rao, J.J. and Suma, V., 2014. Effect of Scope Creep in Software Projects a [euro]" Its Bearing on Critical Success Factors. International Journal of Computer Applications, 106(2).
10. Aizaz, F., Khan, S.U.R., Khan, J.A. and Akhunzada, A., 2021. An Empirical Investigation of Factors Causing Scope Creep in Agile Global Software Development Context: A Conceptual Model for Project Managers. IEEE Access, 9, pp.109166-109195.
11. Shastri, Y., Hoda, R. and Amor, R., 2021. The role of the project manager in agile software development projects. Journal of Systems and Software, 173, p.110871.
12. Turk, W., 2010. Scope creep horror. Defense AT&L, 39(2), pp.53-55.
13. Tuohey, W.G., 2002. Benefits and effective application of software engineering standards. Software Quality Journal, 10(1), pp.47-68.
14. Fowler, M. and Highsmith, J., 2001. The agile manifesto. Software development, 9(8), pp.28-35.
15. "ISO Brief 2019". 2019. Iso.Org. https://www.iso.org/files/live/sites/isoorg/files/store/en/PUB100007.pdf (accessed: 03/06/2022)
16. Theunissen, W.M., Kourie, D.G. and Watson, B.W., 2003. Standards and agile software development. In Proceedings of SAICSIT (Vol. 30, No. 3, pp. 178-188).
17. Reifer, D.J., 2002, August. How to get the most out of extreme programming/agile methods. In Conference on Extreme Programming and Agile Methods (pp. 185-196). Springer, Berlin, Heidelberg.
18. Komal, B., Janjua, U.I., Anwar, F., Madni, T.M., Cheema, M.F., Malik, M.N. and Shahid, A.R., 2020. The impact of scope creep on project success: An empirical investigation. IEEE Access, 8, pp.125755-125775.
19. Cunningham, W., 1992. The WyCash portfolio management system. ACM SIGPLAN OOPS Messenger, 4(2), pp.29-30.
20. Kruchten, P., Nord, R.L. and Ozkaya, I., 2012. Technical debt: From metaphor to theory and practice. IEEE software, 29(6), pp.18-21.
21. Tom, E., Aurum, A. and Vidgen, R., 2013. An exploration of technical debt. Journal of Systems and Software, 86(6), pp.1498-1516.
22. Holvitie, J., Leppänen, V. and Hyrynsalmi, S., 2014, September. Technical debt and the effect of agile software development practices on it-an industry practitioner survey. In 2014 Sixth International Workshop on Managing Technical Debt (pp. 35-42). IEEE.
23. Martini, A., Bosch, J. and Chaudron, M., 2014, August. Architecture technical debt: Understanding causes and a qualitative model. In 2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications (pp. 85-92). IEEE.
24. Martini, A. and Bosch, J., 2015, May. The danger of architectural technical debt: Contagious debt and vicious circles. In 2015 12th Working IEEE/IFIP Conference on Software Architecture (pp. 1-10). IEEE.
25. Li, Z., Avgeriou, P. and Liang, P., 2015. A systematic mapping study on technical debt and its management. Journal of Systems and Software, 101, pp.193-220.
26. Martini, A., Stray, V. and Moe, N.B., 2019, May. Technical-, social-and process debt in large-scale agile: an exploratory case-study. In International Conference on Agile Software Development (pp. 112-119). Springer, Cham.

27. Codabux, Z. and Williams, B., 2013, May. Managing technical debt: An industrial case study. In 2013 4th International Workshop on Managing Technical Debt (MTD) (pp. 8-15). IEEE.
28. Cho, J., 2008. Issues and Challenges of agile software development with SCRUM. Issues in Information Systems, 9(2), pp.188-195.
29. Pedrycz, W., Russo, B. and Succi, G. (2011). A model of job satisfaction for collaborative development processes. Journal of Systems and Software, 84(5), pp.739–752.
30. Conboy, K., Coyle, S. and Wang, X., 2010. People over Process: Key Challenges in Agile Development,(99), 48–57.
31. J. F. Tripp and C. K. Riemenschneider, "Toward an Understanding of Job Satisfaction on Agile Teams: Agile Development as Work Redesign," 2014 47th Hawaii International Conference on System Sciences, 2014, pp. 3993-4002.
32. Melnik, G. and Maurer, F., 2006, June. Comparative analysis of job satisfaction in agile and non-agile software development teams. In International conference on extreme programming and agile processes in software engineering (pp. 32-42).
33. Sillitti, A., Wang, X., Martin, A. and Whitworth, E., 2010. Agile Processes in Software Engineering and Extreme Programming. In 11th international conference, XP (pp. 1-4).
34. A. Amin, S. Basri, M. F. Hassan and M. Rehman, "Software engineering occupational stress and knowledge sharing in the context of Global Software Development," 2011 National Postgraduate Conference, 2011, pp. 1-4.
35. Ersoy, I.B. and Mahdy, A.M., 2015. Agile knowledge sharing. International Journal of Software Engineering (IJSE), 6(1), pp.1-15.
36. Madhura, S., Subramanya, P. and Balaram, P. (2014). Job satisfaction, job stress and psychosomatic health problems in software professionals in India. Indian Journal of Occupational and Environmental Medicine, 18(3), p.153.
37. 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)
38. 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.
39. International Software Benchmarking Standards Group. https://www.isbsg.org/ (accessed: 03/06/2022)
40. Clarke, P., Mesquida Calafat, A.L., Ekert, D., Ekstrom, J.J., Gornostaja, T., Jovanovic, M., Johansen, J., Mas, A., Messnarz, R., Najera Villar, B., O'Connor, A., O'Connor, R.V., Reiner, M., Sauberer, G., Schmitz, K.D., Yilmaz, M.: An Investigation of Software Development Process Terminology. In: Proceedings of the 16th International Conference on Software Process Improvement and Capability dEtermination, pp. 351-361 (SPICE 2016)
41. Leffingwell, D., et al.: Scaled Agile Framework (SAFe). https://www.scaledagileframework.com/ (accessed 03/06/2022)
42. SonarCube: https://www.sonarqube.org/ (accessed 03/06/2022)