12-2-2023

A Systematic Review of Complexity in Agile Software Project Delivery

Taniela Vaipulu
Project Management Research Office, Software Engineering Research Centre, ECMS, Auckland University of Technology, New Zealand, daniel.vaipulu@aut.ac.nz

Ramesh Lal
Project Management Research Office, Software Engineering Research Centre, ECMS, Auckland University of Technology, New Zealand, ramesh.lal@aut.ac.nz

Stephen Thorpe
Project Management Research Office, Software Engineering Research Centre, ECMS, Auckland University of Technology, New Zealand, stephen.thorpe@aut.ac.nz

A Systematic Review of Complexity in Agile Software Project Delivery

Taniela Vaipulu
Project Management Research Office
Software Engineering Research Centre
Department of Computer Science and Software Engineering
School of Engineering, Computer and Mathematical Sciences
Auckland University of Technology
Email: daniel.vaipulu@aut.ac.nz

Ramesh Lal
Project Management Research Office
Software Engineering Research Centre
Department of Computer Science and Software Engineering
School of Engineering, Computer and Mathematical Sciences
Auckland University of Technology
Email: ramesh.lal@aut.ac.nz

Stephen J. Thorpe
Project Management Research Office
Software Engineering Research Centre
Department of Computer Science and Software Engineering
School of Engineering, Computer and Mathematical Sciences
Auckland University of Technology
Email: stephen.thorpe@aut.ac.nz

Abstract

In Agile software delivery, *complexity* factors have become an important part that has been linked to the success of software projects, especially with highly structured plan-driven approaches. Although there are a range of known techniques, practices and structures aimed at improving communication, team, and organisational cohesion, there remains a poor understanding of how organisations and teams can effectively deal with complexity in Agile software delivery. While theoretical and management frameworks of complexity exist, nevertheless, little is known about the important complexity factors at play in the planning, design, and creation of Agile software at the various levels of organisational structures. To better understand complexity and how it may impact Agile delivery across organisational structures, processes and roles, a systematic literature review was undertaken to identify the known complexity factors, to identify gaps in our knowledge, and to identify the approaches that have been undertaken to date to address complexity. This review contributes a fresh perspective to the literature on complexity in Agile software delivery and forwards our conversation on complexity in IT projects, and their effective management. The review was structured around Remington and Pollack's (2016) model identifying different types of complexity. The review's findings indicate that complexities result from aspects in functional teams, organizational structures, roles, Agile tasks, and practice-related issues that were identified in distributed Agile projects. We conclude that there may remain a lack of understanding in practice of the complexity factors that are at play in an Agile software delivery context. These findings are relevant to researchers and practitioners seeking to identify areas of focus for addressing complexity in Agile software delivery.

Keywords Agile Software Development, Agile Project Delivery, IT Project Failure, IT Project Complexity, Agile Structural Complexity, Agile Technological Complexity, Agile Directional Complexity, Agile Temporal Complexity.

1 Introduction

Research into software project complexities has gained recent momentum (Rezende, 2022). Complexity has become a prevalent factor in the delivery of software projects (Zaman, 2019). With the rise of emerging technologies and the globalization of software products and development, projects must deal with a wide range of complexities to minimize the risks in project delivery (Adebayo Agbejule, 2022). Complexity has characteristics and can be categorized into several levels of severity, making it difficult at times to create reliable project plans, to effectively monitor and control projects, and to deliver

outcomes (Remington, 2009). Several complexity factors have been identified in the literature relevant to software project delivery, including in the areas of effective structural setup and the expansion of roles. In attempts to better explain complexity, methods such as classification by types in terms of their properties has been used in complex systems theory to provide an understanding of complexity behaviours (Pringle-Wood, 2013). Remington and Pollack's (2007) research focused on categorizing complex projects, and suggested tool support and project manager attributes were vital to deal with complex projects.

Complexity management is well recognized as a key determinant of project success with large-scale software product development (see Cristobal, 2018; Patanakul, 2014). As such, software products have many unique and innovative requirements, including emergent requirements that are difficult to identify, understand, and to implement. Hence, complexity is anticipated to be a major part of software projects, specifically impacting plan-driven approaches and processes (Diane Strode, 2012; Mesjasz, 2020; Pringle-Wood, 2013). Basic Agile methods (such as Scrum, XP) have been recognized for their limitation to scaling up for large-scale product development. There is little research to show how scaled-Agile methods (such as SAFe, DAD) can effectively deal with project complexities. Hence, we argue that research is required to help understand software development complexities and to potentially identify how to approach complexity in Agile software project delivery.

Over the last two decades, Agile methods have slowly replaced the highly structured plan-driven approaches to software project delivery; changing the nature of how projects are planned and executed (Ahmed, 2021). While Agile offers numerous methods and practices, it also introduces a unique set of challenges; prominently complexity (Cobb, 2011). Despite a limited understanding of Agile complexity, Agile approaches such as 1) the use of cross-functional teams, 2) collaborative decision-making, 3) knowledge sharing, and 4) retrospection practices can be used to deal with project complexities (Ahmed, 2021). While Agile has been widely adopted and commended for its benefits, there has been a lack of robust empirical evidence on how Agile deals with complexity (Rasheed, 2021). Agile case studies and success stories abound, but rigorous academic research validating Agile's effectiveness in handling complexity has been limited (Mishra, 2011). Some of this knowledge gap is related to the fact that Agile projects are often diverse and context-dependent, making it challenging to conduct standardized evaluations across different scenarios (Rasheed, 2021).

Driven by the motivation to provide an understanding on the complexity of Agile project delivery through a major empirical investigation, an essential step of the systematic review presented in this article was to investigate the existing Agile software literature to identify issues and challenges with development and delivery. The aim was to help gain a better understanding of the research problems and to allow us to validate our research questions. Hence, a Systematic Literature Review (SLR) was undertaken to identify, categorise and group complexity factors in Agile software delivery based on the findings presented in literature.

2 Complexity in Software Development

Complexity has become a pervasive phenomenon in software development projects, intricately woven into the various systems, processes, and interactions (Mitchell, 2009). Researchers explicitly focusing on complexity have often tried to explain complexity using simple dictionary definitions, such as, 'consisting of many interconnected parts in terms of physical elements and their interdependencies', and, at times, by appending the term 'uncertainty' to it (Baccarini, 1996; Geraldi & Adlbrecht, 2007; Williams, 1999). While others have tried to explain through *complexity theory*, e.g., Remington and Pollack (2007), and Cooke-Davies, et al. (2007). The foundation of complexity theory itself lies in interdisciplinary research (such as Burnham, 2020; Cicmil et al., 2009; Morris, Bennet & Fine, 1990; Wozniak, 1993; and Baccarini, 1996) that drew insights from project management literature. They emphasized that the emergence of complexity theory was a response to the limitations of traditional plan-driven reductionist approaches and the rise in comprehensive levels of complexity in projects, and in particular, complexity's implications for affecting project goals and objectives, and the effectiveness of various process, project organization, forms, and the management of people. The number of failed software development projects has risen significantly over decades, according to many, such as the work of Alsulamy (2022), Buhl and Meier (2011), and Yeo (2002). Alsulamy (2022) also adds that failures of software projects are a common pattern rather than an exception. The reasons attributed to these failures vary, from reasons to do with complexity, including technology, management, project environments, cost overruns and budget, schedule delays, adoption of traditional approaches and inflexible organizational structures (Patanakul and Omar, 2010; Willcocks and Griffiths, 1994).

2.1 Remmington & Pollack's Tools for Complex Projects

Categorising various components of project complexity to better understand the impact of project complexity can be highly beneficial in supporting large-scale software development. The *Tools for Complex Projects* model (Remington & Pollack, 2016) was designed to be a practical guide for recognizing different types of complexity with the goal of selecting the most appropriate tools and approaches to best manage complexity in projects. Pollack has argued that strategies for managing different kinds of complexity exhibited in various projects or project parts may need to vary enormously (Pollack, 2007). Based on the known sources of complexity, and informed by the work of others, such as, Turner and Cochrane (1993), and Williams (2002), the model suggests four types of project complexity as functional categories for complexity analysis: *structural*, *technical*, *directional*, and *temporal* complexity (Pollack, 2007).

Structural complexity refers to the intricate and interconnected nature of elements within a system, process, or project, where the relationships and dependencies among these elements contribute to the system's overall complexity (Pollack, 2007). Structural complexity measures how intricate and entangled the components are, impacting the system's behaviour, performance, and understandability. In software development projects, structural complexity affects outcomes and introduces challenges (Pollack, 2007). Structural complexity research involves analysing the interconnections and interactions among components and identifying patterns, dependencies, and emergent behaviours (Pollack, 2007).

Technical complexity is often used interchangeably with technological complexity. Both terms refer to the intricate and sophisticated nature of technical systems, processes, or solutions (Baccarini, 1996; Geraldi & Adlbrecht, 2007). Both terms encompass the challenges arising from advanced technologies, intricate methodologies, and specialized knowledge required to successfully design, develop, and implement technical solutions. Whether referred to as technical complexity or technological complexity, the concept remains the same and are observed in software development projects (Cicmil, 2009).

Directional complexity refers to the inherent challenges and intricacies associated with the directionality of project tasks or inter-team processes. Directional complexity involves the management of complexity that arises from navigating specific directions, objectives, or goals within the context of the project's scope and requirements (Pollack, 2007).

Temporal complexity in software projects refers to the intricacies and challenges of managing time-related aspects throughout the software development lifecycle (Baccarini, 1996; Mitchell, 2009; Pollack, 2007). This aspect of complexity is particularly relevant in today's fast-paced and ever-changing technology landscape, where delivering high-quality software products on time is considered crucial for maintaining competitive advantage (Remington, 2009; Cicmil, 2009).

3 Planning the Systematic Literature Review (SLR)

3.1 The Systematic Literature Review Protocol

The protocol for the review was developed following the three stages suggested by Kitchenham (2012) (see Figure 1). A SLR protocol was used to help define the review process, including a search strategy, selection criteria, quality review, and inclusion and exclusion criteria. Such a review protocol is aimed at reducing the likelihood of researcher bias (Kitchenham, 2012).

3.1.1 Aim and Research Question

The review aimed to identify complexity factors based on issues and challenges highlighted in Agile software development literature. Hence, the following research question was articulated:

RQ: *What are the complexity factors in Agile project delivery?*

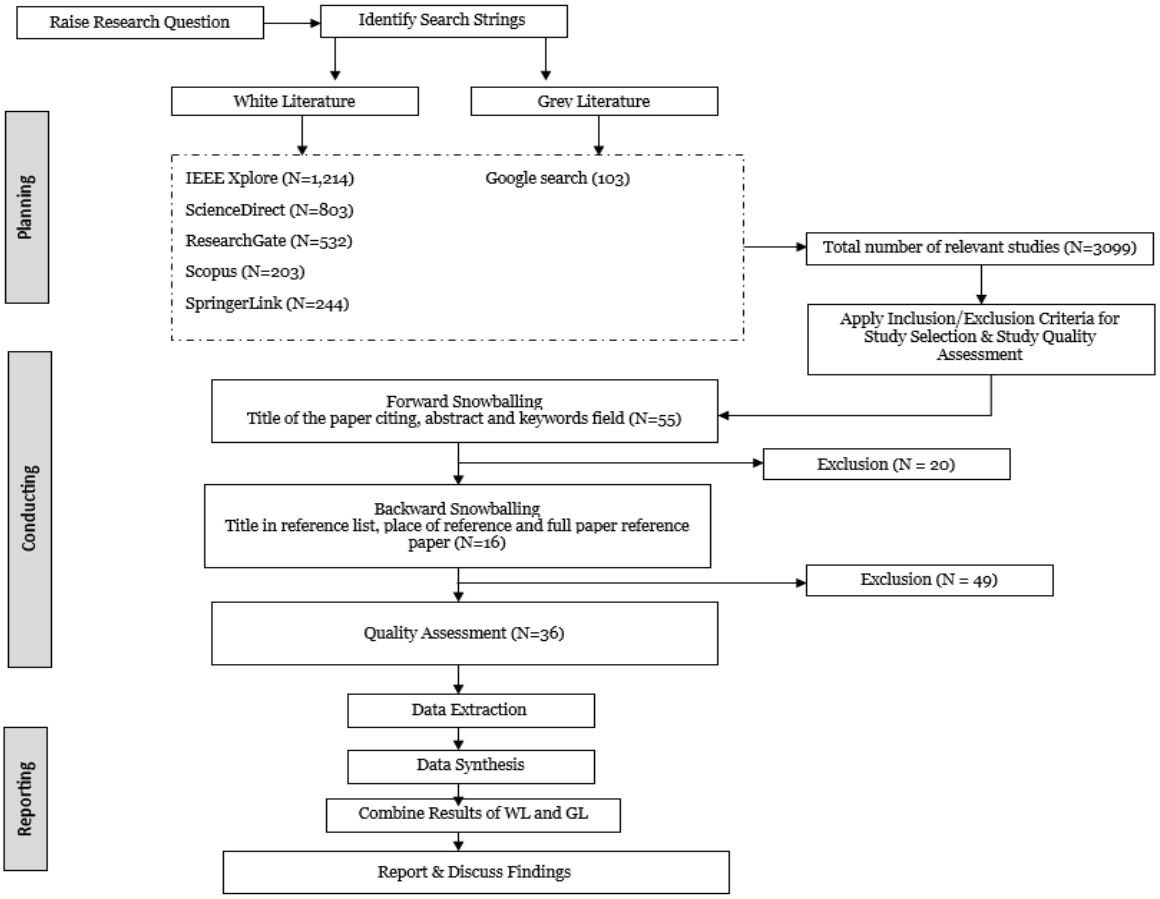

Figure 1 Flow diagram detailing the review process.

3.1.2 Search Strategy and Criteria

Our search strategy identified the following databases for academic publications: IEEE Xplore, ScienceDirect, Scopus, ResearchGate, Google Scholar, and SpringerLink. The study used the search criteria that consist of five substrings A1, A2, A3, A4, and A5, described as follows:

A1 is a string composed of keywords including "complexity" OR "issues" OR "challenges" OR "ambiguity" OR "Uncertainty" OR "constraints".

A2 is a string composed of keywords including "structural" OR "functional setup" OR "role" OR "size" OR "scope" OR "interdependence" OR "duration" OR "organization".

A3 is a string composed of keywords including "Agile project" OR "Agile methodology" OR "Agile method" OR "Agile approach" OR "Agile development".

A4 is a string composed of keywords including "directional" OR "duration" OR "project duration" OR "activities" OR "goals" OR "objective".

A5 is a string composed of keywords including "technological" OR "process" OR "procedures" OR "technical" OR "platform" OR "infrastructure" OR "product".

The five substrings in the search criteria are combined using the conjunctive "AND". Therefore, the first search string used in this study is (A1+A2+A3), which shows ("complexity" OR "issues" OR "challenges" OR "ambiguity" OR "Uncertainty" OR "constraints") AND ("Structural" OR "functional setup" OR "role" OR "size" OR "scope" OR "interdependence" OR "duration" OR "organization") AND ("Agile project" OR "Agile methodology" OR "Agile method" OR "Agile approach" OR "Agile development")

The second search string consists of (A1+A5+A3), which is shown as ("complexity" OR "issues" OR "challenges" OR "ambiguity" OR "Uncertainty" OR "constraints") AND ("technological" OR "process" OR "procedures" OR "technical" OR "platform" OR "infrastructure" OR "product" AND ("Agile project" OR "Agile methodology" OR "Agile method" OR "Agile approach" OR "Agile development")

The third search string consists of (A1+A4+A3), which is shown as ("complexity" OR "issues" OR "challenges" OR "ambiguity" OR "Uncertainty" OR "constraints") AND ("directional" OR "duration" OR "project duration" OR "activities" OR "goals" OR "objective") AND ("Agile project" OR "Agile methodology" OR "Agile method" OR "Agile approach" OR "Agile development")

We set the inclusion and exclusion criteria to assess if a study publication should be included in the review to answer the research questions. The inclusion criteria focused on (1) publications in the period of 2002 to 2022, (2) Only peer-reviewed journal and conference articles written in English were selected (3) Screen titles and abstracts using the inclusion/exclusion criteria, and (4) some reviews may include grey literature. Whereas the exclusion criteria focused on when (1) Full text was not available in English through the database engine, (2) Studies that did not focus explicitly on the interest phenomena, and (3) short papers, PowerPoint presentations, and posters. These were excluded from the study.

3.2 Conducting the Systematic Literature Review

Once the review methodology was designed, the review process began. This section discusses the information search and extraction outcomes from the selected databases.

3.2.1 Study Selection

In the initial round of forward snowballing, we conducted a search in designated electronic databases following the protocol outlined in Section 4.1.2 to acquire the most relevant studies. This preliminary search yielded a total of 3,099 studies. Using Wohlin and Claes's *Guidelines for Snowballing* (2014), a forward snowballing technique was applied, focusing on the title of the paper or article, the abstract, and the keywords fields. This approach generated a list of 55 potential papers containing the specified search text in these areas. After analysing the titles and abstracts of these papers, and by applying the inclusion and exclusion criteria, 20 papers were included in the final review. In the second round, we employed backward snowballing to assess further possible research articles. This involved reviewing the titles in the reference lists, the locations of references, full paper reference articles, and any relevant sections to ensure alignment with the pre-defined criteria. This process resulted in a final list of 65 potential papers. Again, after applying the inclusion and exclusion criteria, 49 papers were removed from consideration as they did not explicitly explore topics related to the search strings or did not relate to our research question. Subsequently, the final selection for the review comprised of 36 studies.

3.3 Quality Assessment

The quality evaluation of this study followed the criteria presented by Garousi et al. in their article *Guidelines for Including Grey Literature and Conducting Multivocal Literature Reviews in Software Engineering* (2019). The quality assessment aimed to gauge the methodological rigour of selected papers. The criteria encompassed five dimensions, each with specific questions to assess the study's credibility and relevance to our research scope. Table 1 offers a detailed breakdown of quality criteria and associated questions. A measurement score from 1 to 4 was used to evaluate each criterion. In the first criterion (A1), 32 papers scored 4 out of 4 for high credibility, while four achieved a medium credibility score of 3 out of 4. Criterion (A2) resulted in 30 high credibility scores and six medium credibility scores. Criterion (A3) yielded a consistent perfect score of 4 out of 4 across all 36 papers. Likewise, criteria (A4) and (A5) garnered high credibility scores, with all 36 papers scoring 4 out of 4.

Criteria	Questions
(A1) Authority of the publisher	Is the publishing organisation reputable/ Author is associated with reputable organisation?
	Is the organisation/ author cited often by others?
	Has the author published other work in the field?
(A2) Source context	Does the source have a clearly stated aim/objective?
	Is the focus of the study on challenges or issues in Agile software development?
	Are any limits clearly stated?
	Is the source supported by documented references?
	Are the conclusions justified by the result?
(A3) Publication Date	Does the source have a clearly stated date?
(A4) Significance of work	Does the source enrich the current research and/or particularly add something unique?
(A5) Publication/literature type	Academic peer-reviewed
	Journal Article
	PhD. /Master thesis
	Conference Paper

Table 1 Quality Criteria Applied to Articles and Papers

4 Findings

Our process for coding and deriving themes involved the first author creating the initial findings which then were jointly reviewed and refined with two other authors. Tables 2, 4, 6, and 8 below present an analysis of the *structural*, *technical*, *directional*, and *temporal* complexity. These tables highlight the specific complexities observed in the studies, offering an overview of the challenges posed by each dimension. Simultaneously, the review delved into strategies, effective or otherwise, undertaken to tackle the identified complexities. In Tables 3, 5, 7, and 9, the research focuses on presenting strategies and methods used to address the structural, technical, directional, and temporal complexity. These tables provide a summary of the insights into the diverse strategies employed by practitioners to mitigate the adverse impacts of complexity. By structuring and organizing the findings, the research aims to facilitate a clearer understanding of the complexities involved in Agile software project delivery. Simultaneously, the approaches in dealing with these complexities offer practical guidance to project managers and practitioners seeking to optimize and improve their project outcomes.

4.1 Structural complexity

We discovered two key structural complexity factors with Agile software development: complexity relating to functional setups or team structures (Table 2) and roles (Table 3). Table 2 identifies complexities relating to functional setups. This table identifies several themes on issues and challenges identified. There were several issues and challenges relating to these organizational structures, but page limitations restricted us from including them here. The complexities resulting from functional setups showed that teams were not producing the desired results and impacting on smooth project workflow. Our data suggests complexities resulting from these structures were due to a lack of understanding of the structures and how they must be effectively adopted.

Functional setups	How effective or ineffective	Adoption- appropriate or lack of
Product planning team (Business& SE)	Ineffective	Lack of appropriate adoption
Portfolio management teams	Ineffective	Lack of appropriate adoption
Development teams	Ineffective	Lack of appropriate adoption
Quality assurance team	Ineffective	Lack of appropriate adoption
Product management	Effective	Lack of appropriate adoption
Documentation team	Ineffective	Lack of appropriate adoption
Program management	Ineffective	Lack of appropriate adoption
Management	Ineffective	Lack of appropriate adoption
Knowledge management	Ineffective	Lack of appropriate adoption
Training	Ineffective	Lack of appropriate adoption
Customers	Ineffective	Lack of appropriate adoption

Table 2 Structural Complexity in Functional Setup

Table 3 shows the further structural complexity factor-roles. The roles were identified based on issues and challenges identified. There were several issues and challenges relating to these roles, but page limitations restrict us to include the specific details here. The complexities resulting from these roles showed that they were impacting smooth project workflow. Our data suggests complexities from these roles were due to a lack of understanding of the roles themselves, and how they are effectively adopted.

Role	How effective or ineffective	Adoption- appropriate or lack of
Product owner	Ineffective	Lack of appropriate adoption
Product manager	Ineffective	Lack of appropriate adoption
Usability engineer	Ineffective	
Team-leader (Scrum master)	Ineffective	
Developer	Ineffective	
Program manager	Ineffective	
Software engineer	Ineffective	
Systems engineer	Ineffective	
Product manager	Effective	Lack of appropriate adoption
Technical writer (documentation)	Ineffective	
Quality assurance engineer	Ineffective	Lack of appropriate adoption
Business stakeholder	Ineffective	

Table 2 Structural Complexity in Roles

4.1.1 Strategies to Deal with Structural Complexity

Also identified were the suggested strategies to deal with the issues and challenges, which we grouped under the two organizational structures –functional setups and roles. These strategies could potentially help minimize structural complexities if adopted appropriately.

Structure	Strategies
Functional (teams) setups	Cross-functional team; collective decision-making; collective ownership; knowledge sharing & knowledge repositories; continuous improvement, learning and skills development; self-organization &leadership, regular collaboration & communication; management support & training; information sharing platform, visual communication, collaboration tools transparent communication channels; continuous customer collaboration; employee empowerment; community of practice; Agile training; simplify processes; Implementation of Agile Project Management; make change transparent; Agile values; self-organize; face to face meeting.
Roles	Self-organization; clear role definition & responsibilities; role rotation; regular retrospectives; mentoring and coaching; Agile leadership role specialization; adaptive role definition; customer & Stakeholder engagement; effective communication; daily Stand-up meetings; user story refinement; leadership; just-in-time documentation; living documentation; pair documentation; documentation retrospective; dedicated on Site Customer; continuous customer collaboration; customer prioritization; customer feedback Loop; role-specific training; on-job training; training feedback and improvement; establish component team.

Table 3 Strategies for Structural Complexity

4.2 Technical and Technological Complexity

Table 5 below shows the technical and technological factors. Tasks were identified based on issues and challenges involved in Agile software development processes. Several issues and challenges relating to these roles, but page limitations restrict us to include them in detail. The complexity factors resulting from tasks showed that they were not effective at helping to produce the desired results or a smooth project workflow. These complexities suggest a lack of understanding of how they are effectively adopted and applied in practice.

Major Agile Task	How effective or ineffective	Adoption- appropriate or lack of
Design	Ineffective	Lack of appropriate adoption
Requirement	Ineffective	Lack of appropriate adoption
Planning	Ineffective	Lack of appropriate adoption
Sprint	Ineffective	
Portfolio	Ineffective	Lack of appropriate adoption

Table 4 Technical and Technological Complexity

4.2.1 Strategies to Deal with Technical and Technological Complexity

Table 6 presents task-related strategies identified that suggested approaches to deal with the issues and challenges with tasks related to technical and technological complexity. These strategies could help to avoid and minimize technical and technological complexities if adopted appropriately.

Major Agile Task	Strategies
Design	Design sprints; design review and validation; design thinking techniques; user-centred approach; design sprints; story mapping; design refinement, prototyping, user story splitting; parallel tracks.
Requirement	User story mapping; prioritization and backlog refinement; progressive elaboration; prototyping and mock-ups; iterative and incremental delivery; change management and

	flexibility; continuous communication and collaboration; branching strategies; security backlog; decompose requirement to smaller units; invest in learning to refine requirement; awareness of QR.
Planning	Iterative and incremental planning; backlog refinement, relative estimation & timeboxing; Agile release planning; capacity planning; visual planning tools; regular planning meetings; partial release.
Sprint	Pre-planning preparation; collaborative effort; user story estimation; capacity and commitment; define sprint goals; task breakdown; timeboxing, test automation
Portfolio	Resource planning; continuous alignment and adaptation; kanban portfolio; lean thinking; portfolio roadmap.

Table 5 Strategies for Technical/Technological complexities.

4.3 Directional Complexity

Table 7 shows the directional complexity factors. The key project tasks identified based on issues and challenges identified in the selected publications involved with Agile software. Several issues and challenges related to tasks causing directional complexity. The complexities resulting from these project tasks showed that they were not helping to produce the desired results for a smooth project workflow. Our data suggests complexities resulting from these tasks were due to a lack of understanding of how they are adopted and applied to projects in an Agile set up.

Major Agile Task	How effective or ineffective	Adoption- appropriate or lack of
Planning	Ineffective	Lack of appropriate adoption
Design Planning	Ineffective	Lack of appropriate adoption
Requirement	Ineffective	Lack of appropriate adoption
Short cycle (Sprint)	Ineffective	Lack of appropriate adoption
Portfolio	Ineffective	Lack of appropriate adoption

Table 6 Directional Complexity

4.3.1 Strategies to Deal with Directional Complexity

Table 8 provides project task-related strategies that were identified in the literature. These are suggested strategies to deal with task issues and challenges relating to task outputs, which other functional setups and roles may depend on. If adopted appropriately, these strategies could help avoid and minimize directional complexities.

Major Agile Task	Strategies
Planning	Iterative planning; regular planning reviews; risk assessment and mitigation; retrospectives; stakeholder Involvement; maintaining an assumption wiki page; lightweight strategies.
Design planning	Cross-functional collaboration; design sprints; design review and validation; Agile prototyping; design thinking techniques; iterative and feedback-driven design.
Requirements	Iterative and incremental delivery; change management and flexibility; continuous communication and collaboration; branching strategies; security backlog; use automated monitoring tools; validation of user requirements; requirements prioritization; user stories; lightweight strategies.
Sprint	Iterative requirement refinement; prototyping and user feedback; acceptance criteria; regular requirement reviews; iterative and incremental delivery.
Portfolio	Resource planning; continuous alignment and adaptation; continuous planning, economic prioritization (WSJF); minimum viable product (MVP).

Table 7 Strategies for Directional Complexity

4.4 Temporal Complexity

Table 9 shows the temporal complexity factors impacting Agile projects. These identified practice-related issues and challenges. Several issues and challenges related to distributed projects were found to be causing temporal complexity. The temporal complexities resulting from these project tasks showed

that they were not helping to produce the desired results for a smooth project workflow. Our review suggests temporal complexities are due to a lack of understanding of how the development practices are adopted and applied; particularly in a distributed agile structure.

Major Agile Task	How effective or ineffective	Adoption – appropriate or lack of
Schedule management practices	Ineffective	
Project coordination (Distributed Sites)	Ineffective	Lack of appropriate adoption
Planning practices	Ineffective	Lack of appropriate adoption
Project coordination	Ineffective	
Portfolio Planning (Resource Management)	Ineffective	Lack of appropriate adoption
Rules and Procedures Practices (Geographical sites)	Ineffective	
Communication Method Practices (distributed sites)	Ineffective	Lack of appropriate adoption
Organizational Practices (distributed sites)	Effective	Lack of appropriate adoption

Table 8 Temporal Complexity

4.4.1 Strategies to deal with Temporal Complexity

Table 10 below presents the suggested strategies to deal with the issues and challenges grouped under the seven temporal complexity factors. Our SLR findings suggest these strategies should help minimize temporal complexities if adopted appropriately.

Major Agile Task	Strategies
Schedule management practices	Schedule management; DSDM schedule management; team schedule; timeboxed iteration; prioritized backlog; release planning; estimation techniques; timeboxing; kanban board.
Project coordination (Distributed Sites)	Communication tools (Jira); improve team awareness and communication; collaboration; adjusting teamwork hours; team calendar.
Planning practices	Iterative planning; regular planning reviews; risk assessment and mitigation; retrospectives; stakeholder involvement; decompose planning to smaller units.
Portfolio Planning (Resource Management)	Planning prioritization, capacity planning; resource forecasting, resource allocation, flexible allocation; resource pools; resource contingency.
Rules and Procedures Practices (Geographical sites)	Iterative rule development; flexibility and adaptability procedures; Agile practices
Communication Method Practices (distributed sites)	Clear communication channels; document and record communication; collaboration tools and platforms; record meetings.
Organizational Practices (distributed sites)	Agile flexibility and adaptability; politics.

Table 9 Strategies for Temporal Complexity.

5 Discussion

Our findings suggest that while Agile methods may be regarded as mainstream, adopting, and applying these methods are not necessarily delivering the intended results and not always providing the seamless work practices desired. Layered functional setups such as in portfolio, program, and product management are not considered part of software development methods under the Agile umbrella. Our review, however, suggests that these structures are necessary elements for software development in current software ecosystems to best deal with project complexities. While scaled Agile methods do recognize these structures, they are highly comprehensive, requiring huge investments for successful adoption. Interestingly, having knowledge management and training structures, including recognizing customers as an integral part of the software development ecosystem, also enables a strategic approach for identifying, developing, and delivering high-value software. Our findings on roles suggest a lack of clarity on Agile roles and expectations in terms of required competency, skills, and role expectations. It appears that Agile roles are not necessarily clearly defined in practice for appropriate task accomplishments and the importance of their contributions to the overall product and project success, hence a source for project complexities.

The literature on technical and technological complexity including directional and temporal complexities suggest a lack of clarity on key tasks in an Agile software environments or ecosystems.

Many of complexities were found to be a result of Agile method tasks and related practices focusing on small-scale projects requiring small-sized teams. On the other hand, scaled Agile methods with comprehensive frameworks required teams of consultants and coaches for training and upskilling to effectively adopt successful Agile practices. A critical thing we can see from our literature findings on Agile complexities is that projects are just part of one element of a layered software development ecosystem. Hence, the wider software development ecosystem also involves business elements such as portfolio and product management. An Agile approach must define a highly integrated product development environment with clearly identified functional setups and roles and practices that integrate the development ecosystems for small as well as large-scale distributed and global software development.

Our findings confirm the appropriateness of Remington and Pollack's (2016) four categories of project complexities which appear to be also prevalent with the Agile approach to project delivery. The Project Management discipline clearly identifies complexity as an attribute of a project, and there is no guarantee that projects will be delivered successfully. Hence, uncertainty in projects can be related to complexity factors in the software development ecosystem. A number of strategies advocated were related to the four types of project complexities. For example, creating cross-functional teams comprising individuals with varied skills and viewpoints aims to dismantle barriers and to facilitate seamless cooperation. Consistent inter-team meetings and workshops appear to enhance mutual understanding and alignment. Early and continuous design sets the foundation for well-coordinated projects, where regular revisits and updates on design decisions based on new information and evolving requirements and pre-planning appear to be effective. Others such as embracing adaptive planning and iterative development help the team respond to changing requirements. Prioritising features based on customer feedback and business value keeps the project aligned with its goals. Other strategies such as timeboxing, where work is confined to fixed intervals, provide some predictability. Techniques like the use of a Kanban board allow teams to visualize and manage their workflow, enhance transparency and project delivery control. Buffering project schedules to accommodate unexpected delays ensures a more realistic timeline. Overall, the strategies identified suggest that Agile adoption introduces some factors of complexity. This may be due to a lack of understanding of what is needed for Agile adoption to produce the desired results and enable smooth project workflow.

6 Conclusion

Our review aimed to empirically examine and identify the complexity characteristics that impact Agile project delivery and to determine the approaches applied to moderate these effects. The results assist in reshaping our research questions for a major empirical investigation of complexity in Agile software delivery. While agility-based approaches and their dynamic ability to deliver projects yield greater flexibility and success in relation to highly structured and plan-driven approaches, there still exists a gap in appropriate adoption in relation to Agile's espoused abilities to deal with complexity. Our empirical analysis of Agile software literature structured around Remington and Pollack's (2016) four types of complexity reveal intricate complexities resulting from functional teams, organizational structures, roles, Agile tasks, and practice-related issues in distributed Agile projects. We find that there remains a lack of understanding of how these structures, practices, and roles are to be effectively adopted and applied in an Agile software delivery context. Our systematic review has shown that complexity exists and confirmed the four types have been relevant as a structure for complexity analysis. Although the literature in this review has identified a range of useful strategies that have been adopted in practice, there remains a need for further research to produce contemporary frameworks and models to better understand the complexity and related effects of Agile project delivery. Future research is to propose a framework to mitigate the identified gap in identifying complexity in Agile projects.

References

Agbejule, A., and Lehtineva, L. 2022. "The Relationship between Traditional Project Management, Agile Project Management and Teamwork Quality on Project Success," International Journal of Organizational Analysis (30:7), pp. 124–136. (https://doi.org/10.1108/ijoa-02-2022-3149).

Ahmed, M., and Shahzad, A. 2021. "Agile vs. Classical Software Development Project Management: Issues and Challenges," Computer Science (16).

Baccarini, D. 1996. "The Concept of Project Complexity—a Review," International Journal of Project Management (14:4), pp. 201–204. (https://doi.org/10.1016/0263-7863(95)00093-3).

Borba, J. C. R. de, Trabasso, L. G., and Pessôa, M. V. P. 2019. "Agile Management in Product Development," Research-Technology Management (62:5), pp. 63–67. (https://doi.org/10.1080/08956308.2019.1638488).

Cicmil, S., Cooke-Davies, T., Crawford, L., and Richardson, K. A. 2017. Exploring the Complexity of Projects: Implications of Complexity Theory for Project Management Practice, [Newtown Square, PA]: Project Management Institute.

Cicmil, S., Williams, T., Thomas, J., and Hodgson, D. 2006. "Rethinking Project Management: Researching the Actuality of Projects," International Journal of Project Management (24:8), pp. 675–686. (https://doi.org/10.1016/j.ijproman.2006.08.006).

Cobb, C. G. 2011. Making Sense of Agile Project Management Balancing Control and Agility, Hoboken, Nj, USA John Wiley & Sons, Inc.

Danilovic, M., and Browning, T. R. 2007. "Managing Complex Product Development Projects with Design Structure Matrices and Domain Mapping Matrices," International Journal of Project Management (25:3), pp. 300–314. (https://doi.org/10.1016/j.ijproman.2006.11.003).

Dikert, K., Paasivaara, M., and Lassenius, C. 2016. "Challenges and Success Factors for Large-Scale Agile Transformations: A Systematic Literature Review," Journal of Systems and Software (119:119), pp. 87–108. (https://doi.org/10.1016/j.jss.2016.06.013).

Garousi, V., Felderer, M., & Mäntylä, M. V. 2019) "Guidelines for including grey literature and conducting multivocal literature reviews in software engineering." Information and Software Technology (106), pp. 101-121.

Geraldi, J. G. 2008. "The Balance between Order and Chaos in Multi-Project Firms: A Conceptual Model," International Journal of Project Management (26:4), pp. 348–356. (https://doi.org/10.1016/j.ijproman.2007.08.013).

Ismail, M. F. bin, and Mansor, Z. 2018. "Agile Project Management: Review, Challenges and Open Issues," Advanced Science Letters (24:7), pp. 5216–5219. (https://doi.org/10.1166/asl.2018.11705).

Kitchenham, B. A. 2012. "Systematic Review in Software Engineering," Proceedings of the 2nd International Workshop on Evidential Assessment of Software Technologies - EAST '12 (44). (https://doi.org/10.1145/2372233.2372235).

Lee, G., and Xia, W. 2010. "Toward Agile: An Integrated Analysis of Quantitative and Qualitative Field Data on Software Development Agility," MIS Quarterly (34:1), p. 87. (https://doi.org/10.2307/20721416).

Mishra, D., and Mishra, A. 2011. "Complex Software Project Development: Agile Methods Adoption," Journal of Software Maintenance and Evolution: Research and Practice (23:8), pp. 549–564. (https://doi.org/10.1002/smr.528).

Mitleton-Kelly, Eve. "Organisation as Co-Evolving Complex Adaptive Systems." ScienceDirect, vol. 21, no. 12, 1 Sept. 1997. Accessed 24 Aug. 2023.

Ralf Müller, Geraldi, J., and J. Rodney Turner. 2007. "Linking Complexity and Leadership Competences of Project Managers," Linking Project Complexity and Leadership Competences of Project Managers (21:6).

Remington, K., and Pollack, J. 2016. Tools for Complex Projects, Routledge. (https://doi.org/10.4324/9781315550831).

Remington, K., Zolin, R., and Turner, R. 2009. "A Model of Project Complexity: Distinguishing Dimensions of Complexity from Severity," A Model of Project Complexity: Distinguish Dimensions of Complexity from Severity (34:22).

Snowden, David. "Multi-Ontology Sense Making: A New Simplicity in Decision Making." Journal of Innovation in Health Informatics, vol. 13, no. 1, 1 Mar. 2005, pp. 45–53, https://doi.org/10.14236/jhi.v13i1.578.

Stacey, Ralph D. "The Science of Complexity: An Alternative Perspective for Strategic Change Processes." Strategic Management Journal, vol. 16, no. 6, 1995, pp. 477–495, https://doi.org/10.1002/smj.4250160606. Accessed 21 Mar. 2019.

Stare, Aljaž. "Agile Project Management – a Future Approach to the Management of Projects?" Dynamic Relationships Management Journal, vol. 2, no. 1, 30 May 2019, pp. 43–53, https://doi.org/10.17708/drmj.2013.v02n01a04.

Thesing, Theo, et al. "Agile versus Waterfall Project Management: Decision Model for Selecting the Appropriate Approach to a Project." Procedia Computer Science, vol. 181, no. 1, 2021, pp. 746–756. ScienceDirect, https://doi.org/10.1016/j.procs.2021.01.227.

Wohlin, C. 2014. "Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering," Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering - EASE '14 (55:38). (https://doi.org/10.1145/2601248.2601268).

Copyright