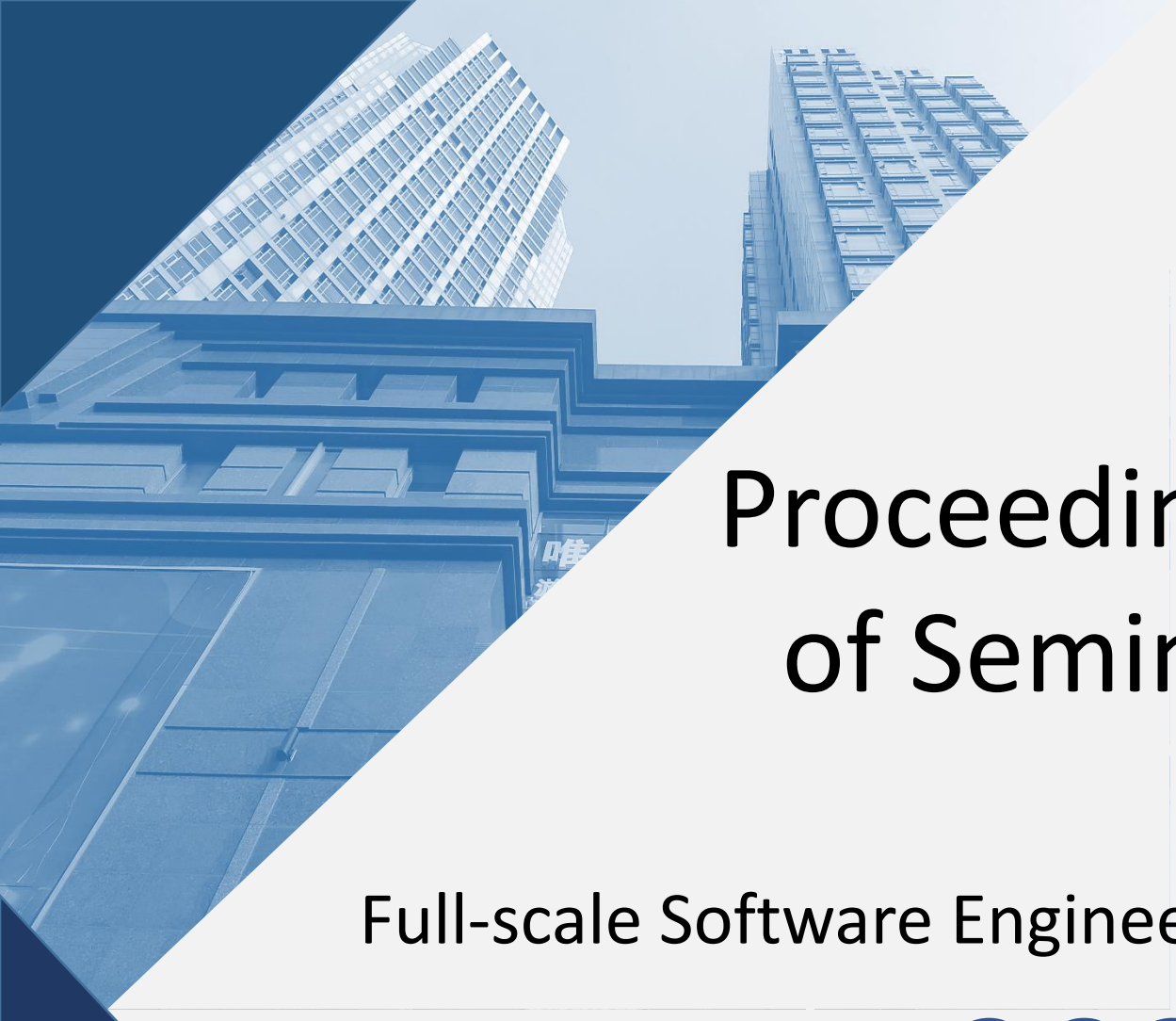


Proceedings of Seminar

Full-scale Software Engineering

2026

Editors: Horst Lichter
Selin Aydin
Alex Sabau
Ada Slupczynski

SWC Software Construction | RWTH AACHEN UNIVERSITY

Table of Contents

Temporal Aspects of Software Modernizations

Sibelius Kruse
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
sibelius.kruse@rwth-aachen.de

Georg Thiesen
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
georg.thiesen@rwth-aachen.de

ABSTRACT

Time plays a central role in software engineering, particularly in the scheduling of software projects and the duration of development phases. In contrast, the role of time in software modernization remains underexplored, with temporal aspects treated as secondary and supported only by fragmented empirical evidence. In today's scientific literature on software modernization, technical, organizational, and operational aspects are the primary focus. To fill this gap, this paper examines software modernization through a temporal lens and provides a structured overview of how time is treated in the current academic literature. Therefore, a systematic literature review of software modernization studies is conducted with a focus on temporal aspects. The analysis highlights four recurring themes: reported durations of modernization projects, descriptions of the length and sequencing of specific modernization phases, discrepancies between planned and actual project durations, and time lags between modernization activities and the realization of benefits. The contribution of this paper is a consolidated map of time-related information in the current software modernization literature. It presents the existing evidence base and confirms the limited quantitative treatment of temporal characteristics in modernization projects. By clarifying the temporal characteristics of software modernization, the results lay a foundation for better research on planning practices. This allows for more grounded evaluations of the associated costs, risks, and benefits.

Categories and Subject Descriptors

D.2 [**Software**]: Software Engineering; D.2.9 [**Software Engineering**]: Management—*productivity, programming teams, software configuration management*

Keywords

Software Modernization, Systematic Literature Review, Project Schedule, Project Duration, Software Modernization Phases

SWC Seminar 2018/19 RWTH Aachen University, Germany.

1. INTRODUCTION

Time is a central concern in the planning, coordination, estimation, and evaluation of software engineering projects. In industry, organizations routinely prioritize short-term benefits, even when this entails higher long-term costs. Technical debt captures this trade-off: teams accelerate delivery in the short term at the expense of slower and more costly change later [1]. Over years, accumulated debt, aging technologies, and evolving requirements produce portfolios of legacy systems that are difficult and costly to evolve. Software modernization has emerged as a central strategy for dealing with these systems. For software modernization the following definition is used: "Software modernization attempts to evolve a legacy system, or elements of the system, when conventionally evolutionary practices, such as maintenance and enhancement, can no longer achieve the desired system properties" [2]. In this paper, the role of project schedules, and timelines in software modernization is investigated. To provide context, the literature on different temporal aspects in software modernization is compared to studies on this topic in the area of general software engineering and software evolution.

1.1 Background

Existing overviews of software modernization and related topics largely organize findings along technical, architectural, organizational, operational, or practice-oriented dimensions rather than explicitly foregrounding time [3–5]. Nevertheless, these works contain extensive temporal observations: Assunção et al. [3] discuss time as the goal of a faster time to market, as process duration, or in the context of maintenance effort. Freire et al. [5] emphasize temporal trade-offs between short-term and long-term priorities, practices for creating time such as negotiating deadlines, iteration frequency, and system age. Ponnusamy and Eswararaj [4] address time in strategic trade-offs between quick wins and long-term value, phased versus big-bang modernization, downtime, and future time costs of technical debt.

Individual case studies foreground temporal issues more explicitly, for example a Jordanian study that models downtime in detail during the modernization of a classical data center to the cloud [6], and Fonseca et al.'s [7] industrial report that provides an explicit migration timeline with step durations and sequencing. However, these single-context studies do not support generalizable claims about time in software modernization. No overview was found that explicitly synthesizes what is known about the temporal characteristics of software modernization; temporal considerations remain

implicit, fragmented, and dispersed across surveys, strategy guides, and case studies, without a consolidated, empirically grounded account of timelines, phases, and temporal risks.

Conversely, temporal aspects are deeply embedded in mainstream software engineering practice. Phase-structured Software Development Lifecycle (SDLC) models have been extensively documented [8, 9]. Across such models, work is organized into explicit stages and iterations that can serve as natural units for measuring duration and tracking progress. On top of these SDLC models, a substantial body of work studies how to estimate development effort [10, 11], which is closely tied to cost and time estimation because personnel costs dominate software project budgets. At a finer granularity, temporal patterns of code evolution are used to track project progress [12, 13]. Finally, forecasting approaches move beyond point estimates to simulate entire project timelines [14]. Collectively, these strands treat time as a first-class concern, structuring processes into temporal phases, providing methods to estimate and forecast effort over time, and analyzing how temporal development patterns affect quality of software engineering projects.

Temporal analyses also play a central role in research on software evolution. There, open-source software repositories are used to reconstruct long-term trajectories of size, complexity, and change activity from time-stamped repositories, and forecast future evolution [15], or to track how code characteristics evolve across successive releases [16]. Furthermore, code evolution and calendar time are analyzed using historical repositories, like the Unix repository [17]. To summarize, the scientific literature contains a rich knowledge base on different temporal aspects in the context of software engineering and software evolution. On the other hand, the body of knowledge regarding time schedules in software modernization projects seems to be underdeveloped.

1.2 Study Aim and Research Questions

To investigate this gap further, this study adopts an explicit temporal lens to analyze software modernization. To this end, a systematic literature review is conducted, focusing on modernization studies that explicitly report temporal aspects. In this review, the focus lies on overall project durations of modernization efforts, the length and sequencing of modernization phases, discrepancies between planned and actual durations and reported Return on Investment (ROI). This paper focuses on identifying, extracting, and classifying the temporal information present in the scientific literature, to lay the foundation for further analysis in future research.

Accordingly, this systematic literature review aims to specifically answer the following questions.

- **RQ1:** What durations and high-level temporal structures are *reported* in existing studies on software modernization?
- **RQ2:** How do existing studies describe the phases in software modernization, and which temporal characteristics are associated with these phases?
- **RQ3:** To what extent do existing studies report planned modernization durations, and what discrepancies are observed where such data are available?
- **RQ4:** When do investments in software modernization break even with the modernization benefits?

This paper's main contribution is a consolidated empirical map of time-related information reported in existing software modernization studies, covering project durations, phase structures, benefit lags, and schedule discrepancies.

The remainder of this paper is organized as follows. In Section 2, the procedure of the systematic literature review is detailed. The section is structured in three main parts: the description of the search strategy, the explanation of the study selection process, and the summary of extracted data. In Section 3 the results of the literature review are reported. The section is structured into three subsections: the report of results with respect to the research questions, the description of further time-related information in software modernization, and the discussion section.

2. METHOD

A systematic literature review of temporal aspects in software modernization is conducted, based on Kitchenham's guidelines for evidence-based software engineering [18]. The review was carried out in November and December 2025. The first step, retrieving all potentially relevant articles, is described in Section 2.1. In Section 2.2, the process for identifying the relevant papers from an initial corpus of several hundred results is outlined. Finally, Section 2.3 explains how information was extracted from the final selection of papers. For this review, a broad scope on software modernization is defined. Seacord et al.'s [2] understanding of modernization is followed including both empirical and conceptual contributions. Case studies, industrial experience reports, and surveys are treated as primary sources, as long as they contained some form of temporal information and originated from peer-reviewed publications.

2.1 Search Strategy

The first step was to compile a comprehensive list of keywords related to software modernization with a focus on time. For this purpose, three large language models (Perplexity Pro [19], Gemini 2.5 Pro [20] and Claude Opus [21]) are used. The prompt was:

> "In software project management the aspect of time is well studied. multiple methods for time estimation exist, multiple phase models of software engineering exist and there are studies that link success probability of a project with its length. Some authors claim, that software modernization projects and general software engineering projects differ, e.g. in the applicability of time estimation methods. In our paper we want to know: What does the scientific literature know about the aspect of time in software modernization projects? I want to perform a systematic literature review and want you to generate keywords that I will use during the review."

Some fundamental keywords, such as "month" and "year," were added, resulting in an initial list of more than 400 keywords. From this set, keywords that were overly specific (e.g., "strangler fig pattern"), overly broad (e.g., "challenge"), outside the scope of the topic (e.g., "stakeholder engagement"), or redundant were removed. After this cleaning process, the list comprised 98 unique keywords. In the next step, each keyword was assigned to one of two groups, time-related

Table 1: Time-related keywords

Subcategory	Keywords
Estimation & measurement	time estimation, schedule estimation, duration estimation, project duration, project length, release duration, development time, person-months
Scheduling & planning	project scheduling, software project scheduling, schedule planning, project planning, milestone planning, release planning, temporal planning
Schedule management	schedule management, scheduling optimization, time management, schedule adherence
Delays & overruns	schedule delay, project delay, time overrun, schedule overrun, project overrun, schedule variance, schedule slippage, delay factors
Deadlines & delivery	deadline, on-time delivery, delivery time
Time-to-market	time-to-market
Temporal concepts	temporal aspects, time constraints
Time units	month, week, year

Table 2: Modernization-related keywords

Subcategory	Keywords
Strategies & approaches	incremental modernization, phased modernization, iterative modernization, continuous modernization, agile modernization, big bang, rewrite, rollout strategy, incremental implementation, migration path
Migration types	system migration, software migration, code migration, programming language migration, platform migration, database migration, infrastructure migration, cloud migration, application migration, application migration to cloud, legacy to cloud
Architectural transformation	architecture modernization, architecture refactoring, architecture transformation, re-architecting, application re-architecting, monolith to microservices, monolithic to microservices, legacy to SOA
Legacy system treatment	legacy modernization, legacy code modernization, legacy transformation, legacy system, legacy software, legacy evolution, system replacement, system wrapping, strangler pattern
Reengineering & renovation	software reengineering, system reengineering, system renovation, software renovation, system remodularization, code transformation
Evolution & maintenance	software evolution, software maintenance, continuous improvement, software aging, system decay
Technical debt	technical debt, design debt, code debt, technical debt reduction
Planning & readiness	modernization readiness, migration planning, transition planning
Change impact	breaking changes
Contextual concepts	traditional software development vs. modernization

or modernization-related. This classification supported the construction of the research queries.

IEEE Xplore and the ACM Digital Library were selected because of their extensive collection of peer-reviewed software engineering research papers. Their coverage includes most major conferences (ICSE, SANER, ICSME, etc.) and journals relevant to modernization and software project management. Multiple queries were necessary because both digital libraries provide different search options. In particular, ACM requires separate queries for journals and conference proceedings. Nevertheless, the core is the same. First, all keywords of one group, "time-related" or "modernization-related" where connected with the boolean operator OR. Then, these two groups where connected with an AND. This approach ensured that each paper is somehow correlating with modernization and also time aspects. The queries were only applied to abstracts, titles and keywords given by the author, which is covered in the IEEE database under "metadata". For ACM this is not the case, so it was constructed manually, with the following schema:

> Abstract:(keyword1 OR keyword2 OR...) OR Title:(keyword1 OR keyword2 OR...) OR Keywords:(keyword1 OR keyword2 OR...) AND (...

To exclude overly outdated publications, the publication period was restricted to studies published between 2000 and 2025. For IEEE Xplore, filters were applied to include only journal and conference publications and to limit results to specific publication topics (see Table 3). Likewise, in the ACM Digital Library, journal articles and conference proceedings were selected. The employed time-related keywords are listed in Table 1, modernization-related keywords in Table 2. Each search query followed the structure described below.

> Metadata: Modernization related keywords **AND** Metadata: Time related keywords **AND** Filters

The database searches with these queries returned 869 candidate studies.

2.2 Study Selection

Only a subset of the retrieved studies contained information relevant to the research topic. Many publications did not

Table 3: Database-specific filters

Database	Filters
IEEE Xplore	Publication Topics: Software Development; Software Engineering; Software Maintenance; Software Architecture; Project Management; Software Quality; Refactoring; Reverse Engineering; Technical Debt; Development Time Publication years: 2000–2025
ACM Digital Library (journals)	Publication type: Journals Publication years: 2000–2026
ACM Digital Library (proceedings)	Publication type: Proceedings Publication years: 2000–2026

Table 4: Literature Inclusion Criteria

Inclusion Criteria
The literature focuses on software modernization and reports time-related information relevant to at least one of the research questions of this paper.
The paper has a minimum length of three pages.
The study is peer-reviewed.
The paper is written in English.
The literature was published in the 21st century.

address temporal aspects, while others focused on unrelated domains. The inclusion criteria for the literature are listed in Table 4. For this step of extracting the relevant subset of the papers, a new list of time-related keywords was created and combined using logical OR operators. This query was applied to the full text of each paper. The query used in this step was as follows:

> day OR week OR month OR year OR person-month OR man-month OR duration OR phase OR length OR took OR take OR last OR "short term" OR short-term OR "long term" OR long-term OR delay OR schedule OR minute OR time OR hour

96 papers were identified to contain potentially relevant information by reading sections containing keywords. In the last selection step, relevant data were extracted from all remaining papers if they matched the definition of software modernization used in this paper. Finally, 27 papers, that contain relevant information regarding at least one of the research questions were included in the analysis.

2.3 Data Extraction

For each included paper, information was extracted into a structured table. The recorded attributes included the type of study, the reported system size, and the type of modernization undertaken. With respect to the core focus of this review, all available temporal information was extracted, including overall project duration and the durations or descriptions of individual phases or steps. Additionally, information on downtime, delays, and the timing of reported benefits was captured where available. Studies that only contained qualitative or vague temporal descriptions, e.g., "several months", "over years", and papers that provided only partial temporal information were kept for further analysis. Missing temporal information were noted, and the available information was included, to not reduce the number of papers too much. Time-related information that did not match any of the proposed data extraction categories was collected separately to provide a holistic view on temporal information in software modernization. An overview of this data is presented in Section 3.2.

3. RESULTS

At the conclusion of the paper selection process, the 27 included research articles exhibited a variety of evidence types. A substantial portion of the studies report industrial case studies in which researchers analyzed real-world software migration or modernization projects [22, 23]. In a few instances, these case studies were used to empirically evaluate a proposed modernization framework, which represented the primary contribution of the respective papers [24]. In contrast, some studies introduce conceptual frameworks or approaches without providing any form of empirical validation [25]. Additionally, Anderson and Bishop describe a re-platforming effort in the form of an experience report, offering practitioner-oriented insights rather than a formal evaluation [26]. The extracted information appeared in several forms: explicit quantitative data expressed as concrete values, qualitative statements that did not report numerical measures but nevertheless conveyed relevant temporal aspects, and implicit information that could be inferred from the described contexts and statements.

This section presents the results of the systematic literature review. It begins with an overview of the explicitly reported quantitative data, followed by an analysis of the research questions and how the findings address them. The final part completes the mapping of temporal aspects of software modernization by presenting additional time-related information that, while not directly answering the research questions, contributes to a more comprehensive understanding of modernization with respect to time. An overview of the number of categories of extracted data is provided in Figure 1. Of the 27 selected studies, 20 report only one or two of the extracted data categories, while only 2 provide data for all 4 categories. Table 5 provides an overview of which category is mentioned in which paper and how big the coverage of the categories is across all included papers. Modernization durations are summarized in Figure 2. The median duration lasted 6 month, with the shortest modernization taking 7 weeks, and the longest modernization taking 5 years.

Table 5: Information Coverage

Attribute	#Reporting Papers (n=27)	Coverage (%)	Sources
Modernization duration/effort	16	59.3%	[22, 23, 26–39]
Planned modernization duration/effort	7	29.6%	[27, 31, 33–35, 39–41]
Modernization phases	18	66.7%	[24, 25, 27, 29–32, 35–37, 40, 42–48]
Modernization phase durations	10	37.0%	[25, 27, 29–32, 36, 43, 45, 46]
Break-even-point of invested time in modernization and gained time	2	7.4%	[27, 41]

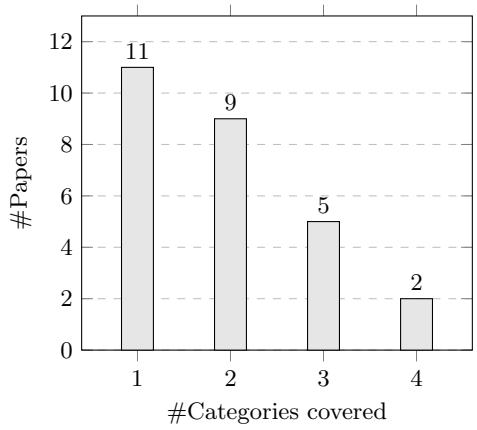


Figure 1: Papers by number of temporal categories covered.

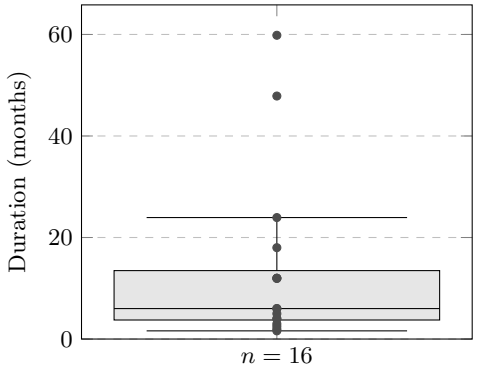


Figure 2: Modernization durations. Median: 6 months (IQR: 3.75–13.48).

3.1 Results with Respect to the Research Questions

The following sections present the results with respect to the individual research questions. Only explicitly reported data that addresses the corresponding research questions is presented, without interpretation beyond the observed evidence. Variations in the availability and level of detail of time-related information are addressed in the discussion (Section 3.3).

3.1.1 RQ1

The first aspect of software modernization examined in this paper is the extent to which modernization durations are reported and how the stated durations relate to planned timeframes. Among the reviewed studies, 16 out of 27 make at least a brief reference to the duration or effort of the modernization process. However, the way temporal information is reported varies considerably across studies. The most common reporting format is calendar-based duration, such as weeks or months, which is used by 15 studies. These reports typically provide a single aggregated duration for the entire modernization effort, for example stating that a project was completed "within six months" [35]. In addition to calendar-based durations, two studies report effort as a percentage of working time invested in modernization activities [39, 41]. One study provides only a vague temporal reference, indicating that the migration process took "several years" [40].

A smaller subset of studies reports effort using person-time units, such as person-hours or developer work months. For instance, one study reports an effort of 42 developer work months, where one unit corresponds to 132 person-hours [22]. Two studies additionally report the number of lines of code that were refactored [23], with Xu [34] further specifying the average number of lines of code processed per day. Such metrics are, however, only applicable to studies focusing on code-level modernization. Xu et al. [37] also report effort in terms of time per iteration, without providing information on the total number of iterations. Planned modernization durations are reported far less frequently. Only 7 of the 27 studies specify planned durations, primarily using calendar-based measures, with one exception that reports effort in person-days [33]. Notably, [41] is the only study that reports either planned or actual duration, but not both, resulting in no overlap between planned and observed timeframes.

3.1.2 RQ2

In addition to the overall duration of modernization, this paper also examines the individual phases of the modernization process, as well as whether such phases are reported at all. 18 of the 27 research articles explicitly describe a division of the modernization process into phases. However, the identified phases vary considerably across studies, partly due to differing modernization contexts, such as cloud migration [38], code refactoring [39], or migration to a new programming language [23]. "Reverse Engineering" is reported in six studies; for example, one study describes how "the legacy system was reverse engineered" to retrieve data models and functional specifications [46]. In four of these

cases, this phase is followed by "Re-specify" and "Re-design," and in two cases by "Forward Engineering." Other phases, such as implementation, testing, and development, are also mentioned multiple times, although their ordering differs between studies. Only one paper reports a detailed effort distribution across modernization phases [32]. The remaining 10 studies provide only partial temporal information for individual phases. In one case, effort is even reported using different measurement units: phase "A. Familiarization with the product and First Measurements" is described as taking approximately two weeks of full-time work, whereas phase "B. Definition of the Performance Engineering Plan" required 527 person-hours [36].

3.1.3 RQ3

The available data on planned modernization durations are even more sparse than the data on actual modernization durations. In this literature review, 7 out of 27 studies reported any information about the planned modernization duration. Notably, the quality of the reported information differed between studies. Only two studies reported both the planned modernization, as well as the amount of effort that was planned [33, 41]. 6 studies reported concrete planned durations ranging from 56 days to 10 months [31, 33–35, 39, 41]. On the other hand, one study only provided qualitative descriptions of two case studies, describing the first case study as finished on schedule and the second case study as taking longer than expected [40]. Among the studies reporting quantitative data on the planned modernization durations four studies completed the modernization in time [31, 33–35]. One study that proposes a systematic approach to refactoring and compares it to traditional refactoring approaches, reported that the actual duration of the modernization of one team using traditional refactoring approaches doubled the planned modernization duration (planned: 6 months, actual: >12 months) [39]. The paper does not provide further data on the planned duration or effort of other teams. One study only reported the planned modernization duration, but did not provide information on the actual modernization duration [41].

Summarizing the data on RQ3, only 22.2% of the included studies report quantitative data on planned modernization duration or effort. Some studies only report partial data or qualitative descriptions. Given the number of studies included in the review, the low percentage of studies reporting on planned duration or effort and the heterogeneity of modernization projects, no generalizable claims about correlations between required and planned duration of modernizations can be made.

3.1.4 RQ4

On the question of the time break-even points of modernization costs and benefits are achieved, only two papers provided concrete data. Rostkowycz et al. describe in [27] the long-term effects of software redocumentation. As redocumentation can be categorized as a software maintenance activity and the project was a dedicated effort, it fits the used definition for software modernization. The paper discusses a case study in which 83% of the software was redocumented. Reduced software maintenance costs outweighed the redocumentation effort 1.5 years after the start of the project. Jepsen et al. [41] describe a case study on the adoption of a software product line approach. They found that the additional effort paid off in reduced effort for multiple software projects in 8 months. With only 2 out of 27 studies reporting break-even points of modernization effort and benefits, a gap in the scientific literature has been identified.

3.2 Completeness of Time-Related Reporting in Software Modernization

In addition to the time-related aspects addressed by the research questions, a small number of studies report further temporal considerations related to software modernization. These aspects are mentioned only sporadically across the reviewed literature but contribute to a more comprehensive view of how time is discussed in the context of modernization. Some studies refer to short-term versus long-term temporal effects without providing concrete durations. For example, Stochel et al. [29] describe refactoring activities as having immediate effects while potentially introducing long-term consequences related to technical debt. Other studies mention time-related outcomes such as time-to-market. In particular, one study reports that modernization of legacy application systems is associated with improvements in time-to-market, although no quantitative timing information is provided [42]. Conversely, challenges affecting modernization efforts, including technical, financial, and organizational factors, are reported as influencing timelines qualitatively rather than quantitatively [42]. Finally, some studies relate modernization to maintenance effort over time. Mishra et al. [48] emphasize the role of re-architecting legacy systems in reducing maintenance activities, which are described as consuming a substantial portion of the software life-cycle, again without explicit temporal measurements. Overall, these observations highlight additional temporal dimensions discussed in the literature, while also underscoring the predominantly qualitative and fragmented nature of time-related reporting in software modernization studies.

3.3 Discussion

Summarizing the literature review, it is evident, that temporal information about software modernization is sparse and scattered. Only 16 out of the 27 identified studies provide information about the duration of the modernization. Given the number of different kinds of modernizations and the difference in the size of systems, the given sample size cannot be used for any generalizable statements. Furthermore, it was observed that the emergence of automated modernization tools, e.g., (semi-)automated translation [49] or refactoring [50] creates a distinct temporal profile compared to manual efforts. This suggests that future research must distinguish between tool-assisted and human-centric modernization when establishing effort-estimation models, as the temporal patterns of these approaches are not directly comparable. One possible explanation for the lack of temporal information in software modernization, compared to software evolution or software maintenance, might be the predominance of agile, iterative software engineering practices. When technical debt is payed continuously, this falls in the category of maintenance or continuous modernization, instead of the definition of software modernization used in this paper. Another possible reason is that modernization reports are rarely peer-reviewed and are often published only by the companies that were modernized.

With only 10 identified studies, that report on phase durations, more granular information is even more rare. This is

problematic, as more granular temporal information would allow for more targeted correlations between time and modernization success indicators. So, extracting prescriptive actions to improve modernization outcomes using temporal information is hardly possible. Amplifying the issue of limited development phase duration data, it has to be acknowledged that different kinds of modernizations require different development phases. This divides the available data on modernizations into even smaller buckets, making valid statements difficult. Modernization categorizations as described in [3] could provide a standard way of describing modernizations. A shared categorization framework for modernizations would facilitate comparison between different modernizations, although the sizes of the modernizations need to be taken into consideration separately. Focusing on the studies that provided data on the duration of modernizations, the common absence of data on planned duration precludes a meaningful comparison between actual performance and intended timelines, making it impossible to establish a benchmark for project success. A further problem identified during the literature review, is the issue of missing effort data. Only 5 of the 16 papers, that report on duration or effort provide data on the effort put into the modernization. Self-Evidently, duration and number of involved staff allow for better predictions of effort, than predictions based on duration alone. With additional data on the number of people working on the modernization, better effort estimations could be made, using historical modernization size to effort correlation data.

It is noteworthy that most studies focus on technically oriented modernization processes, such as migrating from one programming language to another [42] or re-architecting a legacy system [24]. However, Rostkowycz et al. [27] present a case study on re-documentation that was conducted with sufficient scope and depth to be classified as a modernization effort and was therefore included in this review.

3.4 Limitations

This paper outlines the state of the art in software modernization with respect to time. However, several limitations related to the literature review process must be considered. First, the keywords provided by the LLMs used to construct the database queries are not guaranteed to be exhaustive. An attempt was made to capture all relevant keywords by prompting multiple LLMs and by manually adding additional keywords to the resulting list.

More generally, data on time-related aspects of software modernization are very sparse. While some studies mention time-related facts, these are rarely comparable across studies. This significantly complicates the process of synthesizing the available information and, in many cases, the data are insufficient to extract meaningful conclusions.

4. CONCLUSION

This study addressed the gap of missing temporal characteristics of software modernization in software engineering research by providing a systematic literature review of the current state of research. While time is a first-class concern in general software project management, the findings reveal that in the specific domain of modernization, temporal data remains implicit, fragmented, and under-reported. Through an analysis of 27 primary studies, a map of the current landscape of modernization durations, phase structures, and schedule discrepancies was created. The results indicate a significant variance in project timelines, that could be explained by the specific modernization project's scope, the size of the legacy system or the size of the modernization team among other possible factors. Specifically, the frequent omission of planned durations prevents researchers and practitioners from objectively assessing project efficiency or identifying systematic causes of schedule overruns. The review process demonstrated that software modernization projects, are not always titled as such. Projects, that match the definition of modernization used in this paper can be found as "extraordinary maintenance" [32], "extreme maintenance" [23] or "technical debt mitigation" [29] in the context of software evolution or maintenance. By employing a broad keyword strategy beyond the term "modernization" alone, a more diverse set of case studies was successfully captured that would otherwise have been excluded. Unifying the used definitions in the area of software modernization could enlarge the body of literature used to further the knowledge on software modernization. Future research is needed to gather more data on schedule adherence of modernization initiatives and their reasons, effort estimation, as well as ROI estimations for software modernization. Improved knowledge in these areas could lead to cost reductions and simplify the project management of software modernizations.

References

[1] C. Izurieta and J. M. Bieman. "A Multiple Case Study of Design Pattern Decay, Grime, and Rot in Evolving Software Systems". *Software Qual J* 21.2 (2013), pp. 289–323. DOI: 10.1007/s11219-012-9175-x.

[2] R. C. Seacord, D. Plakosh, and G. A. Lewis. *Modernizing Legacy Systems: Software Technologies, Engineering Process and Business Practices*. USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

[3] W. K. G. Assunção, L. Marchezan, A. Egyed, and R. Ramler. *Contemporary Software Modernization: Perspectives and Challenges to Deal with Legacy Systems*. 2024. DOI: 10.48550/arXiv.2407.04017. arXiv: 2407.04017 [cs].

[4] S. Ponnusamy and D. Eswararaj. "Navigating the Modernization of Legacy Applications and Data: Effective Strategies and Best Practices". *Asian Journal of Research in Computer Science* 16.4 (2023), pp. 239–256. DOI: 10.9734/ajrcos/2023/v16i4386.

[5] S. Freire, N. Rios, B. Gutierrez, D. Torres, M. Mendonça, C. Izurieta, et al. "Surveying Software Practitioners on Technical Debt Payment Practices and Reasons for Not Paying off Debt Items". *Proceedings of the Evaluation and Assessment in Software Engineering*. Trondheim Norway, 2020, pp. 210–219. DOI: 10.1145/3383219.3383241.

[6] R. M. H. Hamad and M. Al Fayoumi. "Modernization of a Classical Data Center (CDC) vs. Adoption in Cloud Computing Calculate Total Cost of Ownership for Both Cloud and CDC - Jordanian Case Study". *2018 International Arab Conference on Information Technology (ACIT)*. 2018, pp. 1–8. DOI: 10.1109/ACIT.2018.8672686.

[7] S. C. da Fonseca, C. D. N. Porto, E. F. C. da Cruz, W. A. d. O. Neto, G. Tordin, and R. Trindade. "Migrating Legacy Systems: An Experience Report on the Industrial Environment". *Proceedings of the XXIII Brazilian Symposium on Software Quality*. SBQS '24. New York, NY, USA, 2024, pp. 452–459. DOI: 10.1145/3701625.3701630.

[8] M. Kuhrmann, P. Diebold, J. Münch, P. Tell, V. Garousi, M. Felderer, et al. "Hybrid Software and System Development in Practice: Waterfall, Scrum, and Beyond". *Proceedings of the 2017 International Conference on Software and System Process*. ICSSP '17. New York, NY, USA, 2017, pp. 30–39. DOI: 10.1145/3084100.3084104.

[9] N. B. Ruparelia. "Software Development Lifecycle Models". *SIGSOFT Softw. Eng. Notes* 35.3 (2010), pp. 8–13. DOI: 10.1145/1764810.1764814.

[10] M. Usman, E. Mendes, F. Weidt, and R. Britto. "Effort Estimation in Agile Software Development: A Systematic Literature Review". *Proceedings of the 10th International Conference on Predictive Models in Software Engineering*. PROMISE '14. New York, NY, USA, 2014, pp. 82–91. DOI: 10.1145/2639490.2639503.

[11] L. L. Minku and X. Yao. "How to Make Best Use of Cross-Company Data in Software Effort Estimation?" *Proceedings of the 36th International Conference on Software Engineering*. ICSE 2014. New York, NY, USA, 2014, pp. 446–456. DOI: 10.1145/2568225.2568228.

[12] O. Alam, B. Adams, and A. E. Hassan. "Measuring the Progress of Projects Using the Time Dependence of Code Changes". *2009 IEEE International Conference on Software Maintenance*. 2009, pp. 329–338. DOI: 10.1109/ICSM.2009.5306313.

[13] S. Basri, N. Kama, F. Haneem, and S. A. Ismail. "Predicting Effort for Requirement Changes during Software Development". *Proceedings of the 7th Symposium on Information and Communication Technology*. SoICT '16. New York, NY, USA, 2016, pp. 380–387. DOI: 10.1145/3011077.3011096.

[14] A. S. Bazhenov and V. M. Itsykson. "Forecasting Software Development Project Characteristics Using Meta-Modeling". *Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia*. CEE-SECR '13. New York, NY, USA, 2013, pp. 1–8. DOI: 10.1145/2556610.2556614.

[15] I. Herraiz. "A Statistical Examination of the Evolution and Properties of Libre Software". *2009 IEEE International Conference on Software Maintenance*. 2009, pp. 439–442. DOI: 10.1109/ICSM.2009.5306299.

[16] S. M. Alnaeli, A. D. A. Taha, and T. Timm. "On the Prevalence of Function Side Effects in General Purpose Open Source Software Systems". *2016 IEEE 14th International Conference on Software Engineering Research, Management and Applications (SERA)*. 2016, pp. 141–148. DOI: 10.1109/SERA.2016.7516139.

[17] D. Spinellis. "A Repository with 44 Years of Unix Evolution". *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. 2015, pp. 462–465. DOI: 10.1109/MSR.2015.64.

[18] B. Kitchenham. *Procedures for Performing Systematic Reviews*. Technical Report TR/SE-0401. Keele, UK: Keele University, 2004, pp. 1–26.

[19] Perplexity AI. *Perplexity*. 2026. URL: https://www.perplexity.ai (visited on 01/23/2026).

[20] Google. *Google Gemini*. 2026. URL: https://gemini.google.com (visited on 01/23/2026).

[21] Anthropic. *Claude*. 2026. URL: https://claude.ai (visited on 01/23/2026).

[22] A. Martini, E. Sikander, and N. Medlani. "Estimating and Quantifying the Benefits of Refactoring to Improve a Component Modularity: A Case Study". *2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 2016, pp. 92–99. DOI: 10.1109/SEAA.2016.48.

[23] J. Brant, D. Roberts, B. Plendl, and J. Prince. "Extreme Maintenance: Transforming Delphi into C#". *2010 IEEE International Conference on Software Maintenance*. 2010, pp. 1–8. DOI: 10.1109/ICSM.2010.5609731.

[24] Y. Yu and S. Madiraju. "Enterprise Application Transformation Strategy and Roadmap Design: A Business Value Driven and IT Supportability Based Approach". *2014 Enterprise Systems Conference*. 2014, pp. 66–71. DOI: 10.1109/ES.2014.37.

[25] Z. Yongqiang, L. Xintao, and L. Rui. "Theoretical Study on Hybrid Re-Engineering". *2007 8th International Conference on Electronic Measurement and Instruments*. 2007, pp. 1-107-1–110. DOI: 10.1109/ICEMI.2007.4350400.

[26] S. Faily. "Biting the CHERI Bullet: Blockers, Enablers and Security Implications of CHERI in Defence". *2025 International Conference on Military Communication and Information Systems (ICMCIS)*. 2025, pp. 1–9. DOI: 10.1109/ICMCIS64378.2025.11047961.

[27] A. Rostkowycz, V. Rajlich, and A. Marcus. "A Case Study on the Long-Term Effects of Software Redocumentation". *20th IEEE International Conference on Software Maintenance, 2004. Proceedings*. 2004, pp. 92–101. DOI: 10.1109/ICSM.2004.1357794.

[28] J. Zhou, D. Zhao, and J. Liu. "A Lightweight Component-Based Development Approach for Enterprise Applications". *2011 IEEE 35th Annual Computer Software and Applications Conference Workshops*. 2011, pp. 335–340. DOI: 10.1109/COMPSACW.2011.62.

[29] M. G. Stochel, P. Chołda, and M. R. Wawrowski. "Adopting DevOps Paradigm in Technical Debt Prioritization and Mitigation". *2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 2022, pp. 306–313. DOI: 10.1109/SEAA56994.2022.00055.

[30] D. Moise and K. Wong. "An Industrial Experience in Reverse Engineering". *10th Working Conference on Reverse Engineering, 2003. WCRE 2003. Proceedings*. 2003, pp. 275–284. DOI: 10.1109/WCRE.2003.1287258.

[31] J. Anderson and R. Bishop. "An ITSM for a New Era: Leaving a Self-Supported Internal Legacy System for a Brighter Future in the Cloud(s)". *Proceedings of the 2018 ACM SIGUCCS Annual Conference*. Siguccs '18. New York, NY, USA, 2018, pp. 111–113. DOI: 10.1145/3235715.3235739.

[32] A. De Lucia, A. Pannella, E. Pompella, and S. Stefanucci. "Assessing Massive Maintenance Processes: An Empirical Study". *Proceedings IEEE International Conference on Software Maintenance. ICSM 2001.* 2001, pp. 451–458. DOI: 10.1109/ICSM.2001.972758.

[33] H. Sneed. "Estimating the Costs of a Reengineering Project". *12th Working Conference on Reverse Engineering (WCRE'05)*. 2005, pp. 9–119. DOI: 10.1109/WCRE.2005.18.

[34] B. Xu. "Extreme Programming for Distributed Legacy System Reengineering". *29th Annual International Computer Software and Applications Conference (COMPSAC'05)*. Vol. 2. 2005, pp. 160–165 Vol. 1. DOI: 10.1109/COMPSAC.2005.77.

[35] Y. Xiaohu, X. Bin, H. Zhijun, and S. Maddineni. "Extreme Programming in Global Software Development". *Canadian Conference on Electrical and Computer Engineering 2004 (IEEE Cat. No.04ch37513)*. Vol. 4. 2004, pp. 1845–1848 Vol.4. DOI: 10.1109/CCECE.2004.1347565.

[36] J. van Riet, F. Paganelli, and I. Malavolta. "From 6.2 to 0.15 Seconds – an Industrial Case Study on Mobile Web Performance". *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2020, pp. 746–755. DOI: 10.1109/ICSME46990.2020.00084.

[37] B. Xu, X. Yang, Z. He, and A. Ma. "Global Cooperative Design in Legacy System Reengineering Project". *8th International Conference on Computer Supported Cooperative Work in Design*. Vol. 2. 2004, pp. 483–486 Vol.2. DOI: 10.1109/CACWD.2004.1349237.

[38] F. Ahmad, M. Rangappa, N. Katiyar, M. Staniszewski, and D. Varró. "Hybrid Cloudification of Legacy Software for Efficient Simulation of Gas Turbine Designs". *Proceedings of the 45th International Conference on Software Engineering: Software Engineering in Practice*. Icse-Seip '23. Melbourne, Australia, 2023, pp. 384–395. DOI: 10.1109/ICSE-SEIP58684.2023.00041.

[39] A. N. Abdel-Hamid. "Refactoring as a Lifeline: Lessons Learned from Refactoring". *2013 Agile Conference*. 2013, pp. 129–136. DOI: 10.1109/AGILE.2013.18.

[40] T. Kokko, J. Antikainen, and T. Systä. "Adopting SOA – Experiences from Nine Finnish Organizations". *2009 13th European Conference on Software Maintenance and Reengineering*. 2009, pp. 129–138. DOI: 10.1109/CSMR.2009.41.

[41] H. P. Jepsen, J. G. Dall, and D. Beuche. "Minimally Invasive Migration to Software Product Lines". *11th International Software Product Line Conference (SPLC 2007)*. 2007, pp. 203–211. DOI: 10.1109/SPLINE.2007.30.

[42] T. C. Fanelli, S. C. Simons, and S. Banerjee. "A Systematic Framework for Modernizing Legacy Application Systems". *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. Vol. 1. 2016, pp. 678–682. DOI: 10.1109/SANER.2016.40.

[43] J. Guo. "A Systematic Method of Reusing Objects Extracted from Legacy Systems". *Proceedings Ninth Annual IEEE International Conference and Workshop on the Engineering of Computer-Based Systems*. 2002, pp. 177–184. DOI: 10.1109/ECBS.2002.999836.

[44] B. A. Pinos-Figueroa and G. A. León-Paredes. "An Approach of a Migration Process from a Legacy Web Management System with a Monolithic Architecture to a Modern Microservices-Based Architecture of a Tourism Services Company". *2023 11th International Conference in Software Engineering Research and Innovation (CONISOFT)*. 2023, pp. 9–17. DOI: 10.1109/CONISOFT58849.2023.00012.

[45] S. Strobl, M. Bernhart, T. Grechenig, and W. Kleinert. "Digging Deep: Software Reengineering Supported by Database Reverse Engineering of a System with 30+ Years of Legacy". *2009 IEEE International Conference on Software Maintenance*. 2009, pp. 407–410. DOI: 10.1109/ICSM.2009.5306293.

[46] D. Amalfitano, A. R. Fasolino, V. Maggio, P. Tramontana, G. Di Mare, F. Ferrara, et al. "Migrating Legacy Spreadsheets-Based Systems to Web MVC Architecture: An Industrial Case Study". *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*. 2014, pp. 387–390. DOI: 10.1109/CSMR-WCRE.2014.6747201.

[47] A. Alvaro, D. Lucredio, V. Garcia, A. Francisco do Prado, and L. Trevelin. "Orion-RE: A Component-Based Software Reengineering Environment". *10th Working Conference on Reverse Engineering, 2003. WCRE 2003. Proceedings.* 2003, pp. 248–257. DOI: 10.1109/WCRE.2003.1287255.

[48] R. Mishra, N. Jaiswal, R. Prakash, and P. N. Barwal. "Transition from Monolithic to Microservices Architecture: Need and Proposed Pipeline". *2022 International Conference on Futuristic Technologies (INCOFT)*. 2022, pp. 1–6. DOI: 10.1109/INCOFT55651.2022.10094556.

[49] K. R. Luecke, B. J. Ellis, I. Baxter, R. L. Akers, and M. Mehlich. "Software Code Base Conversions". *2007 IEEE/AIAA 26th Digital Avionics Systems Conference*. 2007, pp. 6.C.4-1-6.C.4–11. DOI: 10.1109/DASC.2007.4391960.

[50] N. Anquetil, A. Etien, G. Andreo, and S. Ducasse. "Decomposing God Classes at Siemens". *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2019, pp. 169–180. DOI: 10.1109/ICSME.2019.00027.

Application of Agile Project Management Techniques in Industrial Practice: A Multivocal Literature Review

Christopher Grimnitz
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
christopher.grimnitz@rwth-aachen.de

Haomiao Xu
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
haomiao.xu@rwth-aachen.de

ABSTRACT

Agile project management (APM) plays a central role in today's software industry and can deliver substantial benefits for teams when applied effectively. There is extensive literature on Agile management techniques (AMTs) from both academia and industry, but it rarely focuses exclusively on the software industry or integrates both perspectives. Furthermore, grey literature (GL) is often excluded, and there is limited analysis of recent developments or comparisons between what academia proposes and what organizations use in practice.

We present a multivocal literature review (MLR) that combines 66 scientific studies, including results identified using artificial intelligence (AI), and four GL reports to provide the most comprehensive and up-to-date overview of APM in practice. Our findings show that both scientific and industry sources focus heavily on Scrum. We also found that hybrid approaches are common, suggesting teams often adapt agile methods rather than following them exactly as described.

Our review shows that successful agile adoption relies on strong management support, effective team communication, team autonomy, and continuous adaptation, according to both industry and academic sources. AI and large language models (LLMs) are starting to support agile work by assisting with planning, task assignment, and improving work quality, but people still make the main decisions. Organizations utilizing APM gain benefits like fast product delivery, rapid response to feedback, and better team satisfaction, although they face challenges, including unclear APM guidelines, customer trust issues, poor documentation, and coordination problems in large or distributed projects. And while academic and industry sources mostly agree on the main techniques and success factors, gaps remain in how agile methods are adapted to different situations and in the study o less common methods and approaches. By combining academic and practical perspectives, this review highlights best practices and suggests where more research could help connect theory and practice.

SWC Seminar 2025/26 RWTH Aachen University, Germany.

Categories and Subject Descriptors

D.2 [**Software**]: Software Engineering; D.2.9 [**Software Engineering**]: Management—*agile project management, industry practices, team dynamics*

Keywords

Agile Project Management, Multivocal Literature Review, Industrial Practice

1. INTRODUCTION

Agile methods have become mainstream choices in software project management in recent years, thanks to their fast product delivery, rapid response to feedback, higher team satisfaction, and many other benefits. Nevertheless, the management of the iterations, including planning, monitoring, and controlling, is quite challenging in practice, especially when the context of modern software projects is getting more and more complex. It is needed to establish a robust understanding of AMTs.

Many of the current studies of AMTs focus beyond the software development industry, and most studies have a limited scope, only in scientific literature (SL). However, much relevant information is presented by GL, including industrial reports and white papers. At the same time, there is a lack of reviews focusing solely on recent approaches.

In order to provide a comprehensive understanding of state-of-the-art AMTs, we will conduct an MLR, including SL, GL, and supplemental literature found with the assistance of AI-based search engines. Through this approach, we will not only gain a systemic understanding of state-of-the-art AMTs, but also make a comparison between the perspectives from academia and the industry.

The rest of this study comprises six sections and is structured as follows: Section 2 derives our research questions (RQs) and explains the methodology of this MLR. Section 3 presents our findings from the literature. In Section 4, we will discuss these findings. Section 5 addresses the implications of this study for industry and research. In Section 6, the multivocal and AI-assisted approach is evaluated. Finally, we conclude this study in Section 7.

2. METHODOLOGY

The methodology provides a clear framework for searching, selecting, and analyzing sources. To explore further perspectives, AI-assisted searches are used in addition to the traditional SL and GL searches.

2.1 Research Questions

The review addresses the following RQs, focusing on RQ2-RQ4, which explore how Agile techniques are applied in practice, what conditions support their success, and actual reported experiences.

`RQ1:` Which APM techniques are described in SL?

`RQ2:` Which APM techniques are actually used in industry?

`RQ3:` What are the prerequisites for successfully implementing them?

`RQ4:` What positive and negative experiences are reported?

2.2 Search Strategy

Our sources are academic databases (DBLP, ACM Digital Library, IEEE Xplore), AI-based search engines (ORKG and Consensus), and GL identified via Google Search. We used different search strings for each database. First, we searched the ACM Digital Library and IEEE Xplore using the following Boolean search terms:

```
("agile project management" OR "agile management")
AND ("techniques" OR "methods" OR "practices") AND
("software" OR "software development" OR "software
engineering")
```

To capture more industry-focused research, we also used the following term:

```
("agile project management" OR "agile manage-
ment") AND ("software" OR "software development"
OR "software engineering") AND ("industry" OR "ex-
perience")
```

As DBLP does not support Boolean operators, adapted keyword-based queries were used:

```
agile project|management techniques|methods| prac-
tices software|development|engineering
agile project|management software|development |en-
gineering industry|experience
```

We used the following search terms for the search using ORKG:

```
Find studies that focus on agile project manage-
ment techniques in the software industry. Include
sources that explain how these techniques are ap-
plied in practice, especially in managing agile
projects, as well as scientific discussions on
agile management approaches.
```

We also added the following text to the search of Consensus, as Consensus allows for longer search terms:

```
Exclude papers where agile is only a secondary
topic.
Pay special attention to literature that ad-
dresses: Which agile project management tech-
niques are described in scientific and grey lit-
erature? Which of these techniques are actually
used in industry? What are the prerequisites for
successfully implementing them? What positive and
negative experiences are reported?
```

For the GL search, Google Search was used with queries derived from RQ2-RQ4. RQ1 was excluded, as scientific techniques were already covered by the academic searches.

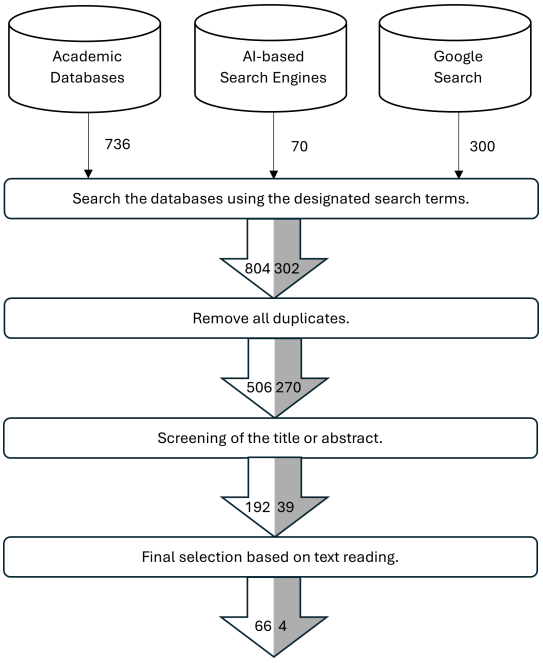


Figure 1: Our search procedure. Numbers on arrows indicate literature moving between steps (white arrow for SL, grey arrow for GL).

We added `filetype:pdf` to focus results on reports rather than basic core technique descriptions (e.g., Scrum), which are not useful for our review.

```
software companies implementing agile report file-
type:pdf
prerequisites for software companies implementing
agile successfully filetype:pdf
experiences of software companies implementing
agile report filetype:pdf
```

2.3 Eligibility Criteria

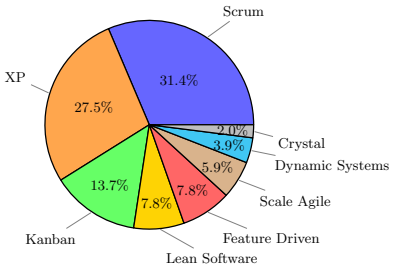
To include only papers that meet our standards and are useful for answering our RQs, we defined clear inclusion criteria in Table 1.

Inclusion Criteria	
IN-1	Literature written in English.
IN-2	SL published between 2015 and 2025 and GL published between 2020 and 2025.
IN-3	Literature should focus on Agile and not only include it as a secondary topic.
IN-4	Literature should focus on the software industry.
IN-5	SL should be published in peer-reviewed journals and workshop conferences, and GL should come from reputable organizations or experts.

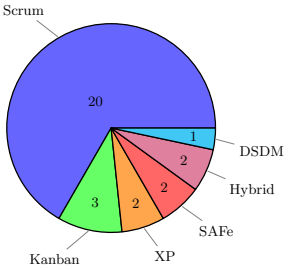
Table 1: Inclusion criteria for literature

2.4 Search Procedure

We began our search as shown in Figure 1, collecting information from databases using the search terms defined in



(a) Trend study of Agile methodologies (2018-2022) redrawn from Trihardiningsih et al. [63].



(b) The distribution of scientific papers in the explanatory literature category we found in our review that discussed each AMT (c.f. Table 2).

Figure 2: A comparison between a study looking at all types of SL on AMTs and our study looking at only explanatory literature.

Section 2.2. We limited results to 50 from ORKG and 100 from each Google search, since both engines yield millions of results, and 20 for Consensus, which has its main results in the first 20. These numbers are sufficient mainly because, as more results are considered, they become repetitive and lose relevance sharply. Next, we removed duplicate literature. We then screened the remaining literature by title and abstract, addressing all questions for SL and focusing on RQ2-RQ4 for GL. Finally, we reviewed the full papers and decide whether to include them.

3. FINDINGS

This section presents the results of our MLR, without analyzing them, and focuses on answering the RQs.

3.1 Overview of Identified Agile Techniques in SL

As a first step, we will address RQ1 and review which AMTs are discussed in the SL.

3.1.1 Core Agile Management Techniques

In the literature, it is important to differentiate between studies that propose or adapt AMTs and studies that analyze the use of existing techniques in practice. The former type, which we will refer to as the explanatory category, includes papers that develop new methods or modify existing approaches, often validating them through case studies or experimental setups. The latter category focuses on observing, reporting, or evaluating how techniques are actually applied in industry, without proposing new methods.

Most reviews, like Trihardianingsih et al. [63], did not distinguish between these categories and report Scrum as the most frequently discussed technique, followed by Extreme Programming (XP), Kanban, and Lean Software Development (see Figure 2a). Other literature with a narrower focus, such as Leite et al. [33] on Agile and technical debt, supports these findings.

Looking at the explanatory category separately matters because it makes it easier to compare what researchers study with what companies actually use, as it may help show where academic insights are not fully applied in industry. For this category, we collected 25 scientific papers discussing the explanatory category. As shown in Figure 2b, Scrum is by far the most popular, with Kanban and XP following far behind. Therefore, our results generally align with those of the other reviews, and some trends are even more pronounced. The dominance of Scrum is clear in both categories and will be discussed later in Section 3.2 as well.

3.1.2 Hybrid Approaches

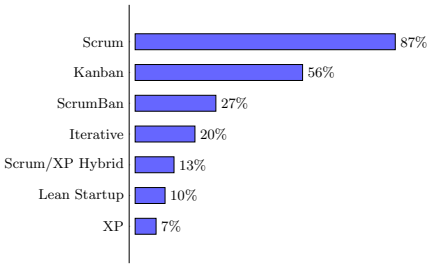
In addition to the established core Agile techniques, several papers described hybrid project management approaches that combine Agile practices with elements from other, sometimes traditional, management methods. A recurring theme across the literature is the need to balance agility with overarching structure. For example, Adelakun et al. [1] described hybrid project management as an attempt to preserve Agile flexibility while introducing additional structure through more traditional management techniques, such as a modified waterfall model in this case. The authors emphasized that no single management approach fits all contexts, whether Agile or traditional.

One example of a hybrid approach is OPENTCQ by Janjua et al. [28], which integrates Agile methods with change management and traditional project management elements. It used the Unified Process (UP) to combine Agile iterative development with the traditional, more structured iteration planning. OPENTCQ also considers practical challenges, such as team size, time constraints, and organizational context, and adapts accordingly. This is supposed to help teams stay flexible while keeping some level of control and coordination in changing environments.

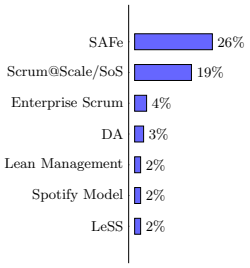
Other hybrid approaches, such as AgileDBR by Smit-Nunes [59], did not combine Agile with traditional management approaches but instead combine with other methods. AgileDBR blends Agile practices with academic models, namely the research methodology design-based research (DBR). But the reasoning for keeping the flexibility of Agile while combining it with traditional, context-specific approaches, and its advantages, remained the same.

3.1.3 Artificial Intelligence and Large Language Models in APM

Recent research has also explored how AI and LLMs could support APM in software development. One area of study is the use of AI as an assistive tool for project and task management. Shahriary et al. [56] compared LLMs acting as virtual scrum masters with traditional human scrum masters and found that AI was good at operational tasks such as ensuring feasibility, clarifying tasks, and organizing sprints. However, humans still outperformed AI in areas requiring hu-



(a) Main AMTs leveraged by the survey respondents from [14], redrawn from [14]



(b) Main scaled Agile frameworks leveraged by the survey respondents from [15], redrawn from [15]

Figure 3: Main AMTs and scaled Agile Frameworks

man judgment, including dynamic project adaptation, risk identification, and team management. Similarly, Dhruva et al. [13] proposed that LLMs could assist in automating task allocation using enterprise data.

AI is also being explored for resource allocation and risk management. Almalki [5] and Patel et al. [42] proposed AI-based systems that, among others, help assign tasks to team members more effectively, identify potential imbalances in workload, and recognize risks early. Almalki's [5] model even outperformed traditional Agile applications in some scenarios, which shows the potential for AI to improve established Agile methodologies.

Another area is enhancing the quality of Agile artifacts, such as user stories and epics. Geyer et al. [21] and Schwedt et al. [54] looked at AI-supported evaluation of these artifacts and showed that AI can check for consistency, identify missing or unclear requirements, and generally support analysis. They also discuss how such tools can be used not only to assess but also to actively improve user stories and epics.

Despite these promising results, several challenges remain according to Saxena et al. [53] and Hamza et al. [24]. AI tools depend on reliable data, and not only do they need to be adapted to the specific project context, but the humans using them need to be trained on them as well. There are also ethical concerns about trust and accountability when AI makes decisions that were traditionally made by human team members.

3.2 Techniques Used in Industry

This section examines SL on case studies and GL to find out which techniques are actually used in real-world organizations to answer RQ2.

3.2.1 *Most Commonly Adopted Techniques*

The US GAO Agile Assessment Guide report (AAGr) [65] and the 16th State of Agile report (SoAr) [14] pointed out that the AMTs commonly adopted in the software industry included Scrum, Kanban, ScrumBan, Iterative, Lean Startup, eXtreme Programming, Scrum/XP Hybrid, and Feature Driven Development (FDD). As indicated in Figure 3a, Scrum keeps its popularity, as it prevails the SL stated in Section 3.1.1, followed by Kanban, leveraged by more than half of the survey respondents from [14]. ScrumBan and Scrum/XP Hybrid are examples of hybrid approaches introduced in Section 3.1.2 applied by companies in the real world.

3.2.2 *Supporting Practices in Industry*

In practice, digital project management tools are developed and utilized to maximize the usability of the core AMTs mentioned before. According to Nandula Kowshik et al. [61] and the 17th SoAr [15], major APM tools included Jira, Trello, Asana, Azure DevOps, Miro, and Microsoft Project. These tools provide features for user stories and backlog management, sprint planning, virtual boards, burndown charts, and collaboration tools across the Software Development Life Cycle (SDLC) managed by Agile techniques, contributing to enhancing collaboration, improving workflow efficiency, and ensuring project success [61].

AI is also explored by practitioners. The 17th SoAr [15] pointed out that about 80% of the survey respondents were trying to integrate AI with their Agile projects. Hamza et al. [24] conducted a survey on using AI in APM, revealing substantial improvement in the accuracy of estimation, resource utilization, and speed of decision-making. Remah et al. [69] observed that Natural Language Process (NLP) methods were applied in Agile projects dealing with the effort estimation process, based on the user stories, although challenged by the lack of datasets for NLP networks creation and training.

Mourad et al. [8] investigated the adoption of Design Thinking (DT) in Agile projects by User Experience (UX) designers, and the results indicated that the practice of user testing in the DT framework and user research autonomously conducted by UX designers contribute to the enhancement of the pace and efficiency of APM.

3.2.3 *Context-Specific Findings*

Core AMTs like Scrum were originally invented for collaboration in a single team. In order to manage large-scale projects carries out by multiple teams at the same time, scaled frameworks such as Disciplined Agile (DA), Dynamic Systems Development Method (DSDM), Large Scale Scrum (LeSS), Scaled Agile Framework (SAFe), Scrum@SScale, Scrum of Scrums (SoS), Lean Management, Spotify Model, Enterprise Scrum, and other frameworks are proposed and commonly used in the industry, as stated by the AAGr [65] and the 17th SoAr [15]. As illustrated in Figure 3b, SAFe is the most prevalent scaled framework, followed by Scrum@Scale and SoS.

In the successful practices of large-scale Agile projects presented by Knut H. et al. [48] and Torgeir et al. [16], extra roles were created, additional common rules were implemented, closer collaboration between customer and supplier was performed, tasks were re-distributed during sprints, mini

demos were showcased during the sprints, and more arenas were added to strengthen inter-team coordination, which provides meaningful lessons for later practitioners on large-scale APM.

3.3 Prerequisites for Successful Implementation

The literature shows that successfully applying APM techniques in industry depends less on the chosen method and more on the conditions in which it is used. Both SL and GL point to a set of organizational, team-, and process-related, as well as context-specific factors that strongly influence whether Agile techniques work in industry.

3.3.1 Organizational Prerequisites

A common finding across many studies is that Agile needs support at the organizational level. Several reviews and empirical studies report that (top) management support, an organizational culture that is open, and an effective process of aligning with Agile values are essential for Agile initiatives to succeed, according to Junior et al. [30] and Fortuna et al. [20]. These critical success factors are essential, as when they are missing, Agile is often applied halfheartedly and conflicts with existing structures, rather than improving them.

Agile tends to work better in environments that allow autonomy, trust in teams, and encourage continuous improvement, while rigid hierarchies often limit its effectiveness according to the AAGr [65]. Industry reports, like the 17th SoAr [15], also support the view that visible support from leadership is a key factor in the effectiveness of implementation, as many reasons for failure are in some way connected to this.

3.3.2 Team-Level Prerequisites

Other commonly named prerequisites for effective Agile implementation are at the team level. Fortuna et al. [20], and Junior et al. [30] emphasize effective communication, team self-organization, and flexibility. Fortuna et al. [20] highlight team empowerment in particular. Fortuna et al. [20] and Palopak et al. [40] additionally both highlight customer focus as important, with Palopak et al. [40] even stating it as the most important factor. Ahsan et al. [2] also point out that the quality of the project manager is of importance, with good teamwork, effective communication, Agile delivery, and implementation experience, as well as stakeholder management being named as the most important qualities.

Industry-focused sources, such as PwC [44], the 17th SoAr [15], and the AAGr [65], report similar observations. They also note that teams often struggle when Agile is introduced without clear guidance or training, emphasizing the importance of continuous learning and adaptation.

3.3.3 Process Prerequisites

From a process perspective, Dingsøyr et al. [17] and Edison et al. [18] stress that Agile practices should be adapted to the project context rather than applied rigidly. Reviews show that tailoring practices to factors such as project size, complexity, or regulatory requirements is especially important in larger environments. Supporting processes such as estimation, monitoring, backlog management, and risk management also need to align with Agile ways of working to avoid unnecessary overhead [7, 17, 62]. The process should also allow for frequent communication, for example, in daily meetings, according to Junior et al. [30].

GL, such as the PwC report [44], also notes that Agile practices are often implemented alongside existing organizational and process structures rather than replacing them outright. In these situations, a gradual adoption that is continuously monitored and developed further is often seen as more manageable than a complete Agile transformation. Hybrid approaches are also mentioned as a possible solution.

3.3.4 Context-Specific Prerequisites

APM techniques are often based on small, dynamic teams. When scaling up, large-scale Agile projects often face significant challenges related to coordination and sustaining Agile practices over time. Research indicates that these environments require tailored practices and strong, motivated management involvement [4, 17, 55]. Because of their size and complexity, large organizations often find it harder to adopt Agile. Teams need to coordinate across multiple departments and projects while still following existing processes and reporting rules.

Industry and public-sector reports, such as the PWC report [44], also note differences in Agile implementation based on company or project size. They highlight that large organizations and government agencies often report leadership issues as problems for implementing Agile. Employees of smaller companies, on the other hand, report having no barriers much more often, as noted in the 17th SoAr [15]. The stricter regulations and policies in the public sector can also make a full transition to Agile particularly challenging, which makes broad support from leaders across the organization especially important, according to the AAGrv [65].

3.4 Reported Experiences

This section collects the benefits and challenges of Agile techniques in practices reported in the reviewed sources to address RQ4.

3.4.1 Reported Benefits

According to Faisal et al. [25], Asmaa et al. [6], Luca et al. [34], and the 17th SoAr [15], the most frequently mentioned benefits of applying Agile techniques could be categorized as: i) Organizational level benefits: better work environment; ii) Team-Level benefits: increased collaboration, increased time for innovation, better team satisfaction, and better knowledge-sharing; iii) Process-Level benefits: better alignment to business needs, better quality software delivered, increased visibility across the SDLC, faster response to competitive threats, better user experience, better customer service, improved risk management, decrease of User Acceptance Testing (UAT) duration, aligned project vision, and faster responsiveness to feedback. The survey about job satisfaction carried out by Martin et al. [32] pointed out that 40% of the plan-driven developers were unsatisfied, while a very high satisfaction rate (85%) was observed for those in Agile projects. Magne [29] found that Agile projects had a higher average success and acceptance rate compared with non-Agile projects for small-, medium-, and large-sized projects.

Benefits of the Agile management tools, such as Jira, are reported by Florian et al. [45], including easy access to additional documents, automated archiving of relevant project

data, quick filtering and searching of requirements, automated generation and visualization of relevant key-figures, and remote access and screen-sharing for distributed teams.

3.4.2 Reported Challenges

However, Agile is never easy to achieve in practice. Obstacles exist both for the adoption and implementation of AMTs.

The 17th SoAr [15], the AAGr [65], Asmaa et al. [6] stated that the barriers for Agile adoption in organizations consisted of: organizational resistance to change, not enough leadership participation, inadequate management support, business team not understanding Agile, insufficient employee training, lack of business owner availability, transformation is led by the technology organization, budget and/or revenue issues, difficulty in timely adoption of new tools, difficulty in establish and maintain technical environments, unclear guidance for Agile, procurement practices may not have supported Agile projects.

The challenges for the implementation of AMTs, as mentioned by the 17th SoAr [15], the AAGr [65], Geetha et al. [51], Luca et al. [34], and Nakash [38], were: i) Organizational level challenges: existing legacy systems requiring mixed approaches, siloed teams causing delays on deliverables, some Agile methods clash with company culture, inconsistent implementation of Agile across teams, changing leadership dynamics, trouble in committing staff; ii) Team-Level challenges: difficulty in collaboration (especially for remote collaboration), difficulty in transitioning to self-directed work, requirement management issues, lack of sufficient Agile experience; iii) Process-Level challenges: low adoption, broken process, insufficient customer feedback, hard to predict and estimate time of deliverables, lack of visibility on dependencies, customers did not trust Agile solutions, hard to execute compliance reviews within an iteration time frame, inconsistency in stakeholder buy-in, too many meetings being distracting; iv) Individual-Level challenges: difficulty in committing to more timely and frequent input.

There are also drawbacks to the Agile management tools. Florian et al. [45] pointed out that the lack of guidelines on combining the digitalized management tools with physical media, such as paper-based boards, which promote team collaboration, reduced the efficiency of Agile and led to an extra extensive learning phase for teams on that.

3.4.3 Context-Driven Differences

Incorporating Agile in large-scale projects leads to more challenges. Through case studies, Hina et al. [52] observed challenges in large-scale Agile projects in addition to those mentioned in Section 3.4.2, including: i) Organizational level challenges: organizational fragmentation and lack of common structure, and delayed decision-making; ii) Team-Level challenges: continuously changing teams, distributed teams, interdependent teams, delayed tasks and team conflicts, lack of knowledge management and sharing, communication and collaboration across teams, and challenges in workload and expertise management across teams; iii) Process-Level challenges: lack of testing, unprioritized product backlog, poor documentation, blind sprints, pending cases in the product backlogs, compromised quality assurance, and deadlines challenges. Brian et al. [27] noted foundational conflicts between Agile principles and techniques and large bureaucratic organizations with well-established plan-driven methods, such as the organization of a great number of specialists outside the development teams and differences in the role definitions in traditional organizations and Agile principles. Meanwhile, the project's relationship with the client organization may suffer as the transition from traditional to Agile methods is radical, and the client does not necessarily accept Agile methodologies [27].

With the economy growing more and more globalized, many software projects are carried out with distributed teams, which entails other challenges. Ingo et al. [47] pointed out that Agile Global Software Engineering (AGSE) was troubled by fragmented documentation in Agile, exacerbated by the distribution of teams.

Agile techniques are also applied in the Research and Development in the field of Internet of Things (IoT). Moedt et al. [36] noticed that the frequent iteration of IoT techniques, platforms, and architectural principles requires flexibility in terms of the end product, and it was recommended to leave some time in a sprint for these unplanned items.

Agile techniques could be utilized by the public sector in order to increase performance as well as cut costs. However, Anna [31] and Dipendra et al. [22] noted challenges such as: i) The detailed description of the solution required by the public projects at the outset making it difficult to introduce subsequent iterative changes; ii) The customer of public sector have no choice and may not go to another provider when the product does not meet their expectation, leading to a high market pressure, making the incremental improvement relatively trivial; iii) Potential restrictions of knowledge sharing between different levels of management in public organizations and lack of transparency might hamper the implementation of Agile techniques.

4. DISCUSSION

The findings of this review reveal clear patterns in how APM is studied and applied in practice. Looking at both scientific and industrial sources and analyzing them helps us put these findings into context, understand similarities and differences between the two perspectives, and identify trends.

4.1 Comparison of SL and GL Coverage of AMTs

Our findings show that scientific and industrial literature often support each other, but they take different approaches. Scientific papers usually dive deep into specific Agile techniques, explaining, testing, or analyzing them in detail, often through case studies or surveys in a small number of organizations. In contrast, GL comes from large industry surveys and reports, such as the State of Agile, PwC, or the AAGr, which provide a much broader view of the practices actually used and the challenges faced in real-world settings.

While perspectives agree on roughly the same most used AMTs, GL surveys a wide range of organizations to see what actually works, while SL dives into the details, often developing new methods and testing them in case studies. So, scientific studies tend to propose and test new ideas, whereas GL analyzes what is happening in the field and what proves effective.

Both SL and GL consistently highlight management support, effective team communication, team autonomy, and continuous adaptation as key prerequisites to success. SL of-

ten attempts to define best practices for these prerequisites, while GL typically derives them from respondent data.

A notable difference also emerges in the coverage of emerging topics such as AI and LLMs. SL explores these technologies in detail, whereas in the GL, they appear mainly as secondary topics within broader industry reports.

Overall, the comparison shows that the scientific and industrial literature largely agree on which Agile techniques are used, which prerequisites matter most, and the general state of Agile. The main difference lies in how they gather information and reach conclusions, with each perspective complementing the other to provide a more complete understanding of Agile in practice.

4.2 Interpretation of Technique Adoption

The results show a clear pattern across both scientific and industry sources that Scrum is by far the most commonly discussed and used Agile technique, followed by Kanban, XP, and a few other recurring approaches. This strong focus on Scrum is evident throughout our review, including the general SL, explanatory studies, and industry reports. Based on our findings, especially in Figure 2 and Sections 3.1 and 3.2, Scrum is even more dominant in explanatory SL.

The reasons for this dominance are not entirely clear, but our results suggest a reinforcing effect between research and practice. Because Scrum is widely used in industry, it is often chosen as a starting point for research and discussion. At the same time, its strong presence in academic work likely contributes to its continued adoption in practice. For many people encountering APM for the first time, Scrum is likely to be the entry point, making it more likely to be used in projects and studied by researchers.

Hybrid approaches are less common than Scrum, but they remain important in both research and industry sources. Industry reports describe hybrid approaches as practical ways to meet specific needs, while scientific studies often explore how Agile methods are adapted or blended rather than used in their original form. In practice, teams rarely stick to pure AMTs, instead tailoring them to their specific context. SL proposes and tests these modifications, while GL confirms their widespread use in practice.

Overall, the patterns we observed show that a small set of well-known techniques dominate both research and practice, but adaptations and combinations are common, especially in practice. This indicates that APM is not applied in the same way everywhere, and teams adjust it to fit their own needs and situations.

4.3 Summary of Key Trends

We observed several trends from the reviewed literature.

1. **Scrum is prevailing:** In academia, 20 out of 25 scientific papers were collected in the explanatory category. In the industry, Scrum is also the most leveraged AMT, as indicated by Figure 3a. AMTs like Kanban and ScrumBan are also leveraged by a large portion of professionals.

2. **Hybrid approaches emerge:** These practices, either combining practices from different Agile methodologies or aligning traditional management techniques with Agile techniques, indicate that the 'perfect' model for any case does not exist.

3. **AI:** The integration of AI, including LLMs and NLP, into Agile management is widely explored by researchers and practitioners. AI techniques are able to assist in improving the performance of Agile management. Although there are challenges and concerns about its usage, the consensus agrees on the potential of AI in the field of Agile.

4. **Large-scale:** The adoption and implementation of AMTs in large-scale projects are examined by practitioners. SAFe is the most widely applied framework to scale Agile methods to coordinate different teams. Tailored practices beyond the scaled Agile framework principles are emphasized in the literature for the success of large-scale Agile projects.

5. **Prerequisites of AMTs:** The circumstances of the in-plan Agile projects play a significant role in the success of the implementation of AMTs. The literature indicates a set of organizational, team-, and process-level factors, together with context-specific factors, as prerequisites of Agile projects.

6. **Experience with AMTs:** AMTs are beneficial, as well as challenging. Agile techniques are never easy to implement; there are barriers lying on the way to success. It reveals that Agile is not just the business of the organizations, but relates to every participating individual.

5. IMPLICATIONS

The findings from our review suggest practical takeaways for industry and highlight a few areas that could be further explored in scientific work.

5.1 Implications for Industry

This literature review provides Agile practitioners with the understanding of state-of-the-art AMTs, as well as the prerequisites for the successful implementation of AMTs, the potential benefits from these practices, and possible obstacles on organizational, team-, and process-level, altogether with context-specific practices.

For practitioners, the prerequisites reported regarding the AMTs are useful in deciding the proper method to use in their projects. Practitioners should know that Agile is not a panacea and never forget that Agile is not simple, and it requires sufficient knowledge of the organizational culture, the team dynamics, and the expertise of each stakeholder, in combination with good communication and collaboration, and careful distribution of workload to overcome the challenges and achieve success. Practitioners should also consider the client's acceptance of Agile methodologies, and if there are legacy systems needed to be coped with. According to the context of the projects to be carried out, practitioners should autonomously adopt tailored practices, such as creating extra roles for large-scale projects. Agile methods are flexible and should never be rigid.

5.2 Implications for Research

The findings of this review suggest several directions for future scientific work. Many of the prerequisites for successful Agile adoption, such as management support, communication, and adaptation, are covered extensively in the

literature. As a result, further research that merely replicates these findings may add little new value.

Future studies could also benefit from a closer examination of the explanatory category in the SL on APM techniques. In this review, we only considered a limited number of papers in this category in Figure 2b. A larger, dedicated review of this type of research could help identify which techniques and assumptions dominate this part of research and how they compare to what is actually used in practice. Such an analysis could also shed more light on where research could better support industry needs, or where existing research insights are not yet reflected in practice. The strong focus on Scrum, even stronger than in general scientific research, as seen in Figure 2, suggests that other Agile techniques and approaches remain underexplored and warrant greater attention. This also applies to general scientific research on AMTs, but on a smaller scale.

Finally, we found only a small amount of literature specifically examining small- and medium-sized organizations and projects. This may be because Agile often works with fewer issues in these settings, but that does not mean there is nothing to learn from them. More generally, many studies point out that Agile depends heavily on context, but in many contexts, there is insufficient information on how practices should change in different situations. Looking more closely at these kinds of context-specific cases could help build a better understanding of how Agile works in practice.

6. EVALUATION OF MULTIVOCAL AND AI-ASSISTED APPROACH

Combining SL with GL helped us better compare academic and industrial perspectives on APM. While these sources often approached problems from different angles, they generally reached similar conclusions and frequently complemented each other by highlighting aspects that the other alone might have missed.

A large portion of the GL we initially identified proved to be of limited value, as reflected in the fact that only 4 of 302 sources were included. However, the relevant reports still provided useful insights, mainly by adding industry-wide data and perspectives that were less common in scientific studies. Even though much of the GL was eliminated early, the included sources helped support and contextualize findings about common practices and trends in the industry.

Our GL search focused mainly on reports, leading to the inclusion of mostly report-style sources rather than detailed descriptions of individual organizations. But using broader search strings often led to basic explanations of core Agile techniques, which were not useful for this review. Overall, focusing on reports turned out to be a good choice, as they still provided relevant insights for our analysis.

The AI-assisted searches also contributed useful material. For example, all papers identified through the Consensus search were included after abstract screening. While some of these papers were of poor quality, including ones generated by ChatGPT, the approach still helped uncover relevant scientific work. In contrast, AI-assisted searches were less effective for identifying useful GL.

Overall, even though not all the sources we found were useful, using both traditional and AI-assisted searches helped us gather input from both academic and industry perspectives, thereby improving the quality of our review.

7. CONCLUSION

This study aims to develop a comprehensive understanding of state-of-the-art AMTs, as well as the prerequisites for applying these techniques and the positive and negative experiences in practice. We specified this goal by defining four RQs (RQ1-RQ4). We conducted an MLR, covering both SL, GL, and literature found by AI-based search engines. From the literature we collected, these questions all got corresponding answers in Section 3. Therefore, this study contributes a unified source for practitioners and researchers for a holistic understanding of AMTs.

The collected literature provides a comprehensive overview of APM. Scrum, Kanban, XP, and hybrid methods are the most common in both research and practice, and supporting techniques using AI are emerging. Industrial reports largely align with scientific studies, showing that Scrum and Kanban are widely used alongside scaled frameworks such as SAFe. Successful implementation depends on strong management support, effective team communication, team autonomy, and continuous adaptation. Agile methods provide benefits such as fast product delivery, rapid response to feedback, and higher team satisfaction. Still, they also face challenges such as resistance to change, unclear APM guidelines, customer trust issues, poor documentation, and coordination problems in large or distributed projects.

The industrial reports included in GL provided very useful information on the state of Agile management and detailed descriptions of the prerequisites and challenges faced by the practitioners. Meanwhile, studies found by AI-based search engines, especially Consensus, offered supplemental understanding in the field of Agile management. Thus, by conducting this MLR, this study serves as evidence that, in addition to the traditional scientific databases, GL and AI-based search engines could be effective in presenting relevant information and eventually help to form a comprehensive understanding of the desired research topic, as well as the difference between academic and industrial perspectives.

Content Item	Associated References
Section 3.1.1	[33, 63]
Figure 2a	[63]
Figure 2b	[3,9,10,11,12,19,23,26,35,37,39,41,43, 46, 49, 50, 53, 57, 58, 60, 64, 66, 67, 68, 70]
Section 3.1.2	[1, 28, 59]
Section 3.1.3	[5, 13, 21, 24, 42, 53, 54, 56]
Section 3.2.1	[14, 65]
Figure 3a	[14]
Figure 3b	[15]
Section 3.2.2	[8, 15, 24, 61, 69]
Section 3.2.3	[15, 16, 48, 65]
Section 3.3.1	[15, 20, 30, 65]
Section 3.3.2	[2, 15, 20, 30, 40, 44, 65]
Section 3.3.3	[7, 17, 18, 30, 44, 62]
Section 3.3.4	[4, 15, 17, 44, 55, 65]
Section 3.4.1	[6, 15, 25, 29, 32, 34, 45]
Section 3.4.2	[6, 15, 16, 34, 38, 45, 51, 65]
Section 3.4.3	[22, 27, 31, 36, 47, 52]

Table 2: Mapping of sections, figures, or tables to their references to provide a better overview of the used sources.

8. REFERENCES

[1] O. Adelakun, R. Garcia, T. Tabaka, and R. Ismail. Hybrid project management: Agile with discipline. In *CONF-IRM 2017 Proceedings*, page 14, 2017.

[2] K. Ahsan and M. Ho. Analysis of agile project manager competencies from recruitment signals. *IEEE Transactions on Engineering Management*, 71:3892–3905, 2024.

[3] A. Alhazmi and S. Huang. Integrating design thinking into scrum framework in the context of requirements engineering management. In *Proceedings of the 3rd International Conference on Computer Science and Software Engineering*, CSSE '20, page 33–45, New York, NY, USA, 2020. Association for Computing Machinery.

[4] F. Ali, M. Usman, M. F. Abrar, S. U. Rahman, I. Khan, and B. Niazi. Practices of de-motivators in adopting agile software development methods at large scale development teams from management perspective. *IEEE Access*, 11:130368–130390, 2023.

[5] S. S. Almalki. Ai-driven decision support systems in agile software project management: Enhancing risk mitigation and resource allocation. *Systems*, 13(3), 2025.

[6] A. Anwar, A. A. Kamel, and E. Ahmed. Agile adoption case study, pains, challenges & benefits. In *Proceedings of the 2nd Africa and Middle East Conference on Software Engineering*, AMECSE '16, page 60–65, New York, NY, USA, 2016. Association for Computing Machinery.

[7] E. D. Canedo, D. P. Aranha, M. d. O. Cardoso, R. P. d. Costa, and L. L. Leite. Methods for estimating agile software projects: A systematic review. In *Proceedings of the 30th International Conference on Software Engineering and Knowledge Engineering (SEKE 2018)*, pages 34–39. KSI Research Inc. and Knowledge Systems Institute Graduate School, 2018.

[8] M. Chouki, M. Hofaidhllaoui, M. Kefi Ben Chehida, L. Giraud, and M. Dabić. "design-led agility: Unraveling the influence of design thinking in the agile methodology journey by user experience designers". *IEEE Transactions on Engineering Management*, 71:15478–15494, 2024.

[9] L. M. H. Cruz and B. B. Díaz. Methodology agilepm-s: A proposal for the agile management of academic software projects. In *2022 17th Iberian Conference on Information Systems and Technologies (CISTI)*, pages 1–6, 2022.

[10] N. Damij and T. Damij. An approach to optimizing kanban board workflow and shortening the project management plan. *IEEE Transactions on Engineering Management*, 71:13266–13273, 2024.

[11] V. J. A. T. de Melo França, R. Balancieri, G. C. L. Leal, and A. C. Rouiller. Mixed integer programming helping requirements allocation for the nrp in scrum teams. In *Proceedings of the XVII Brazilian Symposium on Software Quality*, SBQS '18, page 279–286, New York, NY, USA, 2018. Association for Computing Machinery.

[12] H. T. de Sousa, T. S. Silva, P. S. Santos, R. F. Calhau, and M. B. Costa. Automated support for scrum projects sprint planning. In *Proceedings of the Annual Conference on Brazilian Symposium on Information Systems: Information Systems: A Computer Socio-Technical Perspective - Volume 1*, SBSI '15, page 291–298, Porto Alegre, BRA, 2015. Brazilian Computer Society.

[13] G. Dhruva, I. Shettigar, S. Parthasarthy, and V. M. Sapna. Agile project management using large language models. In *2024 5th International Conference on Innovative Trends in Information Technology (ICITIIT)*, pages 1–6, 2024.

[14] Digital.ai. 16th annual state of agile report. Technical report, Digital.ai, 2022.

[15] Digital.ai. 17th annual state of agile report. Technical report, Digital.ai, 2024.

[16] T. Dingsoeyr, D. Falessi, and K. Power. Agile development at scale: The next frontier. *IEEE Software*, 36(2):30–38, 2019.

[17] T. Dingsøyr, T. Dybå, M. Gjertsen, A. O. Jacobsen, T.-E. Mathisen, J. O. Nordfjord, K. Røe, and K. Strand. Key lessons from tailoring agile methods for large-scale software development. *IT Professional*, 21(1):34–41, 2019.

[18] H. Edison, X. Wang, and K. Conboy. Comparing methods for large-scale agile software development: A systematic literature review. *IEEE Transactions on Software Engineering*, 48(8):2709–2731, 2022.

[19] H. Elmunsyah, W. N. Hidayat, U. Pujianto, H. Suswanto, and M. D. Y. Atsal. Adapting the scrum framework for agile project management: Case study of a sisinta thesis management platform. In *2024 Seventh International Conference on Vocational Education and Electrical Engineering (ICVEE)*, pages 222–227, 2024.

[20] A. Fortuna, C. S. Mattos, A. J. D. C. Andrade, L. F. Ramos, E. Dutra, R. P. D. Santos, and G. Santos. Surveying the relevance of the critical success factors of agile transformation initiatives from a project management perspective. In *Proceedings of the XXII Brazilian Symposium on Software Quality*, SBQS '23, page 110–119, New York, NY, USA, 2023. Association for Computing Machinery.

[21] W. Geyer, J. He, D. Sarkar, M. Brachman, C. Hammond, J. Heins, Z. Ashktorab, C. Rosemberg, and C. Hill. A case study investigating the role of generative ai in quality evaluations of epics in agile software development. In *Proceedings of the 4th Annual Symposium on Human-Computer Interaction for Work*, CHIWORK '25, New York, NY, USA, 2025. Association for Computing Machinery.

[22] D. Ghimire, S. Charters, and S. Gibbs. Scaling agile software development approach in government organization in new zealand. In *Proceedings of the 3rd International Conference on Software Engineering and Information Management*, ICSIM '20, page 100–104, New York, NY, USA, 2020. Association for Computing Machinery.

[23] A. Guerrero, R. Fresno, A. Ju, A. Fox, P. Fernandez, C. Muller, and A. Ruiz-Cortés. Eagle: A team practices audit framework for agile software development. In *Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of*

Software Engineering (ESEC/FSE), pages 1139–1143, New York, NY, USA, 2019. Association for Computing Machinery.

[24] M. F. Hamza, R. Bassam AbuRass, M. B. AbuRass, and T. B. Jber. Agile project management optimization with ai technology: Challenges and impact. In *2025 12th International Conference on Information Technology (ICIT)*, pages 393–397, 2025.

[25] F. Hayat, A. U. Rehman, K. S. Arif, K. Wahab, and M. Abbas. The influence of agile methodology (scrum) on software project management. In *2019 20th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pages 145–149, 2019.

[26] L. Herve Nankap, B. Bouchard, G. Imbeau, and Y. Francillette. Enhancing agile project management for remote teams: A graph theory-based approach. In *Proceedings of the 2025 8th International Conference on Software Engineering and Information Management*, ICSIM '25, page 17–23, New York, NY, USA, 2025. Association for Computing Machinery.

[27] B. Hobbs and Y. Petit. Agile methods on large projects in large organizations. *Project Management Journal*, 48(3):3–19, 2017.

[28] J. I. Janjua, A. Ali, and M. U. Chaudhry. Opentcq: Towards change management in hybrid agile model. In *Proceedings of the 7th International Conference on Computing Communication and Networking Technologies*, ICCCNT '16, New York, NY, USA, 2016. Association for Computing Machinery.

[29] M. Jorgensen. Relationships between project size, agile practices, and successful software development: Results and analysis. *IEEE Software*, 36(2):39–43, 2019.

[30] A. C. P. Junior, V. R. da Silva, and P. T. A. Junior. Critical success factors for agile software development. *IEEE Transactions on Engineering Management*, 71:14807–14823, 2024.

[31] A. Kaczorowska. Traditional and agile project management in public sector and ict. In *2015 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 1521–1531, 2015.

[32] M. Kropp, A. Meier, C. Anslow, and R. Biddle. Satisfaction, practices, and influences in agile software development. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*, EASE '18, page 112–121, New York, NY, USA, 2018. Association for Computing Machinery.

[33] G. d. S. Leite, R. E. P. Vieira, L. Cerqueira, R. S. P. Maciel, S. Freire, and M. Mendonça. Technical debt management in agile software development: A systematic mapping study. In *Proceedings of the XXIII Brazilian Symposium on Software Quality*, SBQS '24, page 309–320, New York, NY, USA, 2024. Association for Computing Machinery.

[34] L. Mainetti and L. Manco. On the effects of introducing agile methodologies in software industry. In *Proceedings of the International Conference on Geoinformatics and Data Analysis*, ICGDA '18, page 31–40, New York, NY, USA, 2018. Association for Computing Machinery.

[35] P. Manaswini, S. Shaik, S. Ashwani, R. M. Balakrishnan, and S. M. Rajagopal. Nova: Revolutionizing agile project management with scrum and kanban integration. In *2024 5th IEEE Global Conference for Advancement in Technology (GCAT)*, pages 1–7, 2024.

[36] W. Moedt van Bolhuis, R. Bernsteiner, M. Hall, and A. Fruhling. Enhancing iot project success through agile best practices. *ACM Trans. Internet Things*, 4(1), Feb. 2023.

[37] D. Montalvão Junior, T. Batista, and E. Cavalcante. An agile management model for distributed software development teams. In *Proceedings of the XXXVII Brazilian Symposium on Software Engineering*, SBES '23, page 122–131, New York, NY, USA, 2023. Association for Computing Machinery.

[38] M. Nakash. Agile software development: The experience of working in sprints [research in progress]. *InSITE Conference*, 2024.

[39] G. R. Neri. The use of exploratory software testing in scrum. *SIGSOFT Softw. Eng. Notes*, 48(1):59–62, Jan. 2023.

[40] Y. Palopak and S.-J. Huang. Correlation of agile principles and practices to software project performance: An ahp–delphi analysis. *International Journal of Software Engineering and Knowledge Engineering*, 32(2):1–25, 03 2022.

[41] J. S. Park, P. E. McMahon, and B. Myburgh. Scrum powered by essence. *SIGSOFT Softw. Eng. Notes*, 41(1):1–8, Feb. 2016.

[42] D. K. Patel, D. R. Doshi, and A. R. Rawal. From heuristics to ai: A deep dive into software project resource allocation. In *2025 International Conference on Artificial Intelligence and Machine Vision (AIMV)*. IEEE, 2025.

[43] M. Paul, F. M. Pranto, A. Al Amin, H. Nur, and S. M. A. Shafi. Hybrid test case prioritization techniques for agile-devops integration. In *Proceedings of the 3rd International Conference on Computing Advancements*, ICCA '24, page 224–231, New York, NY, USA, 2025. Association for Computing Machinery.

[44] PwC. Agile on the rise: Integrating effective controls into agile environments. Technical report, PwC, 2018.

[45] F. Raith, I. Richter, and R. Lindermeier. How project-management-tools are used in agile practice: Benefits, drawbacks and potentials. In *Proceedings of the 21st International Database Engineering & Applications Symposium*, IDEAS '17, page 30–39, New York, NY, USA, 2017. Association for Computing Machinery.

[46] G. Raj, K. Yadav, and A. Jaiswal. Emphasis on testing assimilation using cloud computing for improvised agile scrum framework. In *2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)*, pages 219–225, 2015.

[47] I. Richter, F. Raith, and M. Weber. Problems in agile global software engineering projects especially within traditionally organised corporations: [an exploratory

semi-structured interview study]. In *Proceedings of the Ninth International C* Conference on Computer Science & Software Engineering*, C3S2E '16, page 33–43, New York, NY, USA, 2016. Association for Computing Machinery.

[48] K. H. Rolland, V. Mikkelsen, and A. Næss. Tailoring agile in the large: Experience and reflections from a large-scale agile software development project. In H. Sharp and T. Hall, editors, *Agile Processes, in Software Engineering, and Extreme Programming*, pages 244–251, Cham, 2016. Springer International Publishing.

[49] P. Rosenberger and J. Tick. Suitability of pmbok 6th edition for agile-developed it projects. In *2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI)*, pages 241–246, 2018.

[50] G. S., Y. El-Ebiary, B. S. Rao, R. Rautrao, T. Rao, J. Ramesh, and O. Al-Omari. Challenges and solutions in agile software development: A managerial perspective on implementation practices. *International Journal of Advanced Computer Science and Applications*, 16(3), 2025.

[51] G. L. S, Y. A. B. El-Ebiary, B. S. Rao, R. R. Rautrao, T. S. M. Rao, J. V. N. Ramesh, and O. Al-Omari. Challenges and solutions in agile software development: A managerial perspective on implementation practices. *International Journal of Advanced Computer Science and Applications*, 16(3), 2025.

[52] H. Saeeda, M. O. Ahmad, and T. Gustavsson. Identifying and categorizing challenges in large-scale agile software development projects: Insights from two swedish companies. *SIGAPP Appl. Comput. Rev.*, 23(2):23–43, July 2023.

[53] T. Saxena and M. W. Totaro. Artificial intelligence in project management: Impacts on efficiency, innovation & competitive edge. In *2024 Artificial Intelligence for Business (AIxB)*, pages 80–85, 2024.

[54] S. Schwedt and T. Ströder. From bugs to benefits: Improving user stories by leveraging crowd knowledge with cruise-ac. In *Proceedings of the IEEE/ACM 47th International Conference on Software Engineering*, ICSE '25, page 1385–1395. IEEE Press, 2025.

[55] M. Senapathi and D. Strode. Challenges to sustaining agility: An exploratory case study. In *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing*, SAC '24, page 810–817, New York, NY, USA, 2024. Association for Computing Machinery.

[56] A. Shahriary, M. Sedighi, N. Tajik, M. Shahinfar, and A. R. Asiyabar. Assessing large language models as agile scrum masters: A comparative study of project planning efficiency. In *2025 11th International Conference on Web Research (ICWR)*, pages 150–156, 2025.

[57] A. Singh. Integrating the extreme programing model with secure process for requirement selection. In *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, pages 423–426, 2018.

[58] W. Sisomboon, N. Phakdee, and N. Denwattana. Engaging and motivating developers by adopting scrum utilizing gamification. In *2019 4th International Conference on Information Technology (InCIT)*, pages 223–227, 2019.

[59] G. Smith-Nunes. Agiledbr: Blending industry and academic practice. *IEEE Technology and Society Magazine*, 44(3):80–97, 2025.

[60] F. Sobiech, B. Eilermann, and A. Rausch. A heuristic approach to solve the elementary sprint optimization problem for non-cross-functional teams in scrum. *SIGAPP Appl. Comput. Rev.*, 14(4):19–26, Jan. 2015.

[61] N. K. Sravan, V. A. Vuyyyru, P. Gottapu, A. Theeda, and L. Peddineni. Agile management tools: Technological evaluations and future archetype. In *2024 4th International Conference on Pervasive Computing and Social Networking (ICPCSN)*, pages 914–918, 2024.

[62] I. I. Syed, U. V. Goud, and S. Datta. A survey of agile vs. traditional methods on project risk management. In *2023 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 1583–1587, 2023.

[63] L. Trihardianingsih, A. Alin, M. Istighosah, and M. Asgar. Systematic literature review of trend and characteristic agile model. *JURNAL TEKNIK INFORMATIKA*, 16:45–57, 04 2023.

[64] G. Tsaramirsis and A. Basahel. Scrum to manage humans and intelligent systems. In *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 1353–1356, 2016.

[65] U.S. Government Accountability Office. Gao agile assessment guide: Best practices for adoption and implementation. Technical Report GAO-24-105506, U.S. Government Accountability Office, 2023. Reissued with revisions December 15, 2023.

[66] S. Van Ledden, A. Dogangün, and J. Hermann. Elsi product owner: Integration of ethical, legal and social aspects in agile development processes. In *Proceedings of Mensch Und Computer 2023*, MuC '23, page 313–317, New York, NY, USA, 2023. Association for Computing Machinery.

[67] Z. Wang. Teamworking strategies of scrum team: A multi-agent based simulation. In *Proceedings of the 2018 2nd International Conference on Computer Science and Artificial Intelligence*, CSAI '18, page 404–408, New York, NY, USA, 2018. Association for Computing Machinery.

[68] S. Yordanova and K. Toshkov. An agile methodology for managing business processes in an it company. *Business Management*, 20(3):72–90, 2019.

[69] R. Younisse and M. Azzeh. Application of natural language processing techniques in agile software project management: A survey. In *2023 14th International Conference on Information and Communication Systems (ICICS)*, pages 01–06, 2023.

[70] S. Zapotecas-Martínez, A. García-Nájera, and H. Cervantes. Multi-objective optimization in the agile software project scheduling using decomposition. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, GECCO '20, page 1495–1502, New York, NY, USA, 2020. Association for Computing Machinery.

Towards a Uniform Model for the Secure Software Development Life Cycle

Łukasz Sobczak
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
lukasz.sobczak@rwth-aachen.de

ABSTRACT

Increasing application-level security is a complex issue. A common means to increase security is to start addressing it early in the development process. Therefore, there is a need for a process that takes security into account throughout the whole software development effort. The Secure Software Development Life Cycle (SSDLC) extends the existing Software Development Life Cycle (SDLC) model by incorporating security-oriented activities. Although being a well-established approach in the industry, there is no single, unified model of SSDLC in existing peer-reviewed literature that coherently describes SSDLC. Lack of a unified model makes it difficult for practitioners to follow a consistent process. Currently, they need to utilize multiple sources that differ significantly. This paper fills this gap by providing a unified SSDLC model that incorporates common activities and phases from academic literature. A structured literature review was performed on existing academic sources using earlier defined inclusion and exclusion criteria. Out of 248 papers found, 21 were identified as highly valuable to formulate the results. The results show some agreement between the sources, especially in the pre-development phases. There were also multiple disagreements that this paper aimed to resolve - mainly inconsistent naming and different categorization of activities. This review analyses those inconsistencies and creates a coherent, uniform model for SSDLC.

Keywords

Secure Software Development Life Cycle, Software Engineering, Application-level Software Security

1. INTRODUCTION

The topic of security in software development is an increasingly important issue [18]. The digitization of every aspect of life is progressing at a fast pace. From private companies, offering online solutions for a variety of problems through banking and government activity, reliable software is needed everywhere [28]. The software security topic became even more critical in the era of AI, since there is more software developed than ever, and this software is not always fully correct [29]. This rapid development forced organizations to adopt new strategies with the goal of decreasing different types of software-related risks. Despite ongoing efforts to improve software security, the number of reported vulnerabilities continues to increase year over year [18]. Therefore, there is clearly a need for a complete and standardized approach in Software Development throughout the whole application lifetime.

Software development has been evolving rapidly, and the models guiding the development have been changing over time. More traditional SDLC models, such as Waterfall or the V-model [27], have been more focused on clearly defined inputs and outputs to each phase and assumed that development is a linear rather than iterative process. This provides very little room to make changes in artifacts that were already produced; therefore costs incurred by such changes were high, especially at the later stages. The answer to this issue was iterative models, such as Scrum or Extreme Programming [27], which focused on fast user feedback and embraced change throughout the project lifetime. At the time of creation of the paper usage of traditional models was rare in the industry; therefore, the paper focuses on the later approach.

Despite numerous approaches to integrating security across different types of organizations, such as Microsoft Software Development Lifecycle [16], Cybersecurity Framework from National Institute of Standards and Technology [20] and OWASP Developer Guide from Open Worldwide Application Security Project [21], there is no established, unified academic approach to Secure Software Development Life Cycle (SSDLC). The purpose of this paper is to provide a high-level, phase-based overview of security-oriented activities that can be integrated into the Software Development Life Cycle (SDLC). To achieve this, a review of existing academic sources regarding application-level security activities within the software industry was conducted.

This paper is structured as follows: Section (2) presents work related to this study, Section (3) presents the method used in this research. Section (4) presents the results of the study. It first defines SSDLC and later identifies each phase with activities belonging to that phase. Section (5) discusses the results. Section (6) concludes the study.

2. RELATED WORK

Throughout modern history of computer science, there

SWC Seminar 2025/2026 RWTH Aachen University, Germany.

was an ongoing effort in increasing the security of the created software products, both in academic and industrial research. There are plenty of developed and widely used guidelines and methods in the industry, such as Microsoft SDL [16], OWASP's Software Assurance Maturity Model [22], and NIST's Secure Software Development Framework [19]. There is also a large amount of scientific research, which very often focuses on one specific part of SDLC or discuss specific type of threat and possible mitigation, without a unified model that should be utilized within SDLC. Bernsmed et al. [6] discusses threat modeling in the context of agile software development, Wang et al. [32] writes about a modern approach to penetration testing, and Deepa et al. [8] describes state of the art approaches of defending against specific threats. There are also sources that aim to present some form of SSDLC model. Fujdiak et al. [10] presents a model with few activities added on top of SSDLC, but it presents them very briefly without clearly defining how and why those activities are a part of SSDLC model. Futcher et al. [11] provides guidelines related to some of the SSDLC activities based on standards and NIST, but it does not clearly define what SSDLC is and how is it structured.

Although the number of scientific publications on the topic of application-level software security is large, a lack of a unified SSDLC model presents a clear gap in peer-reviewed sources, even though there are models originating from the industry that are widely used.

3. METHOD

This study uses a systematic literature review methodology following the guidelines proposed by Kitchenham et al. [15] to formulate a unified SSDLC model based on the existing literature. Peer-reviewed publications were searched in the selected scientific databases using pre-defined keywords. Found publications were then screened for relevance by predefined inclusion and exclusion criteria. Relevant publications were analysed to identify the activities that SSDLC includes and to formulate SSDLC definition.

3.1 Research Questions

To create a uniform model for the SSDLC, it is first necessary to formulate a definition of SSDLC based on academic sources. The second step is to identify and define the activities that compose SSDLC and to categorize them by SSDLC phases.

Main research question that this study aims to answer is: **(RQ0):** What is the Secure Software Development Life Cycle (SSDLC)?

To answer this question, two more granular research questions were formulated, which together provide an answear to RQ0:

(RQ1): What phases does the SSDLC consist of?

(RQ2): What activities constitute SSDLC and which phases are they classified into?

3.2 Data source selection

This study used established academic research databases: IEEEXplore, ACM Digital Library, and ScienceDirect.These databases were selected because they cover the research domain and are the most established ones in the Software Development field. Only peer-reviewed publications were considered for this study. From those sources, a total of 248 sources were initially found, of which 38 were selected for detailed review based on inclusion and exclusion criteria defined in subsection 3.4.

3.3 Search strategy and keywords

Search keywords are "Secure Software Development Lifecycle" and "Secure Software Development Life Cycle". This study is purely focused on SSDLC and these keywords reflect this purpose. Althoug the literature has no unified naming, these terms were chosen, because they capture majority of studies that focus on SSDLC.

3.4 Inclusion and exclusion criteria

To filter found publications without bias, a set of inclusion criteria in Table 1 and exclusion criteria in Table 2 has been formulated.

ID	Inclusion criteria
I1	Studies published between 2007 and 2025. This time window was selected since publications before referred to an outdated software development process.
I2	Peer-reviewed conference papers or journals. Selecting only peer-reviewed publications ensures scientific rigor and data quality.
I3	Entirely written in English. The author's preferred language is English, and this avoids errors in understanding related to translation.
I4	Studies that describe SSDLC or security-related activities in SDLC. This ensures that studies are directly related to stated research questions.
I5	Text is available. Publication accessibility ensures that it can be properly analysed and makes it possible for everyone to check the sources of this paper.

Table 1: Inclusion criteria

ID	Exclusion criteria
E1	The paper discusses a strategy to educate software developers and not the SDLC itself. Papers discussing education strategies do not provide information directly about the development process.
E2	The paper focuses on code-level strategy. This paper focuses on project management level strategies, and code-level strategies do not provide any more information on that topic.
E3	The paper focuses on organization-level policies. The scope of this paper is within the lifetime of a certain project, organization level policies do affect the project, but not in a direct way.
E4	The paper describes methods for specific threat prevention. The aim of SSDLC is not to protect against specific vulnerabilities, but rather provide a guideline that increases the overall level of security, so specific threat types are out of scope.

Table 2: Exclusion criteria

3.5 Data extraction and analysis

Data were extracted from each study using a structured approach. First papers were searched using 3 types of queries: targeting only the title, only the keywords, or only the abstract. This search revealed 248 potentially relevant publications. Each of those publications were then screened by title, abstract, conclusions and contents in this exact order. During the screening the publications were discarded based on inclusion Table 1 and Table 2. This screening process revealed 21 relevant publications that were used to formulate the results. Phases within SSDLC are major stages during the software development process. Each of them consists of activities that focus on a similar goals, aligned with the Phase purpose. Phases were identified separately in each source and later analyzed in terms of frequency. The SSDLC phase structure used across this paper is based on phases identified in the the majority of sources. Activities were grouped by each phase, also based on to which phase majority of sources classified given activity. Each activity were then analysed in terms of differences and common points across the sources. Activities with varying names were mapped into the most frequently used names based on the description and name of the activity. Frequency of each activity was counted to identify which activities were the most common ones.

4. RESULTS

This section presents the results based on conducted systematic literature review and presents a unified SSDLC model constructed from 21 reviewed publications.

Analysis of the papers found resulted in formulation of five phases within SSDLC model: Planning and Analysis, Design, Implementation, Testing, Deployment and Operation. While naming differed across the sources, these phases were most commonly identified and represent common abstraction that was used to describe the development process. Each of the activities provided by the sources was then further categorised into those phases. Table 3 presents all SSDLC activities categorized into five phases. For each activity there is a list of sources that mention specific activity that makes it possible to quantify the amount of attention each activity and phase got in the scientific sources.

From the data sources, it was found that in mentioned activities, only part of them is performed solely for security reasons, while the rest incorporates only elements or tasks that are security oriented. Code Review (4.4.2) is an example of task that is performed not just for security, but also for maintenance, performance, etc. Thus this paper includes categorization of the activities in those terms - activities that are performed not only for security reasons are marked with an asterisk (*).

4.1 SSDLC

The SSDLC is a model that extends the existing SDLC model and adds security-related activities to it, but it does not replace or remove any SDLC activities. It is intended to be used throughout all SDLC phases where security-focused tasks are performed alongside the standard SDLC. This approach allows teams to treat security as an integrated part of the software and not as an additional layer added on top of the software. This way is proven to detect security vulnerabilities and bugs early, even before the code is created, by that reducing the overall cost of the project [26]. Since SSDLC significantly increases the load of already complex SDLC, therefore some structured management approach is usually required [10]. SDLC as well as SSDLC can be viewed as a process divided into phases. Phases identified for the purpose of this paper are: Planning and Analysis, Design, Implementation, Testing, and Deployment and Operation. The iterative process of SSDLC is visualised in Figure 1.

4.2 Planning and Analysis Phase

In the reviewed papers, the planning and analysis phase is frequently pointed out as a very important starting point for introducing security considerations. Out of 21 papers, 19 point to the planning phase as a starting point for introducing security-related activities besides those typical in SDLC. Although terminology differs between the sources - some of the publications refer to it as requirements phase - most of them emphasize the need to account for security from the beginning. Besides standard SDLC activities, reviewed papers most frequently focus on formulating security requirements, discussed in 90% of reviewed papers; performing risk analysis, 43% of the reviewed papers; detailing misuse cases, in 38% and setting quality gates in 5%.

4.2.1 Security Requirements Elicitation

Security requirements elicitation activity focuses on producing security requirements alongside functional and non-functional requirements. Security requirements are security concerns that should be considered when building software. All potential security risks in a system should be identified and documented. It is important to identify them as early as possible since they will be used in further phases of SSDLC. When forming security requirements, a baseline list of security requirements can be used, such as one provided in OWASP CLASP [11]. Requirements need to be verified in terms of quality and suitability for the specific project, which is usually ensured by a special validation process.

4.2.2 Security Risk Assessment

Security risk assessment is a broad activity aiming to evaluate all identified security issues. Multiple different approaches are used across the industry, the DREAD model is an example of such an established method [30]. Risk analysis is a part of risk assessment. It aims to rank architectural flaws by severity, so the mitigation process will have priorities. Ignoring risk assessment leads to problems with high cost later. This activity can also be partially performed in the design phase, as it is connected with threat modeling [2].

4.2.3 Detailing misuse cases

Detailing misuse cases activity focuses on creating abuser and security stories, where a bad actor aims to execute malicious action against the system [23]. These stories can then be used to write corresponding automated tests checking those scenarios.

*4.2.4 Setting Quality Gates**

Setting quality gates aims to create checkpoint definitions that define certain quality criteria that the system has to pass to proceed. Although this activity is not purely security related, it can be used to control quality of produced features before proceeding which forces a more detailed checks and thus decreases chance of security bugs [24].

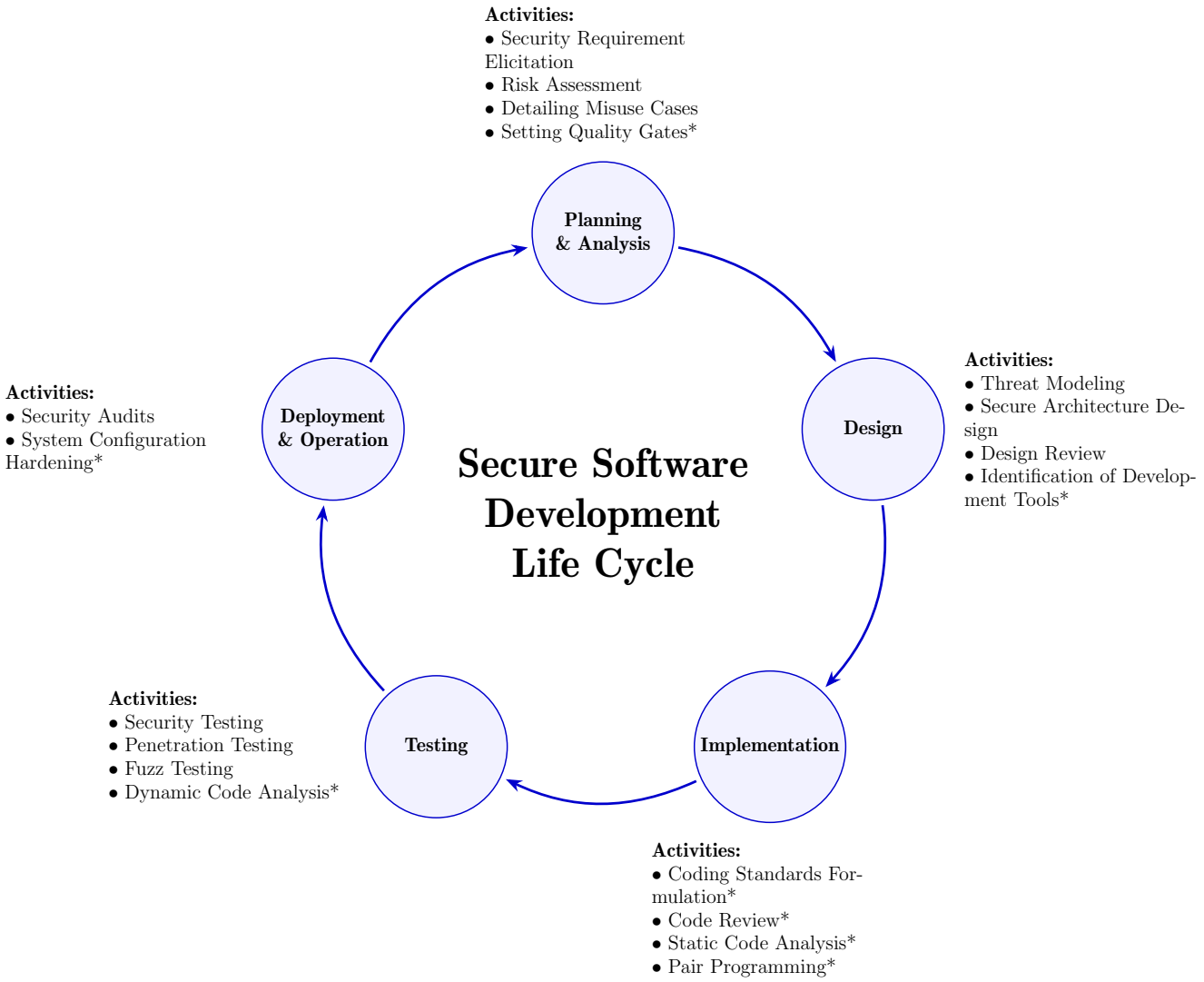


Figure 1: Proposed Unified SSDLC Model

4.3 Design Phase

Design phase focuses on designing the architecture and components of the system. Security practices need to be integrated in architecture of the system instead of being something added on top of an already designed system. Threat modeling is discussed in 67% of reviewed papers; secure architecture design in 33%; design review in 14%; identification of development tools in 10%.

4.3.1 Threat Modeling

Threat modeling process aims at analyzing which parts of the system are exposed to the highest risk and identifying the threats that can harm them. Results of this process are a driving factor of the system's security architecture design, code review, and testing [11]. There are proven approaches to threat modeling, which include attack trees, using the STRIDE model and the DREAD model [33][11]. Threat modeling is performed on different detail levels, from a specific algorithm used to a system's component [12].

4.3.2 Secure Architecture Design

Security architecture design aims to enhance the security by modeling the system in such a way that identified security risks through the design of its components. System should be immune to multiple classes of exploits by design. The list of recommended software frameworks to be used in the system should be created. Security design patterns should be identified and promoted for the implementation [24] [7]. There is a variety of strategies in securing systems' components that are available in modern development frameworks. One advised approach is to utilize a multilayer architecture and to use appropriate security mechanisms at different layers of abstraction.

*4.3.3 Design Review**

Design review is a structured evaluation of the artifacts produced in the design phase. It should be performed by experts and take into account requirements created during planning and analysis phase and check if all identified security flaws. Design review is also used to assess the quality of the artifacts, thereby it is not strictly a security related

Phase	Activity	Papers	Count
Planning and Analysis	Security Requirement Elicitation	[7][23][4][10][11][14][13][12][25][24][26][34][1][17][31][33][30][3][2]	19
	Risk Assessment	[23][4][10][11][12][25][1][17][2]	9
	Detailing Misuse Cases	[23][26][17][31][33][2][14][25]	8
	Setting Quality Gates*	[24]	1
Design	Threat Modeling	[23][10][5][11][12][25][24][26][1][17][31][33][30][2]	14
	Secure Architecture Design	[7][9][23][10][24][1][31]	7
	Design Review*	[10][24][23]	3
	Identification of Development Tools*	[4][12]	2
Implementation	Coding Standards Formulation*	[7][23][11][25][24][1][31][33][30][2]	10
	Code Review*	[7][23][10][11][12][25][24][26][34][1][33][30][2]	13
	Static Code Analysis*	[7][23][4][10][5][14][13][12][25][1][17][31][33][30][3][2]	16
	Pair Programming*	[23]	1
Testing	Security Testing	[7][9][23][10][14][12][25][26][34][1][17][31][30][2]	14
	Penetration Testing	[7][4][10][12][25][24][26][1][31][33][3][2]	12
	Fuzz Testing*	[23][4][12][25][24][26][1][2]	8
	Dynamic Code Analysis*	[7][23][4][10][14][12][25][24][1][17][30][3][2]	13
Deployment and Operation	Security Audits	[23][12][25][24][1][31][2]	7
	Security Configuration Hardening*	[23][10][12][25][26][7][4][34][1][31][33][9]	12

Table 3: Activities and sources per SSDLC phase

activity [10].

4.3.4 Identification of Development Tools*

Identification of development tools aims to plan the development environment and select the tools that will support the development. Those tools should be checked for possible security flaws, especially when it comes to tools that access the code directly, for example continuous integration system [12]. This can prevent security incidents during implementation and testing phases.

4.4 Implementation Phase

Implementation phase focuses on producing the system's source code. The process is based on earlier defined requirements and architecture decisions, while taking into account identified security threats. Multiple approaches can be used to reduce the risk of security vulnerabilities in the produced system. Most frequently advocated include coding standards formulation, reported by 47% of the reviewed publications; practicing code review, mentioned in 61% of the sources; scanning the source code using static code analysis tools, advised in 76% of the sources; and practicing pair programming, 5% of the sources.

4.4.1 Coding Standards Formulation*

Coding standards formulation is an activity that produces programming guidelines and rules that help software developers create more maintainable, scalable and secure solutions. Entries that include security considerations are a good strategy in reducing the number of flaws in the produced code, thus leading to fewer security vulnerabilities [11]. Developer teams should have a clear coding standards baseline that is being followed and checked in code review. An example of such a rule can be the deprecation of unsafe functions with a defined replacement solution and rationale [2].

4.4.2 Code Review*

Code review is a process of reading and looking for issues in code, done by a professional. They are a good way of finding mistakes in implementations of correct designs, not necessarily related to security, but also maintainability and scalability of the solution. The code should be reviewed before being committed to the main branch, and the development environment should allow tracking all reviewed changes [7]. Issues that are found in code reviews are usually of a different nature than issues found in testing, so the former cannot be replaced by the latter. They are also a good way of sharing knowledge about security vulnerabilities and secure coding practices between engineers. Code reviews can be further improved by creating a review checklist and identifying high-risk code, which is reviewed in more detail [24].

4.4.3 Static Code Analysis*

Static code analysis tools are able to calculate several code quality metrics, which help in identifying code that should be improved in terms of readability, security or performance by refactoring. Among the tools that provide such functionality is SonarQube, which provides support for wide range of programming languages [12]. Static code analysis can be further automated by running it as a part of a Continuous Integration (CI) service [24].

4.4.4 Pair Programming*

Pair programming is a coding practice where two software developers work together (at a single workstation) at the same time on the same task. It aims to increase code quality, but is rarely used [23]. This practice can help in producing safe and correct code, especially in more complex parts of the system.

4.5 Testing Phase

Testing phase aims to verify whether the system produced in the implementation phase satisfies the requirements. This is done mainly by testers who run various types of manual and automatic tests on the system. Besides tests typical for SDLC, SSDLC adds a few types of verification. The most popular ones are various forms of security testing, described in 67% of reviewed papers; fuzz testing, included in 38% of the papers; penetration testing, advised by 57% of the papers; and dynamic code analysis, 62% of the papers.

4.5.1 Security Testing

Security tests are tests focused on verifying the established security requirements. This should be done in a dedicated testing environment that is very similar to the production environment. A series of automated tests from security test suites should be executed. This test suite, apart from testing the initial security requirements, should include some findings from development and other testing activities [12][30]. Similarly to coding standards, testers should use testing standards and best practices in black/gray/white box tests to increase the chance of detecting vulnerabilities [11].

4.5.2 Penetration Testing

These tests aim to breach the system the same way as an attacker would. Results from such tests can be used to patch vulnerabilities and improve the system's security. Sometimes considered a part of security testing (4.5.1).

4.5.3 Fuzz Testing*

Fuzz testing is a computationally expensive technique of verifying a program's behavior on large amounts of random, invalid, or malformed input data. This type of verification is often viewed as costly and not as beneficial as other techniques, but it is often advised as a complementary strategy in software testing [23]. Whether to include fuzz testing in the testing phase depends largely on the system type, its purpose and used abstraction level, with low-level systems often being well suited for fuzz testing [24][2].

4.5.4 Dynamic Code Analysis*

Dynamic code analysis is a method of examining a system's behavior while it is running. It can find issues between different software components, something that static analysis can't do [17]. Dynamic code analysis is also often used with goals different than security, for example performance.

4.6 Deployment and Operation phase

Deployment and operation phase is the part where the production environment is being created and later operated. The main tasks that need to be done revolve around ensuring the correct configuration of system's components and appropriate observability setup. Security audits are mentioned in 33% of the sources and security configuration hardening in 57%.

4.6.1 Security Audits

Security audits are an external form of verification used to assess whether a system complies with the expected security standards. They are typically conducted by an independent, external expert who review the system's source code and documentation. This activity is an expensive approach and should be performed after internal testing is done [10].

4.6.2 Security Configuration Hardening*

Security configuration hardening is an activity of improving and securing the configuration to meet the security requirements. Configuration should be created to correctly secure the system, database and setup backups. This includes setting up logging and monitoring correctly and securely, that the application state is visible in real time as well as logs can be used to investigate when an issue appears. The final configuration should be reviewed and evaluated by an expert, audit can be performed to check it [10].

5. DISCUSSION

This study examined existing reviewed sources regarding SSDLC structure and which activities are most often included in the SSDLC. From the referenced literature, it is obvious that SSDLC is an integrated part of SDLC and should be used across all phases. Most focus goes to the beginning of the project, as multiple sources state that security should be included as a part of the project from the beginning and not added after an incident happens.

One of the key finding is strong bias towards importance of early phases in SSDLC. The majority of sources agree that addressing security as early as possible is crucial to creating a secure project. Also early phases contain majority of activities related purely to security, which further emphasizes the importance of adopting security practices early into the project. When designing the system, the earlier a security vulnerability is detected, acknowledged and further development takes it into account, the lower the resulting cost of such an issue - the difficulty of changing the implementation in a specific way increases with the project size.

Some of the activities, most notably Setting Quality Gates (4.2.4), (4.3.3) and (4.4.4) remain underexplored in the context of SSDLC, with very little sources mentioning and discussing those. At the same time Security Requirement Elicitation (4.2.1), Threat Modeling (4.3.1) and (4.4.3) are discussed in detail in over 50% of the sources. This imbalance suggests that greater effort is put into discussing those activities, perhaps because of the overall impact they have on the project's security.

Sources differ in terms of terminology used, phases included in the SSDLC, and which activities belong to which phase, but this paper tried to create coherent descriptions of each activity, taking into account all relevant sources. As pointed out by Neil et al. terminology in cybersecurity space is not unified and this is clearly seen in this paper.

It is worth noting that Microsoft SDL was referenced multiple times throughout different papers as a source for SSDLC practices. While Microsoft is not the only large company researching and using SSDLC, it is most commonly cited by scientific sources that discuss SSDLC. Specific phases also referenced OWASP SAMM and CLASP, as well as Common Criteria.

5.1 Limitations

When gathering data from multiple sources, it could have been possible that two distinct, but closely related activities were understood and presented as a single one. Amount of sources is relatively small for using it as a base to create a uniform SSDLC model, especially when there is a lot more information about it in grey literature that was not considered. Differences in terminology in the sources may have been interpreted incorrectly. This added a risk of miss categorizing activities. The field of software security is evolving rapidly, there is not enough peer-reviewed academic literature on the on security impact of generative AI, so this publication may not be up to date in terms of industry standards. This paper only provides a proposal of SSDLC model which was not validated through case studies and experiments, so it needs further work to be fully utilized.

6. CONCLUSIONS

This paper showed a unified and structured Secure Software Development Life Cycle (SSDLC) model. Structured approach based on systematic literature review was used to gather data from 21 publications. This study identified activities composing SSDLC and categorized them into phases.

The proposed gathers SSDLC sources and creates a unified model which can be utilized in further studies on SSDLC. The model identifies and describes phases of SSDLC and categorizes activities based on those phases while also considering whether activities are performed purely for security reasons.

This paper is a first draft of a unified SSDLC model and it lays ground work for further work on SSDLC model, such as practical application and additional refinement. While based on the literature this model needs to be further studied in practice, fur example by surveing the practitioners and applying the model in a real project while observing the results.

7. REFERENCES

[1] H. A. Al-Hashimi, R. A. Khan, H. S. Alwageed, A. M. Algarni, S. Ayouni, and A. O. Almagrabi. Exploring the role of generative ai in enhancing cybersecurity in software development life cycle. *Array*, 28:100509, 2025.

[2] D. Baca and B. Carlsson. Agile development with security engineering activities. In *Proceedings of the 2011 International Conference on Software and Systems Process*, ICSSP '11, page 149–158, New York, NY, USA, 2011. Association for Computing Machinery.

[3] A. Barabanov, A. Markov, A. Fadin, V. Tsirlov, and I. Shakhalov. Synthesis of secure software development controls. In *Proceedings of the 8th International Conference on Security of Information and Networks*, SIN '15, page 93–97, New York, NY, USA, 2015. Association for Computing Machinery.

[4] A. Barabanov, A. Markov, and V. Tsirlov. Procedure for substantiated development of measures to design secure software for automated process control systems. In *2016 International Siberian Conference on Control and Communications (SIBCON)*, pages 1–4, 2016.

[5] A. Bekturova, K. Kursabayeva, Z. Salamatkyzy, S. Kusdavletov, and B. Azibek. Defender-attacker model based secure software development lifecycle for common web vulnerabilities. In *2023 IEEE International Conference on Smart Information Systems and Technologies (SIST)*, pages 272–277, 2023.

[6] K. Bernsmed, D. S. Cruzes, M. G. Jaatun, and M. Iovan. Adopting threat modelling in agile software development projects. *Journal of Systems and Software*, 183:111090, 2022.

[7] R. Brasoveanu, Y. Karabulut, and I. Pashchenko. Security maturity self-assessment framework for software development lifecycle. In *Proceedings of the 17th International Conference on Availability, Reliability and Security*, ARES '22, New York, NY, USA, 2022. Association for Computing Machinery.

[8] G. Deepa and P. S. Thilagam. Securing web applications from injection and logic vulnerabilities: Approaches and challenges. *Information and Software Technology*, 74:160–180, 2016.

[9] E. B. Fernandez and X. Yuan. Securing analysis patterns. In *Proceedings of the 45th Annual ACM Southeast Conference*, ACMSE '07, page 288–293, New York, NY, USA, 2007. Association for Computing Machinery.

[10] R. Fujdiak, P. Mlynek, P. Mrnustik, M. Barabas, P. Blazek, F. Borcik, and J. Misurec. Managing the secure software development. In *2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–4, 2019.

[11] L. Futcher and R. von Solms. Guidelines for secure software development. In *Proceedings of the 2008 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries: Riding the Wave of Technology*, SAICSIT '08, page 56–65, New York, NY, USA, 2008. Association for Computing Machinery.

[12] D. Granata, M. Rak, and G. Salzillo. Metasend: A security enabled development life cycle meta-model. In *Proceedings of the 17th International Conference on Availability, Reliability and Security*, ARES '22, New York, NY, USA, 2022. Association for Computing Machinery.

[13] R. A. Khan and S. U. Khan. A preliminary structure of software security assurance model. In *Proceedings of the 13th International Conference on Global Software Engineering*, ICGSE '18, page 137–140, New York, NY, USA, 2018. Association for Computing Machinery.

[14] R. A. Khan, S. U. Khan, M. Ilyas, and M. Y. Idris. The state of the art on secure software engineering: A systematic mapping study. In *Proceedings of the 24th International Conference on Evaluation and Assessment in Software Engineering*, EASE '20, page 487–492, New York, NY, USA, 2020. Association for

Computing Machinery.

[15] B. Kitchenham. Procedures for performing systematic reviews. *Keele, UK, Keele Univ.*, 33, 08 2004.

[16] Microsoft Corporation. Microsoft Security Development Lifecycle (SDL). `https://www.microsoft.com/en-us/securityengineering/sdl`, 2026. Accessed: 2026-01-19.

[17] N. M. Mohammed, M. Niazi, M. Alshayeb, and S. Mahmood. Exploring software security approaches in software development lifecycle. *Comput. Stand. Interfaces*, 50(C):107–115, Feb. 2017.

[18] National Institute of Standards and Technology. National Vulnerability Database (NVD) Statistics. `https://nvd.nist.gov/general/nvd-dashboard`, 2026. Accessed: 2026-01-19.

[19] National Institute of Standards and Technology. Secure Software Development Framework (SSDF). `https://csrc.nist.gov/projects/ssdf`, 2026. Accessed: 2026-01-19.

[20] National Institute of Standards and Technology (NIST). Cybersecurity framework (nist csf) resource center. `https://www.nist.gov/cyberframework`, 2026. Official resource page for the NIST Cybersecurity Framework, including CSF 2.0 guidance and supporting materials; accessed 2026-01-25.

[21] OWASP Foundation. Owasp developer guide. `https://owasp.org/www-project-developer-guide/`, 2026. Official project page for the OWASP Developer Guide, an accessible introduction to security concepts for application and system developers; accessed 2026-01-25.

[22] OWASP Foundation. OWASP Software Assurance Maturity Model (SAMM). `https://owasp.org/www-project-samm/`, 2026. Accessed: 2026-01-19.

[23] K. Rindell, S. Hyrynsalmi, and V. Leppänen. Busting a myth: Review of agile security engineering methods. In *Proceedings of the 12th International Conference on Availability, Reliability and Security*, ARES '17, New York, NY, USA, 2017. Association for Computing Machinery.

[24] K. Rindell, S. Hyrynsalmi, and V. Leppänen. Aligning security objectives with agile software development. In *Proceedings of the 19th International Conference on Agile Software Development: Companion*, XP '18, New York, NY, USA, 2018. Association for Computing Machinery.

[25] K. Rindell, J. Ruohonen, and S. Hyrynsalmi. Surveying secure software development practices in finland. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*, ARES '18, New York, NY, USA, 2018. Association for Computing Machinery.

[26] A. Selva-Mora and C. Quesada-López. Security practices in agile software development: A mapping study. In *Proceedings of the 7th ACM/IEEE International Workshop on Software-Intensive Business*, IWSiB '24, page 56–63, New York, NY, USA, 2024. Association for Computing Machinery.

[27] I. Sommerville. Software engineering (ed.). *America: Pearson Education Inc*, 2011.

[28] Statista. Global internet penetration rate 2009–2024, by region (statista #265149). `https://www.statista.com/statistics/265149/internet-penetration-rate-by-region/`, 2024. Accessed: 2026-01-25; Published: May 2024; Based on worldwide survey data 2009–2024; Europe had highest penetration at 91%, Africa the lowest at 38%.

[29] Statista. Software development: App developers statista dossier (study no. 15207). `https://www.statista.com/study/15207/app-developers-statista-dossier/`, 2024. Statista report on software and app development market and trends; released 2024; accessed 2026-01-25.

[30] Y.-H. Tung, S.-C. Lo, J.-F. Shih, and H.-F. Lin. An integrated security testing framework for secure software development life cycle. In *2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 1–4, 2016.

[31] A. V. Uzunov, E. B. Fernandez, and K. Falkner. Ase: A comprehensive pattern-driven security methodology for distributed systems. *Computer Standards Interfaces*, 41:112–137, 2015.

[32] Y. Wang, S. Liu, W. Wang, C. Zhou, C. Zhang, J. Jin, and C. Zhu. A unified modeling framework for automated penetration testing. *Computers Security*, 162:104787, 2026.

[33] H. R. Wicaksono, I. F. Tampati, N. B. N. Putra, H. Setiawan, and D. R. Firmansyah. A design for comprehensive information system management framework integrating secure software development, resource management, and real-time monitoring. In *2024 7th International Conference on Informatics and Computational Sciences (ICICoS)*, pages 209–214, 2024.

[34] Y. Xie, G. Tian, Y. Tao, G. Li, and Q. Hu. Research on application development and implementation method based on "platform +app" mode. In *Proceedings of the 2021 5th International Conference on Electronic Information Technology and Computer Engineering*, EITCE '21, page 689–694, New York, NY, USA, 2022. Association for Computing Machinery.

Software Modernization and Change Management: A Mapping Study of Their Intersection

Zain Sajid
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
zain.sajid@rwth-aachen.de

ABSTRACT

Successful software modernization projects require addressing both human and organizational factors alongside technical aspects, yet many projects fail when these dimensions are treated in isolation. At its core, software modernization represents an organizational change process that requires managing how stakeholders transition to new systems, technologies, and practices. Change management, a well-established research field, offers systematic approaches to help organizations navigate such transitions and ensure successful adoption of change. While it provides proven frameworks for managing organizational changes, assessments of how these frameworks apply to software modernization settings remain limited. To address this gap and examine how change management principles can improve software modernization outcomes, a systematic mapping study was conducted to identify connections between the two domains and determine how they complement each other. The study examines research articles spanning both domains and compares their fundamental characteristics, including processes, tools, stakeholder involvement, and evaluation methods, to identify similarities and differences that determine how change management frameworks can be applied in software modernization settings. This comparison reveals several overlapping characteristics, including the critical role of stakeholder engagement, the need for skill development and upskilling, and recurring challenges related to organizational resistance and knowledge transfer during transitions. Taken together, these findings suggest that change management approaches are well suited to software modernization initiatives, as modernization primarily targets technical system evolution, whereas change management emphasizes organizational adoption, offering practitioners systematic approaches to address the human and organizational dimensions critical to project success.

FsSE Seminar 2025/26 RWTH Aachen University, Germany.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management—*change management, software modernization, stakeholder management*; D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*modernization, legacy systems*

Keywords

Software modernization, Change management, Legacy systems, Systematic mapping study, Stakeholder engagement, Organizational factors, Human factors

1. INTRODUCTION

Software systems in organizations around the world are aging. While regular updates and patches address immediate issues, many critical systems require fundamental architectural changes to remain viable. Some challenges faced by these legacy systems include integrating new technologies, handling increasing demands, and meeting modern security requirements. At the same time, organizations recognize that modernization is essential for competitiveness, safety, and cost efficiency.

The goal of these modernization projects is to add new technology to make systems easier to use and more helpful for the organization while preserving important business logic and data accumulated over several years [3]. Critical systems, including those built decades ago as well as more recent ones, now struggle to keep up with growing demands, work with newer technologies, and meet today's security standards [2]. Organizations know they need to modernize these systems to stay competitive, keep operations safe, and control costs.

However, modernization projects often run into problems that prevent them from succeeding. Many fail completely, go over budget, take longer than planned, or don't deliver the improvements they promised. Research shows that technical issues aren't the only reason these projects fail [16]. Organizational and human factors, such as organizational restructuring and how well people adapt to change, are just as significant in determining whether modernization succeeds or fails.

Modernizing software today requires more than just technical solutions. Organizations must manage the human and organizational aspects of change, which include building stakeholder support, training employees, and establishing clear communication throughout the modernization process [12]. When modernization initiatives neglect these factors

and focus exclusively on technology, they risk encountering resistance from staff, losing institutional knowledge, and failing to achieve the intended business outcomes [19]. This raises the question of whether and how change management strategies, which prioritize these human and organizational aspects, might influence modernization outcomes [12].

Organizations attempting to address these organizational and human challenges have increasingly explored combining technical modernization roadmaps with structured change management frameworks [16]. Such approaches may include early stakeholder analysis, co-creation of a shared modernization vision, and incremental delivery of visible benefits [12]. Some initiatives also incorporate feedback loops, organizational structure adjustments, and continuous communication mechanisms into the modernization program [19]. However, the relationship between these change management practices and technical modernization strategies remains underexplored in the literature.

This paper explores the intersection of software modernization and change management by comparing the characteristics of technical modernization strategies with those of change management approaches [1]. Key characteristics in both domains are analyzed to identify potential areas of alignment and divergence, examine challenges that emerge during transformation [16], and compare practices that organizations employ throughout modernization initiatives [12]. Through this comparative analysis, the aim is to understand how these two domains relate to one another and what implications this relationship might have for practitioners navigating legacy system modernization.

2. RELATED WORK

This section presents existing terms and related work from the research fields related to software modernization and change management. The papers discussed in this section serve as contextual background to establish foundational concepts and terminology in both domains. These papers are not part of the systematic analysis presented in the Results section (Section 4), which is based on the 24 papers identified through the systematic mapping study methodology described in Section 3. Notably, examples in the literature that explore change management strategies for modernization are limited to very specialized fields within software engineering.

Software modernization encompasses strategies for transforming legacy systems to meet contemporary technological and business requirements. Bakar et al. [3] identify six implementation phases in modernization, including planning, requirements determination, design, development, testing, and implementation. Different modernization approaches serve distinct purposes depending on organizational needs and system characteristics. Primary motivations include reducing maintenance costs, increasing system flexibility, addressing technology obsolescence, and enabling integration with modern platforms. M'baya et al. [14] propose a framework addressing the full modernization lifecycle through qualification, selection, transition, and validation phases. Koskinen et al. [10] reveal through empirical study that while economic evaluation is pursued in modernization decisions, quantifying benefits remains challenging and decisions often rely partly on expert judgment.

Change management is about optimizing the process of change from the beginning state to the final goal. The content of the goal itself is not included in the definition [11]. There are three starting points for change management: structure, culture and the individual. The implementation of change always requires the active support of employees [11]. In addition to the dimension of people, technology and processes are the other integral dimensions [15].

While change management theories apply broadly across organizational contexts, their application to software modernization remains relatively underexplored. Enterprise Resource Planning (ERP) systems represent one of the few contexts where the intersection of software modernization and change management has been explicitly examined. ERP transformations involve both upgrading complex technical systems and fundamentally changing how organizations work, making them a natural domain where both technical and organizational change approaches must be addressed. Research in this area demonstrates that change management frameworks are increasingly being integrated into ERP modernization efforts. Prakash and Pinto [17] provide a table that displays where change management has been referenced in ERP transformations. Seven well-known change management models have been applied to various parts of ERP modernization. One example is the ADKAR model by Jeff Hiatt [6] used to focus on preparing employees for the successful deployment of the modernized ERP system by Leppänen [13]. The ADKAR model is especially well suited for ERP implementations, as its focus lies on the people involved in change and its connection to project success [17]. Successful ERP transformation depends on both technology and the people involved and is an organizational change. The integration of change management helps to prepare, engage and successfully complete the system modernization [17].

Wendland et al. [20] identify the need to integrate change management into their modernized cloud system to address organizational challenges that arise during cloud migration. These and other tasks have to be resolved in the industry [20]. Addressing this need, Yusof [21] describes change management strategies that are essential for cloud migration: leadership support, phased implementation, performance incentives, and the dynamic process of continuous improvement. The author emphasizes that in addition to the human elements addressed by change management, technical aspects are important as well. Cloud migration is an organizational transformation and a technical modernization [21].

Hayretci and Aydemir [5] observed and compared the legacy system migration of three large retail banks between 2014 and 2020. They found that successful migration depends on technological expertise as well as effective change management practices to deal with risks and transitions. The identified best practices include the phased migration approach, the usage of hybrid systems during transition, and robust project management techniques.

Beyond organizational and strategic considerations, changing requirements play a crucial role in software modernization [3]. The process of analyzing the old system requirements and synthesizing the new ones are two of the six implementation phases characterized by Bakar et al. [3]. There are various change management activities that have been developed to manage requirements change [3].

3. RESEARCH METHOD

To explore the relationship between software modernization and change management, this study conducts a systematic mapping study to identify connections between the two domains and compare their key characteristics. The approach involves systematically reviewing literature from both domains to identify and compare their defining characteristics, practices, and challenges. This comparative analysis enables the identification of potential alignments, divergences, and areas where change management approaches may be applicable to software modernization contexts. A systematic mapping study was chosen to provide a structured overview of the intersection between both domains within the available time and resources. Due to time constraints, the search was limited to a single database, which may reduce coverage and limit the generalizability of the results. Since both areas contain a large body of literature, the study applies a focused search and screening process to ensure transparency and sufficient rigor. The steps of the study are detailed in this chapter.

3.1 Research Questions

Building on the methodology outlined above, the primary goal of this mapping study is to identify how software modernization and change management are connected and to determine whether they can reinforce each other. As the literature review revealed that existing examples are limited to specialized contexts such as ERP systems, cloud migration, and banking sector transformations, the research questions aim to explore a more generalized understanding of the relationship between these domains. In particular, the following research questions will be explored:

> *RQ 1: What are defining characteristics of change management and software modernization, respectively?*

> *RQ 1.1: To what extent are change management approaches applicable to software modernization?*

3.2 Search Strategy

To capture relevant research across both software modernization and change management domains, the following search strategy was used.

3.2.1 Search String

To include both areas as well as find research that combines them, the following search query was used:

> *("All Metadata": software moderni*ation change management)*

The query searches across all metadata fields to find studies that address both software modernization and change management. By searching all metadata rather than limiting to specific fields like title or abstract, the query captures a broader range of relevant literature, including studies that do not prominently feature the search terms in the title but nevertheless address the relationship between these domains. The wildcard in "moderni*ation" accommodates both British and American spelling variants (modernisation and modernization).

3.2.2 Data Extraction

The search query was executed on the IEEE Xplore database on November 11, 2025. IEEE Xplore is the largest digital library for research in electrical engineering, computing, and electronics. All papers returned by the search were downloaded and imported into Zotero, a reference management tool, for systematic organization and screening. An initial screening was conducted to assess the relevance of each paper to the research questions. During this screening process, each paper was categorized using tags to indicate its primary focus: *software modernization* (papers primarily addressing modernization strategies and technical aspects), *change management* (papers primarily addressing change management theories and practices), *both* (papers explicitly addressing the intersection of both domains), or *not relevant* (papers outside the scope of this study). This tagging system helped identify papers on the relationship between the two domains.

The initial search returned 48 papers in total.

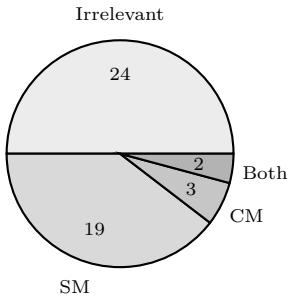


Figure 1: Distribution of screened papers (48 total) (SM = Software Modernization, CM = Change Management).

3.2.3 Data Synthesis

Following the initial screening, papers tagged as *software modernization*, *change management*, or *both* (24 papers in total) were selected for detailed analysis. Papers marked as *not relevant* were excluded from further consideration. The relevant papers were thoroughly reviewed to extract key characteristics, practices, and challenges related to both software modernization and change management. Nine key characteristics emerged as recurring themes across the literature: motivation, definition of change, phases, tools, core activities, risks, evaluation, stakeholder management, and upskilling. These characteristics were selected based on how frequently they appeared in the papers and their importance for understanding how each domain approaches transformation. For each characteristic, the papers were analyzed to compare how software modernization and change management differ in their approaches and to determine the extent to which change management practices are applicable to software modernization contexts.

4. RESULTS

This section summarizes the main findings of the systematic mapping study by comparing key characteristics of software modernization and change management. Table 1 provides a concise overview of how both domains align and where they differ.

Table 1: Summary of the comparison between software modernization and change management across key characteristics

Characteristic	Software Modernization	Change Management
Motivation	Technical sustainability (cost, risk of becoming obsolete, reliability, compliance).	Strategic adaptation (competitiveness, resilience, transformation readiness).
Definition of change	Technical system evolution (architecture, code, platform).	Organizational transition (people, culture, practices).
Stakeholders	Stakeholders support decisions; weak engagement risks conflict and poor estimates.	Stakeholders are central; participation increases acceptance and reduces resistance.
Phases	Structured around technical delivery and implementation.	Structured around readiness, participation, and anchoring change.
Core activities	Reverse engineering, migration, refactoring, reengineering.	Workshops, communication, training, feedback and alignment.
Tools	Automation and engineering support (e.g., transformation tooling).	Facilitation and adoption support (e.g., stakeholder mapping, readiness measures).
Upskilling	Learning new tools/technologies and modern development practices.	Building organizational capabilities to sustain new ways of working.
Risks	Technical failure, disruption, compatibility, cost and complexity.	Resistance, low buy-in, inertia, misalignment.
Evaluation	System and productivity metrics (quality, performance, maintainability).	Adoption and readiness indicators (understanding, adhesion, participation).

4.1 Motivation

Understanding what motivates software modernization and change management initiatives is essential because motivation determines which problems organizations prioritize, how they allocate scarce resources, and whether transformation efforts receive sustained commitment when challenges inevitably emerge.

Software modernization initiatives are typically driven by technical problems and operational sustainability concerns. Organizations modernize because legacy systems become increasingly difficult, time-consuming, and costly to maintain [3]. Houekpetodji et al. [7] identify the core motivation as companies' need to "deliver good quality products fast", requiring improvement in development practices to remain competitive. M'baya et al. [14] emphasize that as technology changes rapidly, enterprises must upgrade legacy systems to provide suitable modernization or risk progressive obsolescence. Koskinen et al. [10] document that modernization initiatives arise from multiple sources including legislative changes that mandate system updates, vendors ceasing maintenance support, obsolete technology platforms, and business strategy shifts requiring new system capabilities. These technical and regulatory motivations enable organizations to justify modernization investments through quantifiable cost reductions, improved system performance, reduced business continuity risks, and enhanced ability to meet evolving business requirements.

In change management, motivation emphasizes organizational competitiveness and strategic adaptation. Le Grand and Deneckere [12] frame motivation around Industry 4.0 and digital transformation, noting that companies must modernize their practices and tools to complete future programs successfully. The motivation extends beyond mere survival. Ruiz et al. [18] argue that enterprises need to evolve "to be adapted with changes in their context," emphasizing proactive adaptation to changing markets, customer expectations, and competitive landscapes. Change management treats transformation as essential for maintaining relevance, with motivation stemming from the need for organizational agility. These strategic motivations enable organizations to frame change initiatives as investments in competitive advantage, workforce capability, and organizational resilience rather than merely reactive responses to problems.

The key difference is that software modernization is motivated by technical problems that threaten operational continuity, because systems that no longer work effectively force action, while change management is motivated by strategic opportunities and competitive pressures that promise organizational advantage. Effective transformation requires balancing both motivations. Technical urgency can drive rapid system upgrades without securing organizational commitment, while strategic ambition can generate strong support for changes without matching technical capability, creating a mismatch between why the change is wanted and what can realistically be delivered.

4.2 Definition of Change

Software modernization and change management define "change" in fundamentally different ways, which shapes how each field approaches transformation. Software modernization sees change primarily as technical system evolution. Cánovas and Molina [4] describe it as "understanding and evolving existing software assets to maintain their business value", focusing on architectural transformation, code migration, and platform upgrades. Houekpetodji et al. [7] echo this technical view, describing modernization as improving "software development practices" to deliver quality products efficiently. In this perspective, change is measured through technical metrics like code complexity and system interoperability.

Change management, however, sees change as a much broader organizational transformation. Le Grand and Deneckere [12] define change as "the change from one state to another" but explicitly recognize its human impact: "change can be experienced as a trauma from employees" and must address practices, work conditions, tools, management, business, strategies. This definition includes organizational structures, employee behaviors, and cultural shifts, which are dimensions that software modernization literature rarely

discusses. Ruiz et al. [18] try to bridge this gap, noting that "enterprise systems need to evolve to be adapted with changes in their context" but the fundamental difference remains.

This difference creates two perspectives that complement each other but are incomplete on their own. Software modernization's technical focus provides precision for transforming systems but overlooks how people adapt, which may explain why modernization projects face resistance. Change management's broader view addresses organizational dynamics but lacks technical detail for complex system transformations. The intersection of these fields shows that successful transformation needs an integrated definition that acknowledges both technical system evolution and the human-organizational dimensions required for lasting adoption.

4.3 Stakeholders

Stakeholder management appears as a clear theme in a subset of the reviewed literature, with three papers directly addressing stakeholder dynamics in software modernization and change management contexts.

Koskinen et al. [10] examine multi-stakeholder environments in software modernization projects, identifying key stakeholder groups including developers, managers, client organizations, and suppliers. Their empirical study reveals that client organizations retain final decision-making authority while collaborating with IT managers and expert groups. The authors document specific consequences of poor stakeholder engagement, including decision-making based on intuition rather than evidence, inaccurate benefit estimates, and conflicts between suppliers and clients.

Le Grand [12] investigates stakeholder participation in change management processes, presenting quantitative evidence that stakeholders who participate in designing changes demonstrate 97% higher acceptance rates. Conversely, excluded stakeholders exhibit resistance and feelings of abandonment that impede progress. This work critiques traditional change management approaches for treating stakeholders as passive recipients rather than active participants in the change process.

Ionita et al. [8] highlight that involving stakeholders during modernization can trigger organizational change, such as introducing new roles, adjusting structures, and dealing with continuing reliance on legacy systems.

4.4 Phases

Both software modernization and change management organize their work through sequential phases, but how these phases define completion determines whether transformations succeed or fail. The critical difference is what each field considers "done", one focuses on technical functionality, the other on human readiness. This mismatch explains why technically successful modernizations often fail to achieve adoption.

To illustrate these differences, we examine phase models from the reviewed literature. Bakar et al. [3] identify six modernization phases namely planning, old requirements determination, new requirements determination, design and development, testing, and implementation, where "reverse engineering was highlighted as an important technique" for extracting business logic. Each phase builds linearly toward the next, treating technical deliverables as primary completion criteria. In contrast, Le Grand and Deneckere [12] propose three change management phases: Define, Experiment, and Anchor, where the Experiment phase serves as "the heart of the agile change model" through iterative workshops that cycle until a sufficient level of satisfaction is obtained for all stakeholders. Unlike modernization's step-by-step technical approach, change management deliberately loops through experimentation, measuring phase success through employee "information rate", "understanding rate", "adhesion rate" and "participation rate".

The phase models from the cited papers illustrate how phase completion criteria differ between the two domains. This difference creates projects that can technically finish all modernization phases while failing to achieve human adoption. Successful integration requires redefining what it means to complete a phase by acknowledging that neither technical functionality nor human readiness alone represents genuine transformation. Both dimensions must be satisfied before declaring any phase complete.

4.5 Core Activities

Understanding the core activities in software modernization and change management is crucial because these activities reveal how each discipline operationalizes its approach to transformation and determines what can realistically be achieved within organizational constraints.

In software modernization, core activities center on technical transformation processes. The work typically progresses through phases of planning, requirements extraction, design and development, testing, and implementation [3]. Central to this process is reverse engineering, which involves analyzing existing code to extract business logic and understand system architecture before transformation can begin [14]. These technical activities enable organizations to migrate legacy systems to modern platforms, improve system performance, and reduce technical debt through automated processes and architectural refactoring. The focus remains primarily on the system itself: understanding what exists, determining what should exist, and executing the technical changes needed to bridge that gap.

Conversely, change management activities emphasize human engagement and organizational adaptation. The process involves defining the change roadmap, experimenting through participatory workshops, and anchoring changes in organizational culture [12]. These workshops include sessions to address concerns, brainstorming to generate solutions, and experimentation to test new ways of working. They enable organizations to involve stakeholders in designing the change itself rather than imposing it. Complementing these workshops are control activities that measure adoption through indicators of information, understanding, adhesion, and participation, allowing change managers to adjust their approach based on stakeholder readiness.

The critical distinction is that software modernization activities enable technical transformation through systematic engineering processes, while change management activities enable organizational adoption through participatory engagement. Technical change without adoption leads to unused systems, while adoption without technical capability leads to frustrated expectations.

4.6 Tools

Tool selection reveals different philosophies about who

should drive transformation, whether humans should execute predefined changes or actively participate in shaping them. This difference matters because tools carry assumptions about who controls change, directly affecting whether people feel imposed upon or engaged, which determines resistance or acceptance.

Software modernization and change management use fundamentally different types of tools. Cánovas and Molina [4] describe Architecture-Driven Modernization (ADM) using "metamodeling and model transformation" to automate "basic activities in software change processes," enabling reverse engineering and code generation with minimal manual effort. These tools treat transformation as an engineering problem that can be solved through automation, minimizing human involvement. In contrast, Le Grand and Deneckere [12] propose tools like "cartography of change" and "participatory workshops" for mapping impacts and fostering dialogue. The ICAP barometer measures "information rate", "understanding rate", "adhesion rate" and "participation rate" rather than technical metrics. These tools work as facilitation aids, creating structured spaces for stakeholder involvement rather than automating tasks away.

Successful integration must balance automation's efficiency against participation's adoption benefits. Tools designed purely for technical transformation risk excluding the human engagement necessary for organizational acceptance, while tools focused only on participation may lack the efficiency needed for complex technical changes. Successful transformation requires combining tools, some that automate technical tasks and others that create meaningful opportunities for stakeholder participation in decisions affecting their work.

4.7 Upskilling

Learning new skills and upskilling is discussed in a limited subset of the reviewed literature, with two papers directly addressing the challenge of stakeholders learning new skills in software modernization and change management contexts.

In software modernization projects, learning new skills presents significant technical challenges. When modernizing software development practices, developers must acquire new competencies in tools and methodologies they have never used before [7]. The technology itself can create additional obstacles; obsolete technologies make it difficult to adapt both the technology and the development practices it encourages to modern standards.

Similarly, change management requires substantial skill development, but focuses on broader organizational capabilities rather than purely technical skills. Successful transformation programs require support for people to drive them, and skill alignment is a fundamental challenge [12].

With only two papers addressing learning and skill development, upskilling appears underexplored at the modernization–change management intersection. Future research could evaluate training strategies, define knowledge transfer metrics, and examine how upskilling affects adoption.

4.8 Risks

Understanding risk in software modernization and change management is critical because risk determines which transformation initiatives proceed, how resources are allocated, and what mitigation strategies organizations must deploy to protect both technical investments and organizational stability.

In software modernization, risks center on technical failure and system integrity. Organizations face the risk of business disruption if legacy systems fail during migration, loss of critical business logic embedded in old code, and compatibility issues with modern platforms [3]. Houekpetodji et al. [7] document how obsolete technologies create additional difficulties in adapting both the technology itself and the development practices it encourages to modern standards. The maintenance of legacy systems becomes increasingly difficult, time-consuming, and costly, yet organizations cannot simply discard these systems because they contain valuable business information accumulated over years [3]. M'baya et al. [14] note that redeveloping systems from scratch is considered risky and expensive, positioning modernization as the lesser of two risky alternatives. These technical risks enable organizations to quantify potential system downtime, estimate migration costs, and plan for technical contingencies through phased rollouts and parallel running of old and new systems.

Conversely, change management risks emphasize human resistance and organizational inertia. Le Grand and Deneckere [12] describe how change can be experienced as traumatic by employees accustomed to established routines, noting that resistance to change is a defining challenge that change management must address. When individuals feel uninvolved in change decisions, resistance intensifies. Their research shows that employees who lack opportunity to participate in planning the change are significantly more likely to reject transformation initiatives. Ruiz et al. [18] extend this perspective by noting that legacy systems involve "complex interrelationships of information, organisation culture and environment" meaning that even when technology is replaced, legacy work models persist and result in resistance. These human-centered risks enable organizations to anticipate adoption challenges, design participatory interventions, and measure readiness through indicators of understanding, adhesion, and participation.

The fundamental difference is that software modernization risks threaten system functionality and business continuity through technical failure, while change management risks threaten organizational effectiveness through resistance and cultural rejection. Successful transformation requires addressing both. Technical excellence without adoption produces systems nobody uses, while adoption without technical capability leads to chaos when promised changes cannot be delivered.

4.9 Evaluation

Understanding how software modernization and change management measure success is essential because metrics determine which initiatives receive continued investment, how progress is communicated to stakeholders, and whether transformation efforts are considered successful or failures requiring course correction.

In software modernization, evaluation focuses on technical system characteristics and operational efficiency. Organizations measure code quality metrics, system performance benchmarks, and architectural improvement indicators. Kamimura et al. [9] emphasize measuring business logic complexity to prioritize modernization efforts and estimate required work. Houekpetodji et al. [7] document

practical metrics including average development time per ticket, ticket lifetime from opening to closing, and the number of closed tickets per programmer, metrics that directly quantify developer productivity and system maintainability. M'baya et al. [14] propose quality metrics within an assessment framework to verify that claimed modernization benefits have been achieved. These technical metrics enable organizations to demonstrate return on investment through measurable improvements in system performance, reduced maintenance costs, faster feature delivery, and decreased technical debt.

In contrast, change management evaluation centers on human adoption and organizational readiness. Le Grand and Deneckere [12] introduce the ICAP barometer measuring four critical adoption indicators: information rate (whether people know about the change), understanding rate (whether people comprehend what will change), adhesion rate (whether people accept the change), and participation rate (whether people actively engage in implementing change). These metrics track the psychological and behavioral journey of stakeholders through transformation. The ICAP approach enables change managers to identify adoption bottlenecks. For instance, high information rates but low understanding rates signal communication problems, while high understanding but low adhesion indicates resistance requiring intervention. The emphasis remains on measuring human states rather than system states, recognizing that successful change manifests in altered behaviors, increased capability, and sustained organizational transformation.

The fundamental distinction is that software modernization metrics make technical progress visible and quantifiable through system measurements, while change management metrics make human adoption visible through behavioral and attitudinal indicators. Effective transformation requires both types of measurement. Technical metrics alone can produce systems that meet specifications but are not adopted, while adoption metrics alone can create support for changes that cannot be delivered or sustained technically.

5. DISCUSSION

The comparison shows a clear pattern across all the characteristics we examined: software modernization and change management approach transformation from different perspectives that complement each other rather than conflict. Software modernization focuses on technical system changes, treating transformation as an engineering problem that can be solved through automation, technical expertise, and systematic processes. Change management focuses on human and organizational adaptation, treating transformation as a social process that needs participation, dialogue, and behavioral change.

This difference shows up in every dimension. Software modernization defines change as technical system evolution, organizes phases around technical deliverables, uses automated tools, measures success through system metrics, and is motivated by technical problems. Change management defines change as organizational transformation, organizes phases around human readiness, uses participatory tools, measures success through adoption indicators, and is motivated by strategic opportunities. Neither perspective alone is sufficient for successful transformation, as one side risks building systems nobody will use, while the other risks creating expectations that cannot be met.

The key insight is that these two perspectives create frameworks that complement each other but are incomplete on their own. Software modernization provides precision for executing technical transformation but lacks ways to manage human resistance, build organizational capability, and ensure adoption. Change management provides ways to manage human dynamics but lacks technical detail for executing complex system transformations. Successful transformation requires bringing both frameworks together. Technical excellence must be combined with human engagement, system metrics with adoption indicators, automated efficiency with participatory legitimacy.

Despite the need for this integration, the study reveals that the two fields remain largely separate. Of the 48 papers examined, only 2 explicitly addressed both software modernization and change management together, while most papers focused on one domain or the other. Software modernization literature sometimes mentions stakeholder engagement and training but treats these as secondary concerns rather than core requirements. Change management literature acknowledges technical implementation but focuses mainly on human and organizational factors. The few examples of integration found remain limited to specialized contexts like ERP implementations and cloud migrations, which limits our understanding of how these frameworks can be combined more broadly.

This gap has real consequences. Organizations that treat modernization purely as a technical project often discover too late that resistance, skill gaps, and organizational inertia prevent adoption of technically successful systems. Organizations that emphasize change management may build enthusiasm and readiness but struggle when technical complexity exceeds what they can deliver. Effective transformation requires treating modernization as both a technical and organizational change process from the start, with integrated approaches that address both dimensions throughout all phases.

6. CONCLUSION AND FUTURE WORK

This study examines the intersection between software modernization and change management. It shows that modernization focuses on technical delivery, while change management focuses on adoption and organizational readiness.

6.1 Summary

This paper examined the relationship between software modernization and change management through a systematic mapping study of 48 papers from IEEE Xplore. The analysis identified nine key characteristics and revealed that the two domains approach transformation from complementary but separate perspectives. Software modernization focuses on technical system changes using automated tools and technical metrics, while change management focuses on human and organizational changes using participatory approaches and adoption metrics.

The comparison shows that the two perspectives address different aspects of transformation. Software modernization focuses on system changes while change management focuses on human adaptation. This separation suggests that focusing on only one perspective may be insufficient: technical changes without addressing human factors could face adop-

tion challenges, while organizational changes without technical capability could face implementation challenges.

6.2 Future Work

Several research directions emerge from this study. First, expanding the literature review to other databases would provide a more comprehensive understanding. Second, case studies comparing projects that integrate change management with those that do not could provide evidence of practical benefits. Third, applying established change management frameworks such as ADKAR [6] to software modernization contexts beyond ERP systems could demonstrate their broader applicability and effectiveness. Finally, developing an integrated framework combining modernization phases with change management practices could offer actionable guidance for practitioners.

7. REFERENCES

[1] W. K. G. Assunção, L. Marchezan, L. Arkoh, A. Egyed, and R. Ramler. Contemporary Software Modernization: Strategies, Driving Forces, and Research Opportunities. *ACM Trans. Softw. Eng. Methodol.*, 34(5):142:1–142:35, May 2025. doi:10.1145/3708527.

[2] W. K. G. Assunção, L. Marchezan, A. Egyed, and R. Ramler. Contemporary Software Modernization: Perspectives and Challenges to Deal with Legacy Systems, July 2024. doi:10.48550/arXiv.2407.04017.

[3] H. K. A. Bakar, R. Razali, and D. I. Jambari. Implementation Phases in Modernisation of Legacy Systems. In *2019 6th International Conference on Research and Innovation in Information Systems (ICRIIS)*, pages 1–6, Dec. 2019. doi:10.1109/ICRIIS48246.2019.9073628.

[4] J. Canovas and J. G. Molina. An Architecture-Driven Modernization Tool for Calculating Metrics. *IEEE Software*, 27(4):37–43, July 2010. doi:10.1109/MS.2010.61.

[5] H. E. Hayretci and F. B. Aydemir. A Multi Case Study on Legacy System Migration in the Banking Industry. In M. La Rosa, S. Sadiq, and E. Teniente, editors, *Advanced Information Systems Engineering*, pages 536–550, Cham, 2021. Springer International Publishing. doi:10.1007/978-3-030-79382-1_32.

[6] J. Hiatt. *ADKAR: A Model for Change in Business, Government, and Our Community*. Prosci, 2006.

[7] M. H. Houekpetodji, N. Anquetil, S. Ducasse, F. Djareddir, and J. Sudich. Report From The Trenches A Case Study In Modernizing Software Development Practices. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 515–524, Sept. 2021. doi:10.1109/ICSME52107.2021.00052.

[8] A. D. Ionita and S. A. Radulescu. Metamodeling for Assigning Specific Roles in the Migration to Service-Oriented Architecture. In *2012 Third International Conference on Emerging Intelligent Data and Web Technologies*, pages 293–297, Bucharest, Romania, Sept. 2012. doi:10.1109/EIDWT.2012.33.

[9] M. Kamimura, A. Matsuo, and Y. Maeda. Measuring Business Logic Complexity in Software Systems. In *2015 Asia-Pacific Software Engineering Conference (APSEC)*, pages 370–376, Dec. 2015. doi:10.1109/APSEC.2015.26.

[10] J. Koskinen, H. Lintinen, J. Ahonen, T. Tilus, and H. Sivula. Empirical study of industrial decision making for software modernizations. In *2005 International Symposium on Empirical Software Engineering, 2005.*, pages 10 pp.–, Nov. 2005. doi:10.1109/ISESE.2005.1541832.

[11] T. Lauer. *Change Management: Fundamentals and Success Factors*. Springer, Berlin, Heidelberg, 2021. doi:10.1007/978-3-662-62187-5.

[12] T. Le Grand and R. Deneckere. COOC: An Agile Change Management Method. In *2019 IEEE 21st Conference on Business Informatics (CBI)*, volume 02, pages 28–37, July 2019. doi:10.1109/CBI.2019.10093.

[13] J. Leppänen. Change management in ERP system transition : A case study of procurement department at Meyer Turku Shipyard. http://www.theseus.fi/handle/10024/876256, 2024.

[14] A. M'baya, J. Laval, and N. Moalla. An assessment conceptual framework for the modernization of legacy systems. In *2017 11th International Conference on Software, Knowledge, Information Management and Applications (SKIMA)*, pages 1–11, Dec. 2017. doi:10.1109/SKIMA.2017.8294120.

[15] C. S. V. Murthy. *Change Management*. Himalaya Publishing House Pvt. Limited, 2007.

[16] S. Ponnusamy and D. Eswararaj. Navigating the Modernization of Legacy Applications and Data: Effective Strategies and Best Practices. *Asian Journal of Research in Computer Science*, 16(4):239–256, Nov. 2023. doi:10.9734/ajrcos/2023/v16i4386.

[17] A. Prakash and A. Pinto. Navigating the Human Element: Change Management in Digital Transformation. *Vidyavardhaka Journal of Management*, pages 01–15, July 2025. doi:10.18311/vjm.v2i1.2025.34.

[18] M. Ruiz, S. Espana, O. Pastor, and A. Gonzalez. Supporting organisational evolution by means of model-driven reengineering frameworks. In *IEEE 7th International Conference on Research Challenges in Information Science (RCIS)*, pages 1–10, Paris, France, May 2013. doi:10.1109/RCIS.2013.6577682.

[19] C. C. Tan. The Theory and Practice of Change Management. *Asian Business & Management*, 5(1):153–155, Mar. 2006. doi:10.1057/palgrave.abm.9200152.

[20] M.-F. Wendland, M. Kranz, C. Hein, T. Ritter, and A. García Flaquer. Model-based testing in legacy software modernization: An experience report. In *Proceedings of the 2013 International Workshop on Joining AcadeMiA and Industry Contributions to Testing Automation*, JAMAICA 2013, pages 35–40, New York, NY, USA, July 2013. doi:10.1145/2489280.2489291.

[21] Z. B. Yusof. Managing the Human Factor in Cloud Migration: Strategies for Effective Change Management and Employee Training. *Transactions on Machine Learning, Artificial Intelligence, and Advanced Intelligent Systems*, 14(10):1–13, Oct. 2024.

Model Cards: the Good, the Bad and the Ugly

Alexander Kulibaba
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
alexander.kulibaba@rwth-aachen.de

Philipp Lentzen
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
philipp.lentzen@rwth-aachen.de

ABSTRACT

Machine learning (ML) transparency has become a critical requirement in high-stakes domains, with model cards emerging as the widely adopted standard for standardizing model reporting. Despite their adoption in industrial repositories, recent empirical evidence suggests a significant disparity between the framework's aspirations and its practical utility.

This study presents a Systematic Literature Review (SLR) of 26 peer-reviewed publications, combining empirical findings into a review of *the good* (standardization benefits), *the bad* (operational friction), and *the ugly* (malpractice and ethics washing). Additionally, we evaluate the landscape of technical tools designed to support model card creation.

The results indicate that although model cards have successfully established a common vocabulary, they frequently fail to communicate actionable risks and limitations due to liability concerns and a lack of verification. Regarding tooling, we find that while tools such as RiskRAG and DocML demonstrate effectiveness, the broader landscape prioritizes the speed of content generation over verifying data.

We conclude that for model cards to function as effective safety instruments rather than marketing artifacts, the field must pivot from generative assistance to automated content verification and enforced documentation standards.

Categories and Subject Descriptors

D.2.0 [**Software Engineering**]: General—*Documentation*; I.2.6 [**Artificial Intelligence**]: Learning—*Machine learning, transparency*; K.4.2 [**Computers and Society**]: Social Issues—*Social ethics*

Keywords

Model Cards, Machine Learning Documentation, Systematic Literature Review, AI Transparency, ML Documentation Tools

SWC Seminar 2025/26 RWTH Aachen University, Germany.

1. INTRODUCTION

Model cards are specialized documentation artifacts designed to encourage transparent reporting of model characteristics, similar to nutritional labels for consumer goods [1, 2]. Typically presented as short documents (see example model cards by Google DeepMind [3]), they serve the primary goal of clarifying intended use cases and standardizing ethical reporting for AI systems [1]. To achieve this, the framework requires a structured organization consisting of nine essential sections, as shown in Figure 1 [1]. These artifacts primarily use a descriptive approach to communicate important facts, such as the model's architecture, the training data used, and its potential limitations [4, 5]. A central feature of model cards is the inclusion of benchmarked evaluation results across relevant cultural, demographic, and phenotypic groups, which helps diverse stakeholders assess fairness and suitability [1, 6].

The creation of model cards stemmed from the rapid adoption and inherent complexity of ML models, particularly those deployed in critical domains such as healthcare and law enforcement [1]. Historically, ML adoption outpaced existing documentation standards. As a result, models are often distributed without the essential information users need to understand their systematic impacts [2, 5]. This lack of transparency led to significant failures, including instances where opaque software predicted a higher risk of future crime for African-Americans compared to White individuals [5, 7]. Consequently, Mitchell et al. [1] proposed model cards to address these gaps and minimize the risks of deploying models in contexts for which they are not suited.

In the current landscape, model cards have matured from a proposed solution to a widely adopted standard for model reporting [8]. Their integration into central repositories, most notably Hugging Face with the *Model Card Writing Tool* [9], has cemented their role in the ecosystem, with major companies such as Google and OpenAI routinely attaching them to model releases [10, 8]. Furthermore, this adoption is increasingly driven by a shifting regulatory environment rather than voluntary consensus alone [11]. Legislative frameworks, such as the EU AI Act [12], are consolidating documentation requirements into formal laws, effectively transitioning model cards from optional best practices to strict compliance necessities for high-risk systems [13, 14].

2. RELATED WORK

The related work for this study can be categorized into two main categories: empirical analyses of model card adoption

Model Card

- **Model Details**
  Information about the model, including developer, version, type, publishing date, licensing, citation, contact information, and summarizes information about the training approach.
- **Intended Use**
  Use cases and users for which the model was designed, along with out-of-scope or inappropriate uses.
- **Factors**
  Relevant demographic, environmental, or technical factors considered during model development and evaluation.
- **Metrics**
  Performance measures, decision thresholds, and variability analyses chosen to reflect potential real-world impacts.
- **Evaluation Data**
  Overview of the datasets used for evaluation, including motivation for their selection and major preprocessing steps.
- **Training Data**
  High-level description of training data sources and key characteristics, including distributions over important factors where available. May not be possible to provide.
- **Qualitative Analyses**
  Overall performance results and intersectional analyses across relevant groups.
- **Ethical Considerations**
- **Caveats and Recommendations**

Figure 1: Summary of model card sections and suggested prompts for each. [1]

and technical proposals for documentation assistance. Regarding empirical adoption, most existing research focuses on specific components of the gap, though a few papers provide a more comprehensive overview. A notable example is the systematic study by Liang et al. [8], which offers an overview of all model cards on Hugging Face with a primary focus on their completeness. Another similar study was later conducted that includes a wider range of models while shifting the analytical focus toward risk [7]. However, there is no paper which gives a broad overview of problems with model card adoption across all dimensions.

In the context of tooling, there is currently a lack of peer-reviewed evaluations covering all established tools, which represents a clear gap in the existing literature. While formal evaluations are missing, curated lists of tools and their specific use cases can be found in Hugging Face blog articles, such as their model card landscape analysis [15]. Our research aims to directly address and close this gap.

3. PROBLEM STATEMENT & RESEARCH QUESTIONS

As described in the previous sections, model cards have matured into an industry standard [8]. However, despite the widespread adoption and regulatory pressure [8], empirical findings consistently reveal a substantial gap between the ambition of model cards and their practical utility [2, 14]. The quality of current documentation often falls short of established practices for traditional software [2] or safety-critical manufacturing standards [16]. Especially essential sections such as ethics or security considerations are heavily dependent on the execution of the individual creator [16, 5, 13]. While questions have been raised regarding whether model cards achieve their intended goals in practice [7], there is currently no comprehensive review of these empirical insights.

The large amount of evaluations of model cards and proposed tools for solving their problems make it difficult to obtain a clear overview of empirically evaluated proposals or case studies. To date, there has been no consensus on the collective empirical evidence regarding the specific pitfalls and gaps in model card adoption and a classification of the range of tools offered to assist their creation. The main purpose of this study is to bridge this gap by conducting a SLR that synthesizes disparate empirical findings into a consolidated view of *the good* (benefits), *the bad* (operational friction), and *the ugly* (malpractice). In addition, this study investigates the technological landscape to determine what supporting tools exist and how they address the identified shortcomings. We evaluate whether these tools explicitly solve the adoption gaps or if they lack sufficient empirical validation. This consolidated overview contributes new knowledge by linking identified adoption failures directly to the capabilities of existing tooling, or the lack thereof.

This study makes use of a scientific literature review, aimed at answering the following two research questions

- **RQ1:** What are strengths and weaknesses of model cards based on empirical studies?
- **RQ2:** What supporting tools for the creation of model cards have proven effective, according to the literature?

The remainder of this paper is organized as follows: Section 4 outlines the methodology of the SLR, detailing the search strings, the databases consulted, and the specific inclusion and exclusion criteria used to filter the corpus. Section 5 summarizes the findings on the empirical utility of model cards, categorizing them into strengths (*the good*), operational limitations (*the bad*), and systemic risks (*the ugly*). It also reviews the technological landscape to evaluate the tools available for generating model cards. Finally, Section 6 interprets these findings by identifying specific gaps in current tools that fail to address the weaknesses highlighted in the empirical data. Section 7 concludes the paper by discussing future work.

4. METHODOLOGY

To answer the research questions (RQs) defined in this study, a SLR was conducted in accordance with the established guidelines proposed by Kitchenham [17]. This approach provides a structured, transparent, and reproducible

process for identifying, selecting, and synthesizing relevant literature on model card practices and tools. The review process consisted of four main phases: (i) formulation of a search strategy, (ii) creation of inclusion and exclusion criteria, (iii) selection of studies based on these criteria, and (iv) extraction and synthesis of data.

4.1 Search Strategy

The search strategy was designed to retrieve relevant publications that explicitly discuss model cards in the context of ML documentation. To achieve this, a search string was constructed that targets publications at the intersection of the ML domain and the model card artifact.

4.1.1 Search String

To ensure high relevance, a search string was constructed that combines terms related to the domain with terms specific to the target artifact.

- **Domain:** Terms related to Machine Learning (e.g., "machine learning", "ML", "AI").
- **Artifact:** Terms related to model cards (e.g., "model card").

This formulation intentionally restricts the search space to publications that engage directly with model cards rather than general documentation approaches or other usages of model cards in other domains. The final search string used across all databases is as follows:

```
("machine learning" OR "ML"
OR
"artificial intelligence" OR "AI")
AND
( "model card" OR "model cards" )
```

4.1.2 Databases

To ensure broad coverage of relevant literature in computer science and software engineering, the search was executed across the following three scientific databases:

- ACM Digital Library
- IEEE Xplore Digital Library
- Scopus

The search was restricted to the abstract field to avoid retrieving papers that merely reference model cards within the full text, and limited to papers published after 2019. This restriction ensured that the results emphasized publications in which model cards represent a primary or substantial topic of interest.

4.2 Inclusion and Exclusion Criteria

To ensure the quality and relevance of the selected studies, we established a set of strict criteria prior to the screening process. These criteria were applied consistently to determine whether a publication directly contributed to answering our research questions.

To be included in the final corpus, a study was required to meet all of the following conditions:

IC1 The central theme and primary focus of the paper must be model cards.

IC2 The paper must present one of the following:

- Empirical evidence regarding the utility, effectiveness, or quality of model cards.
- A proposal and/or evaluation of a concrete technical tool or method for the generation, management, or assessment of model cards.

IC3 The paper must be a peer-reviewed academic publication.

After applying the inclusion criteria, we filtered the remaining publications using the following exclusion criteria to remove studies that did not meet our quality or accessibility standards:

EC1 The paper is peer reviewed in a conference which lacks credibility.

EC2 The paper is not accessible for us to review.

4.3 Study Selection

The study selection process followed the search strategy previously outlined and applied the predefined inclusion and exclusion criteria to ensure consistency throughout the review.

The initial search on abstracts across all databases yielded a total of 126 publications. After removing duplicates, 80 unique records remained for screening. Both authors independently reviewed the records by applying the criteria to the title, abstract, and full text of each publication. When relevance could not be determined from the abstract, the full text was examined to reach an informed decision. Disagreements between reviewers were discussed until a consensus was reached. Following this process, 26 studies met all criteria and were incorporated into the final analysis.

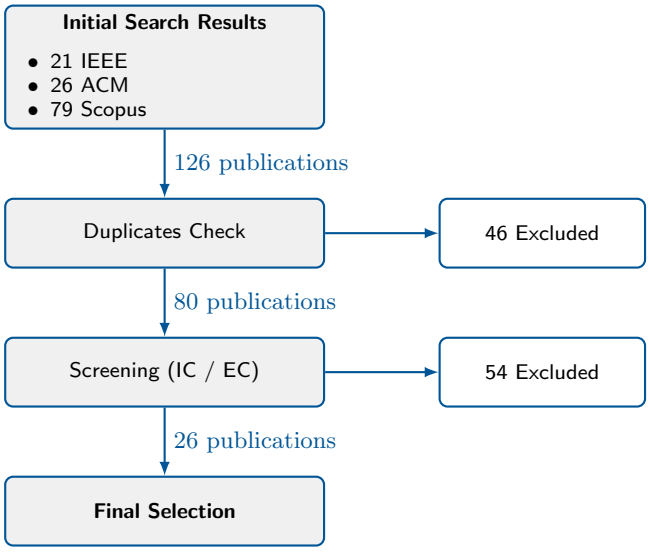


Figure 2: Systematic literature review selection process

4.4 Categorization of Findings

Following the study selection, empirical findings were extracted and coded thematically. To answer RQ1, each finding was assigned to one or more of the three categories: findings describing demonstrable benefits of model cards (e.g., standardization or adoption) were coded as *the good*; findings identifying practical limitations, frictions, or unmet goals were coded as *the bad*; and findings indicating deliberate omission, performative compliance, or misleading documentation practices were coded as *the ugly*. Ambiguous findings spanning multiple categories were discussed qualitatively rather than forcefully assigned into a single category.

5. RESULTS

The findings reported in this section are derived from the thematic coding of the 26 studies described in Section 4, with empirical observations grouped according to their dominant implications for the utility, limitations, and misuse of model cards. First, we analyze the empirical evidence regarding the utility and limitations of model cards in real-world scenarios, categorizing findings into strengths and significant adoption gaps. Subsequently, we examine the technological landscape to evaluate the tools and frameworks that have been proposed to address these challenges.

5.1 RQ1: The Good, the Bad and the Ugly

The synthesis of empirical evidence reveals a clear contrast in the current status of model cards. While the framework has achieved its primary goal of standardization, the literature highlights a divergence between this theoretical utility and practical implementation. This section first outlines the successful adoption of the framework (*the good*), before analyzing the operational shortcomings (*the bad*) and instances of systemic misuse (*the ugly*) that have emerged alongside it.

5.1.1 The Good: Standardization and Adoption

The literature consistently identifies a primary success characteristic of *the good*: the framework has established a standardized vocabulary for AI transparency and achieved widespread adoption across industry platforms such as Hugging Face [8, 10]. Model cards were introduced to serve as the primary resource for understanding reusable software components [1], and empirical analysis confirms that this terminology has been formally integrated into infrastructure in both academia and industry [10].

This standardization has provided tangible benefits for users. For instance, "Intended Use" sections are present in almost all analyzed model cards, aiding users in validity assessments [8].

Beyond these successes, however, empirical studies predominantly report operational shortcomings (*the bad*) and instances of systemic misuse (*the ugly*). Crucially, these negative outcomes are often mixed and derive thematically from "the good" adoption of the framework. For example, the broad accessibility intended by the standard can lead to "ugly" implications regarding reproducibility. Therefore, rather than separating findings into binary categories, we analyze these disparities across three key dimensions: the systematic lack of documentation completeness, the obfuscation of limitations and risks, and the persistent tension between technical reproducibility and broad accessibility.

5.1.2 The Lack of Completeness

Despite the broad adoption described above, empirical analysis reveals a substantial gap between the presence of a model card and the depth of actual reporting [2]. Documentation quality remains highly variable and often superficial. One analysis indicated that 18 out of 22 information types proposed in the original framework appear in less than half of existing model cards [8]. Consequently, the utility of individual cards remains heavily dependent on the integrity and diligence of the creators. This is, for example, reflected in the length and granularity of the model card: depending on the popularity of the model, the word count can range from 191 words at the population level, to an average of 521 words inside the top 100 models on Hugging Face [8].

However, even if sections are filled out, their correctness is unclear. The *Environmental Impact* section, for example, is only filled out in 2% of the cases, and about 84.8% of those sections are automatically generated by other AI tools [8]. A survey of NLP literature revealed that explicit reporting of energy consumption and greenhouse gas emissions remains negligible, prompting proposals for specialized Climate Performance Model Cards [18].

The most significant disparity between the proposal's aspiration and current practice, however, lies in ethical reporting. Although model cards were designed to encourage forward-looking fairness analysis and accountability [1, 19], empirical studies portray a major gap in execution. In a study of over 32,111 model cards, risks and biases were documented in less than 20% of cases [8]. A more recent analysis of 450,000 models found that only 14% reported risks, with 96% of that content being copied rather than original [7]. This evidence suggests that developers often selectively document ethical issues to avoid liability, framing remarks to prevent data misuse rather than transparently addressing inherent model biases [20, 5].

In addition to these reporting gaps, the integrity of documentation is further compromised by model drift. Since ML models are improved in an iterative process, static documentation may no longer accurately reflect the current systems, undermining the goal of transparency [2]. Later iterations of the model card, such as the method card, include a suggested *Robustness* section to prompt developers on detecting and mitigating data and model drifts, but real-world adoption remains limited [21, 4].

5.1.3 Unclear Limitations and Risk Factors

A core objective of the framework is to clarify a model's validity for specific use cases and, crucially, to minimize misuse in inappropriate contexts [1]. While the *Intended Use* is frequently documented, the documentation of limitations and *out-of-scope* use cases is frequently neglected. Surveys indicate that only 32% of model documentation explicitly addresses out-of-scope scenarios [8]. Furthermore, sections which explain the risks of a model are often criticized for being overly vague or generic, limiting their practical application in decision-making processes [7].

Findings in an interview-based study by Nunes et al. [20] also suggest that developers tend to record choices which they considered ethical, while omitting reflections on unethical design decisions. For example, one participant verbally noted a potentially unethical action regarding the inclusion of personal data, but intentionally omitted it from the model card [20]. This suggests that developers are less inclined or

incentivized to document their limitations thoroughly [20]. This avoidance of accountability is further evidenced by linguistic patterns. During the interview developers frequently employ passive voice to obscure their agency, distancing themselves from responsibility and relying on human mediation of algorithmic performance [20, 22].

5.1.4 Accessibility vs. Reproducibility

Model cards aim to serve a diverse range of stakeholders, from data scientists to non-expert consumers and policymakers. However, the standard static model card remains tailored primarily to individuals with ML or Natural Language Processing (NLP) expertise [23]. Research indicates that for non-expert analysts such as patients and healthcare providers, technical model cards are often minimally comprehensible [24, 25]. In radiology, there are improved model cards tailored specifically for ML models and datasets [26], but the inclusion of full technical detail can decrease interpretability for non-experts due to information overload. An experiment involving non-expert participants evaluating a real model card found that the original, full version was perceived as significantly less understandable compared to shorter versions that excluded technical content [24].

While improving accessibility is critical, it creates a tension with the objective of reproducibility. To ensure transparency, model cards aim to improve reproducibility by documenting architecture and training procedures [1]. In practice, however, reproducibility is often hindered by proprietary constraints. To protect intellectual property, creators often leave out details regarding training data [4]. Even when proprietary concerns are absent, descriptions are often vague. Users frequently turn to discussion forums to clarify basic technical details, indicating that the published cards often fail to stand alone as sufficient technical documentation [10]. These issues highlight a persistent design challenge: balancing the technical depth required for reproducibility with the simplicity needed for broad accessibility.

5.2 RQ2: How Tools are closing the Gaps

The adoption of model cards is frequently hindered by the significant effort required to produce them, a lack of standardization, and the tendency for resulting documentation to be incomplete or superficial. To address these challenges, the literature proposes various tools designed to streamline creation, enhance documentation quality, and ensure accountability. These tools generally fall into three categories: those enhancing risk reporting, those improving accessibility through layering, and those streamlining creation and management.

5.2.1 Enhancing Risk Reporting

A primary shortcoming identified in model card implementation is the vagueness of risk analysis and the difficulty non-experts face in interpreting technical documentation.

To address the lack of specific, actionable risk reporting, the most empirically validated tool in the literature is RiskRAG. RiskRAG is a retrieval-augmented generation (RAG) system that automatically generates model-specific risk reports by matching a given model description to a database of unique risks from 2,672 and 649 real-world AI incidents.

The effectiveness of RiskRAG was evaluated through multiple user studies. In a preliminary study with 50 AI developers, 74% of participants preferred the RiskRAG report over baseline model cards. A subsequent study involving 115 stakeholders, including developers and designers, demonstrated that these reports encouraged more deliberate decision-making. Quantitatively, RiskRAG significantly outperformed baseline documentation. On a 5-point Likert scale, it scored 4.46 for risk coverage compared to the baseline's 3.40 ($p < 0.001$). Furthermore, it improved the clarity of mitigation strategies, scoring 4.08 against the baseline's 3.08 ($p < 0.001$). However, the evaluation also highlighted drawbacks: the tool's performance in generating mitigation strategies was lower than its risk identification capabilities, and some users experienced a higher cognitive load when adapting to RiskRAG's structured matrix format compared to traditional text [7]. To address broader societal risks, other frameworks like the ESG Model Card have been proposed to extend the standard format by explicitly tracking environmental metrics and security vulnerabilities throughout the model lifecycle [27].

Category	Tools and Frameworks
Enhancing Risk Reporting	RiskRAG [7] ESG Model Card [27] AHA! [7] ExploreGen [7] FarSight [7]
Improving Accessibility	Interactive Model Cards [23, 24] Model Card Authoring Toolkit [6] Robustness Gym Reports [23]
Streamlining Creation and Management	DocML [2] CardGen [28] Model Card Toolkit [19] Themisto [2] WrangleDOC [2] Laminator [29] Model Card Report Ontology [30] Method Cards [4]

Table 1: Categorization of tools and frameworks for model card enhancement and automation.

5.2.2 Improving Accessibility through Layering

Targeting the accessibility gap, Interactive Model Cards (IMCs) extend static model cards with interactive visualizations, such as performance dashboards and robustness reports, to help non-expert stakeholders explore model behavior along relevant dimensions (Figure 3). IMCs allow users to interactively select data slices, compare performance metrics across subgroups, and inspect robustness test results without requiring direct access to the underlying data.

Evaluations indicate that 18 out of 20 non-expert analysts found IMC prototypes easier to understand than standard cards, noting improved conceptual understanding of the model [23]. Despite this preference, empirical results showed that viewing IMCs did not significantly alter a user's intention to use an AI system. Furthermore, researchers found that including deep technical content, even in an interactive format, created a barrier to interpretability for non-experts [24].

Other tools in this category address accessibility by structuring how information is authored or explored. These tools

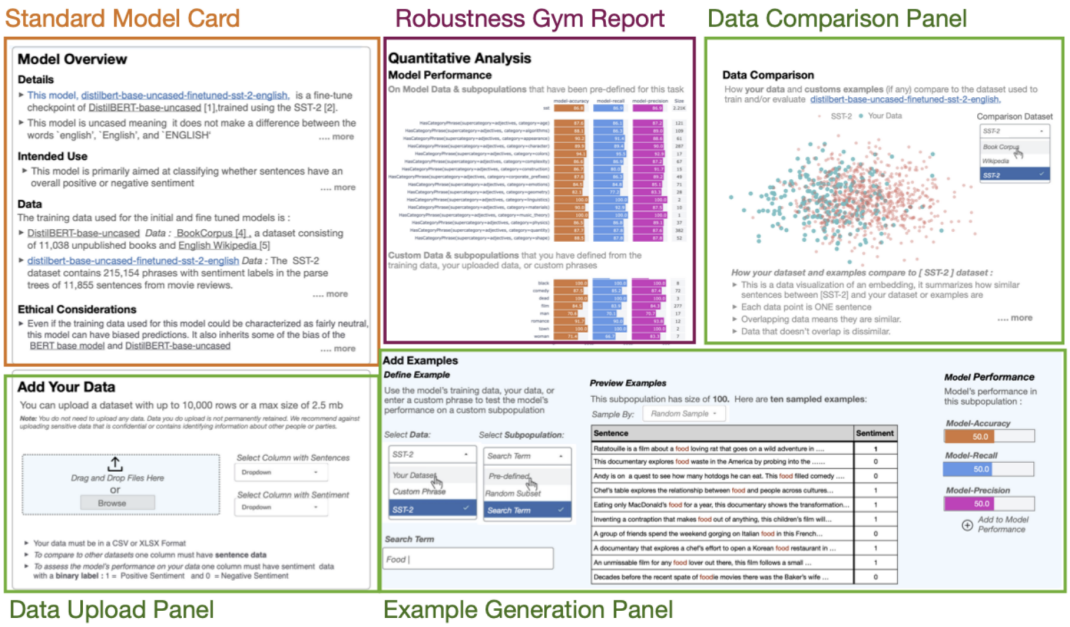


Figure 3: Concept of an Interactive Model Card including the Data Comparison Panel and Robustness Gym Report, as proposed by Crisan et al. [23].

primarily support specific sections of the model card rather than end-to-end documentation. The Model Card Authoring Toolkit supports collaborative authoring of model cards. It provides structured prompts and interfaces, guiding multiple stakeholders through predefined documentation fields, such as intended use, evaluation data, and ethical considerations [6]. Robustness Gym Reports integrate test suites and data slices into the documentation process, enabling users to systematically explore how model performance metrics change across predefined dimensions.

Risk-assessment tools like AHA!, ExploreGen, and FarSight [7] utilize large language models (LLMs) to help developers anticipate harms and regulatory risks by generating speculative risk scenarios, compliance prompts, or harm-related questions based on a model description. While these tools can broaden risk awareness early in development, they lack the extensive empirical validation of RiskRAG [7], which adapted their speculative methodologies into a more rigorous and structured evaluation framework.

5.2.3 *Streamlining Creation and Management*

A second major barrier to adoption is the disconnect between the documentation process and the research process where the ML model capabilities are measured. This often leads to inconsistencies or dismissal of thorough documentation, which is why many tools suggest an integration of documentation into the coding environment.

The most effective tool for addressing workflow integration is DocML, which embeds model card prompts into computational notebooks, nudging data scientists to document important design decisions and ethical considerations as they implement and evaluate their models. In a lab study involving 16 data scientists, DocML successfully "nudged" users toward better practices. Participants demonstrated more deliberate consideration of ethical concerns compared to a control group. The tool also effectively ensured traceability, with nearly all participants in the experimental group maintaining consistency between the code and the resulting model card. While successful in a controlled setting, the study noted a learning curve for users, and the long-term impact on continuous documentation evolution remains unverified [2].

For fully automated generation, CardGen utilizes LLMs within a RAG framework to synthesize model cards from research papers and repositories [28]. Evaluations using OpenAI's GPT-3.5 showed that CardGen achieved higher scores than human-generated cards regarding completeness, objectivity, and understandability [28]. However, the literature warns that, because the tool synthesizes existing text, it risks replicating established problems, such as vague risk descriptions, and is susceptible to the hallucinations inherent in generative AI [28, 7].

Despite high conceptual visibility, the Model Card Toolkit (MCT) from Google, which provides JSON schemas and helper functions to generate model cards from structured configuration data has seen low adoption [19]. While it standardizes data via the schemas, it fails to assist with the qualitative sections of the card that require manual input, such as ethical considerations [2]. Empirical studies of GitHub practices identified only a single repository utilizing the MCT, suggesting that its focus on automated metrics does not sufficiently reduce the burden of documentation. [2]. Other tools in this category include Themisto and WrangleDOC. These tools focus on documentation in earlier stages of the ML pipeline, code cells, and data wrangling steps, respectively, with the aim of preserving traceability from low-level operations to the final model card [2]. However, these tools still rely on the honesty of the reporter. To ensure the integrity of these claims, Duddu et al. [29] introduced Laminator, which leverages hardware-assisted attestations to create verifiable "inference cards" that cryp-

tographically prove a model's properties and link outputs directly to their underlying training data.

Beyond creation tools, Amith et al. [30] proposed the Model Card Report Ontology (MCRO) to standardize the underlying representation of these documents, making them machine-readable and interoperable. This ontology-based approach transforms static reports into semantic artifacts, enabling automated reasoning and checking adherence to FAIR [31] data principles. Oftentimes ML models also get deployed on systems which are limited in memory and compute, especially in the domain of systems engineering in the aerospace sector. To account for these limitations, there are proposals to extend machine readable model card schemas with additional data points to allow for fast evaluation of sensor fusion models against strict hardware constraints [32].

Distinct from the tools above, Method Cards are proposed as a prescriptive documentation standard that provides the *recipe* rather than the description of the *meal*. Unlike model cards, which describe a specific trained model, Method Cards guide ML engineers in the proper use of underlying algorithms and techniques, enhancing reproducibility and preventing misuse. Although they are conceptually significant for filling the transparency gap before model deployment, Method Cards currently lack empirical data regarding their adoption or effectiveness in industrial settings [4].

6. DISCUSSION

The results of this SLR reveal a distinct discrepancy in the state of model cards. On one hand, *the good* is evident: model cards have successfully transitioned from a theoretical proposal to an industry standard, achieving high adoption rates on platforms like Hugging Face and becoming a prerequisite for perceived model legitimacy. Importantly, this success should not be understated, as the widespread adoption of model cards enables many of the shortcomings identified in this review to be empirically observed in the first place. However, on the other hand, *the bad* and *the ugly* reveal that standardization and adoption alone do not translate into effective transparency, as documentation quality remains inconsistent and frequently superficial. This section combines these findings to interpret why current practices fail to meet the original aspirations of the framework and how current tooling impacts this trajectory.

6.1 Superficial Compliance

The literature confirms that while the structure of model cards has been standardized, the content remains inconsistent. A recurring theme across empirical studies is the focus on procedural completion. In this approach, developers produce documentation to satisfy a requirement or checklist without engaging in the deep reflection that the artifact is meant to capture. This is most evident in the documentation of intended uses versus limitations. Developers are incentivized to populate sections that advertise capabilities (*Intended Use*), yet they consistently neglect sections that might suggest liability or reduce adoption (*Out-of-Scope Use* and *Risks*). The empirical evidence showing that risk sections are often generic or copied suggests that for many developers, the model card has become a marketing brochure rather than a safety manual. This indicates that voluntary standardization is insufficient. Without external verification or regulatory audits such as those emerging in the EU AI Act [12], the presence of a model card is a poor proxy for model safety.

6.2 Importance of Validation

Our review of support tools (RQ2) highlights the misalignment between the problems identified in RQ1 and the solutions currently offered. Most existing tools focus on reducing friction by making it easier to write a model card, rather than ensuring the card is complete. Tools like CardGen, which use LLMs to generate documentation, are a double-edged sword. While they solve *the bad* (time-consuming creation), they exacerbate *the ugly* (hallucinations and generic fluff). If an LLM generates a risk assessment based on a generic template, it reinforces the superficiality of current reporting. The literature suggests that the field does not need tools that write faster, it needs tools that validate harder [7].

A promising finding is the success of "nudging" tools such as DocML and RiskRAG. Unlike the Google Model Card Toolkit (MCT), which failed due to its rigid, manual JSON schema, DocML succeeds by integrating into the data scientist's native workflow. Through this integration into the notebook it prompts reflection during development rather than after.

Similarly, RiskRAG is effective because it reduces the cognitive load of remembering specific risks by retrieving relevant examples. This suggests that future tooling should move away from passive forms to active, context-aware assistants that challenge developers to be specific.

6.3 Separation of Concerns

The literature exposes a fundamental design flaw in the current model card paradigm: the single-document fallacy. The original vision for model cards was to act as a version of nutritional labels, which only provide a simple overview. However, current implementations oscillate between being too technical for non-technical people (e.g., patients, policymakers) and too vague for experts.

IMCs are attempting to bridge this gap, but as findings show, technical detail can paradoxically lower non-expert understanding. This suggests that the industry may need to rethink the standard. A model card may serve as the high-level summary for downstream users, while a distinct Method Card or technical specification serves the reproducibility needs of engineers. Attempting to force both into a single markdown file on a repository has resulted in a document that serves neither audience effectively.

6.4 Ethics Washing and the Limits of Documentation

Perhaps the most concerning finding is the systematic ethics washing identified in *the ugly* category. The fact that bias reports are frequently omitted or framed to avoid liability indicates that documentation is a downstream symptom of an upstream culture problem. Tools cannot solve this issue alone.

The literature indicates that unless documentation is tied to accountability mechanisms, where a false or misleading model card carries reputational or legal consequences, the quality of ethical reporting will remain stagnant. Documentation must shift from being a static description of the model to a dynamic record of the process.

6.5 Threats to Validity

We acknowledge potential threats to the validity of this SLR.

- **Search Strategy:** Our search was restricted to three major databases (ACM, IEEE, Scopus). While these cover the majority of computer science literature, relevant pre-prints on arXiv (common in the fast-moving ML field) may have been excluded if they were not peer-reviewed.
- **Industry Research:** Successful documentation practices or tools which may be used internally in corporate environments do not always result in a academic papers accessible for the public, leaving a gap in research that could improve model cards.
- **Tool Availability:** Several tools discussed (e.g., LAMINATOR, Method Cards) are theoretical proposals or prototypes not yet widely available for industrial testing, limiting our ability to assess their long-term impact.

7. CONCLUSION

This SLR evaluated the current state of model card adoption and the technological landscape that supports its creation.

7.1 Summary

In regard to RQ1, our analysis reveals that the model card framework has successfully established a standardized vocabulary for AI documentation (*the good*). However, the practical utility of model cards reveals persistent shortcomings. While the section titled *Intended Use* is consistently addressed, crucial elements related to *Limitations*, *Risks*, and *Bias* are frequently underreported or formulated in a vague manner (*the bad*). Beyond these omissions, the review also identifies underlying issues in which the absence of rigorous validation mechanisms permits model cards to function as symbolic compliance artifacts rather than instruments that genuinely enhance transparency or safety (*the ugly*). Empirical evidence from included studies indicates that, in the absence of more robust accountability structures, model cards may be used for performative transparency and ethics washing rather than as a functional safety tool.

Regarding RQ2, the analysis indicates that the tooling for model card generation has evolved from rigid, template-based utilities toward more intelligent, workflow-integrated systems. Early tools such as the Model Card Toolkit, which focused on schema conformity, experienced limited adoption due to high friction and lack of integration into development workflows. Recent approaches, including workflow-embedded assistants such as DocML and retrieval-augmented generation tools like RiskRAG, demonstrate significant potential to enhance documentation quality and developer engagement. However, a critical deficiency still remains. Current tools prioritize the speed of generation over the verification of content, leading to the creation of plausible but unverified claims.

7.2 Contribution

This review contributes to the body of knowledge on ML documentation by giving a comprehensive overview of the fragmented research landscape regarding model cards. This analysis categorizes the achievements, pitfalls, and consequences of model card adoption through a reproducible selection process, mapping these empirical realities directly to *the good, bad, and ugly* framework. By combining recurring findings across the broader research landscape, this study identifies the specific patterns of incomplete reporting and misleading documentation that currently limit the effectiveness of model cards as transparency instruments.

Furthermore, this research finds a critical misalignment by positioning the tools against the adoption gaps: while current tools effectively reduce the friction of creation, they largely fail to address the deeper issues of verification and accountability. This insight exposes that the current tooling ecosystem is solving for efficiency rather than integrity, leaving the risks of ethics washing and performative compliance unaddressed. Finally, by highlighting the persistent tension between technical reproducibility and broad accessibility, this work challenges the efficacy of the "single-document" paradigm, suggesting a need for specialized artifacts to serve distinct stakeholder groups effectively.

7.3 Future Work

Based on the gaps identified in this review, we propose several directions for future research. Firstly, tooling efforts should prioritize verification over generation. Rather than depending on LLMs to generate documentation, future work should investigate automated auditing frameworks that evaluate whether a model's actual behavior corresponds to the claims made in its documentation.

Another promising direction involves exploring interactive or dynamic model cards that move beyond static formats. The integration of documentation with monitoring pipelines has the potential to integrate the real-time update of performance and drift information.

Moreover, as regulatory frameworks such as the EU AI Act [12] are implemented, empirical studies are necessary to evaluate whether these legal requirements enhance the quality of risk reporting or just increase the volume of defensively crafted, legally sanitized documentation.

8. REFERENCES

[1] Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. Model Cards for Model Reporting. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, FAT* '19, pages 220–229, New York, NY, USA, 2019. Association for Computing Machinery. event-place: Atlanta, GA, USA.

[2] Avinash Bhat, Austin Coursey, Grace Hu, Sixian Li, Nadia Nahar, Shurui Zhou, Christian Kästner, and Jin L.C. Guo. Aspirations and Practice of ML Model Documentation: Moving the Needle with Nudging and Traceability. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, New York, NY, USA, 2023. Association for Computing Machinery. event-place: Hamburg, Germany.

[3] Google DeepMind. Model Cards. `https://deepmind.google/models/model-cards/`, 2024. Accessed: 2025-12-21.

[4] David Adkins, Bilal Alsallakh, Adeel Cheema, Narine Kokhlikyan, Emily McReynolds, Pushkar Mishra, Chavez Procope, Jeremy Sawruk, Erin Wang, and Polina Zvyagina. Prescriptive and Descriptive Approaches to Machine-Learning Transparency. In *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems*, CHI EA '22, New York, NY, USA, 2022. Association for Computing Machinery. event-place: New Orleans, LA, USA.

[5] Haoyu Gao, Mansooreh Zahedi, Christoph Treude, Sarita Rosenstock, and Marc Cheong. Documenting Ethical Considerations in Open Source AI Models. In *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '24, pages 177–188, New York, NY, USA, 2024. Association for Computing Machinery. event-place: Barcelona, Spain.

[6] Hong Shen, Leijie Wang, Wesley H. Deng, Ciell Brusse, Ronald Velgersdijk, and Haiyi Zhu. The Model Card Authoring Toolkit: Toward Community-Centered, Deliberation-Driven AI Design. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '22, pages 440–451, New York, NY, USA, 2022. Association for Computing Machinery. event-place: Seoul, Republic of Korea.

[7] Pooja S. B. Rao, Sanja Šćepanović, Ke Zhou, Edyta Paulina Bogucka, and Daniele Quercia. RiskRAG: A Data-Driven Solution for Improved AI Model Risk Reporting. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, CHI '25, New York, NY, USA, 2025. Association for Computing Machinery.

[8] Weixing Liang, Nazneen Fatema Rajani, Xinyu Yang, Ezinwanne Ozoani, Eric Wu, Yiqun T. Chen, Daniel Scott Smith, and James Y. Zou. Systematic Analysis of 32,111 AI Model Cards Characterizes Documentation Practice in AI. *Nature Machine Intelligence*, 6(7):744 – 753, 2024. Type: Article.

[9] Modelcard Creator - a Hugging Face Space by huggingface. `https://huggingface.co/spaces/huggingface/Model_Cards_Writing_Tool`. Accessed: 2026-01-25.

[10] Tajkia Rahman Toma, Balreet Grewal, and Cor-Paul Bezemer. Answering User Questions About Machine Learning Models Through Standardized Model Cards. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, pages 1488–1500, 2025.

[11] Iman Naja, Milan Markovic, P. Edwards, Wei Pang, Caitlin D. Cottrill, and Rebecca Williams. Using Knowledge Graphs to Unlock Practical Collection, Integration, and Audit of AI Accountability Information. *IEEE Access*, 10:74383 – 74411, 2022. Type: Article.

[12] Future of Life Institute. EU Artificial Intelligence Act: Up-to-Date Developments and Analyses. `https://artificialintelligenceact.eu/`, 2024. Accessed: 2025-12-20.

[13] Cara Ellen Appel. Expanding ML-Documentation Standards for Better Security. In *2025 IEEE 33rd International Requirements Engineering Conference Workshops (REW)*, pages 275–282, 2025.

[14] Tim Puhlfürß, Julia Butzke, and Walid Maalej. Model Cards Revisited: Bridging the Gap Between Theory and Practice for Ethical AI Requirements. In *2025 IEEE 33rd International Requirements Engineering Conference (RE)*, pages 280–291, 2025.

[15] Ezi Ozoani, Marissa Gerchick, and Margaret Mitchell. Model Card Guidebook. `https://huggingface.co/docs/hub/en/model-card-guidebook`, 2022. Accessed: 2026-01-25.

[16] Paola Natalia Cañas, Marcos Nieto, Oihana Otaegui, and Igor Rodríguez. A Methodology to Enhance Transparency for Trustworthy Artificial Intelligence for Cooperative, Connected, and Automated Mobility. *SAE International Journal of Connected and Automated Vehicles*, 8(1), 2024. Type: Article.

[17] Barbara Kitchenham et al. Procedures for Performing Systematic Reviews. *Keele, UK, Keele University*, 33(2004):1–26, 2004.

[18] Daniel Hershcovich, Nicolas Webersinke, Mathias Kraus, Julia Anna Bingler, and Markus Leippold. Towards Climate Awareness in NLP Research. pages 2480 – 2494, 2022. Type: Conference paper.

[19] Abhishek Wadhwani and Priyank Jain. Machine Learning Model Cards Transparency Review: Using Model Card Toolkit. In *2020 IEEE Pune Section International Conference (PuneCon)*, pages 133–137, 2020.

[20] José Luiz Nunes, Gabriel Diniz Junqueira Barbosa, Clarisse S. de Souza, and Simone Diniz Junqueira Diniz Junqueira Barbosa. Using Model Cards for Ethical Reflection on Machine Learning Models: An Interview-Based Study. *Journal on Interactive Systems*, 15(1):1 – 19, 2024. Type: Article.

[21] David Adkins, Bilal Alsallakh, Adeel Cheema, Narine Kokhlikyan, Emily McReynolds, Pushkar Mishra, Chavez Procope, Jeremy Sawruk, Erin Wang, and Polina Zvyagina. Method Cards for Prescriptive Machine-Learning Transparency. In *Proceedings of the 1st International Conference on AI Engineering: Software Engineering for AI*, CAIN '22, pages 90–100, New York, NY, USA, 2022. Association for Computing Machinery. event-place: Pittsburgh, Pennsylvania.

[22] José Luiz Nunes, Gabriel Diniz Junqueira Barbosa, Clarisse S. de Souza, Hélio Côrtes Vieira Lopes, and Simone Diniz Junqueira Diniz Junqueira Barbosa. Using Model Cards for Ethical Reflection: A Qualitative Exploration. In *ACM International Conference Proceeding Series*, 2022. Type: Conference paper.

[23] Anamaria Crisan, Margaret Drouhard, Jesse Vig, and Nazneen Rajani. Interactive Model Cards: A Human-Centered Approach to Model Documentation. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '22, pages 427–439, New York, NY, USA, 2022. Association for Computing Machinery. event-place: Seoul, Republic of Korea.

[24] Vanessa Bracamonte, Sebastian Pape, Sascha Löbner, and Frederic Tronnier. Effectiveness and Information Quality Perception of an AI Model Card: A Study Among Non-Experts. In *2023 20th Annual International Conference on Privacy, Security and*

Trust (PST), pages 1–7, 2023.

[25] Stephen Henry Gilbert, Rasmus Adler, Taras Holoyad, and Eva Weicken. Could Transparent Model Cards With Layered Accessible Information Drive Trust and Safety in Health AI? *npj Digital Medicine*, 8(1), 2025. Type: Note.

[26] Charles E. Kahn, Abhinav Suri, Safwan S. Halabi, and Hari M. Trivedi. Towards a Formal Description of Artificial Intelligence Models and Datasets in Radiology. *Lecture Notes in Computer Science*, 14845 LNAI:140 – 144, 2024. Type: Conference paper.

[27] Thomas Bonnier and Benjamin Bosch. Towards Safe Machine Learning Lifecycles with ESG Model Cards. *Lecture Notes in Computer Science*, 14182 LNCS:369 – 381, 2023. Type: Conference paper.

[28] Jiarui Liu, Wenkai Li, Zhijing Jin, and Mona T. Diab. Automatic Generation of Model and Data Cards: A Step Towards Responsible AI. volume 1, pages 1975 – 1997, 2024. Type: Conference paper.

[29] Vasisht Duddu, Oskari J Rvinen, Lachlan J. Gunn, and N. Asokan. Laminator: Verifiable ML Property Cards Using Hardware-Assisted Attestations. pages 317 – 328, 2025. Type: Conference paper.

[30] Muhammad Tuan Amith, Licong Cui, Degui Zhi, Kirk E. Roberts, Xiaoqian Jiang, Fang Li, Evan Yu, and Cui Tao. Toward a Standard Formal Semantic Representation of the Model Card Report. *BMC Bioinformatics*, 23, 2022. Type: Article.

[31] Mark D. Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E. Bourne, Jildau Bouwman, Anthony J. Brookes, Tim Clark, Mercè Crosas, Ingrid Dillo, Olivier Dumon, Scott Edmunds, Chris T. Evelo, Richard Finkers, Alejandra Gonzalez-Beltran, Alasdair J. G. Gray, Paul Groth, Carole Goble, Jeffrey S. Grethe, Jaap Heringa, Peter A. C. 't Hoen, Rob Hooft, Tobias Kuhn, Ruben Kok, Joost Kok, Scott J. Lusher, Maryann E. Martone, Albert Mons, Abel L. Packer, Bengt Persson, Philippe Rocca-Serra, Marco Roos, Rene van Schaik, Susanna-Assunta Sansone, Erik Schultes, Thierry Sengstag, Ted Slater, George Strawn, Morris A. Swertz, Mark Thompson, Johan van der Lei, Erik van Mulligen, Jan Velterop, Andra Waagmeester, Peter Wittenburg, Katherine Wolstencroft, Jun Zhao, and Barend Mons. The FAIR Guiding Principles for Scientific Data Management and Stewardship. *Scientific Data*, 3(1):160018, March 2016.

[32] Thomas M. Booth and Sudipto Ghosh. Machine Learning Model Cards Toward Model-Based System Engineering Analysis of Resource-Limited Systems. In *Proceedings of SPIE - The International Society for Optical Engineering*, volume 12547, 2023. Type: Conference paper.