Contemporary Software Modernization: Strategies, Driving Forces, and Research Opportunities

WESLEY K. G. ASSUNÇÃO, Department of Computer Science, North Carolina State University, Raleigh, North Carolina, United States
LUCIANO MARCHEZAN, Institute of Software Systems Engineering, Johannes Kepler University Linz, Linz, Austria
LAWRENCE ARKOH, Department of Computer Science, North Carolina State University, Raleigh, North Carolina, United States
ALEXANDER EGYED, Institute of Software Systems Engineering, Johannes Kepler University Linz, Linz, Austria
RUDOLF RAMLER, Software Competence Center Hagenberg GmbH, Hagenberg, Austria

Software modernization is a common activity in software engineering, since technologies advance, requirements change, and business models evolve. Differently from conventional software evolution (e.g., adding new features, enhancing performance, or adapting to new requirements), software modernization involves re-engineering entire legacy systems (e.g., changing the technology stack, migrating to a new architecture style, or programming paradigms). Given the pervasive nature of software today, modernizing legacy systems is paramount to provide customers with competitive and innovative products and services, while keeping companies profitable. Despite the prevalent discussion of software modernization in gray literature, and the many papers in the literature, there is no work presenting a "big picture" of contemporary software modernization, describing challenges, and providing a well-defined research agenda. The goal of this work is to describe the state of the art in software modernization in the past 10 years. We collect the state of the art by performing a rapid review (searching five digital libraries), identifying potential 3,460 studies, leading to a final set of 126. We analyzed these studies to understand which strategies are employed, the driving forces that lead organizations to modernize their systems, and the challenges that need to be addressed. The results show that studies in the last 10 years have explored eight strategies for modernizing legacy systems, namely cloudification, architecture redesign, moving to a new programming language, targeting reuse optimization, software modernization for new hardware integration, practices to leverage automation, database modernization, and digital transformation. Modernization is triggered by 14 driving forces, with the most common ones being reducing operational costs, improving performance and scalability, and reducing complexity. In

The research reported in this paper has been supported by the Linz Institute of Technology (LIT-2023-12-SEE-128), by the Austrian ministries BMK and BMAW and the State of Upper Austria in the frame of the SCCH COMET competence center INTEGRATE (FFG 892418), as well as the Austrian Science Fund (FWF P31989-N31).
Authors' Contact Information: Wesley K. G. Assunção (corresponding author), Department of Computer Science, North Carolina State University, Raleigh, North Carolina, United States; e-mail: wguezas@ncsu.edu; Luciano Marchezan, Institute of Software Systems Engineering, Johannes Kepler University Linz, Linz, Austria; e-mail: lucianomarchp@gmail.com; Lawrence Arkoh, Department of Computer Science, North Carolina State University, Raleigh, North Carolina, United States; e-mail: larkoh@ncsu.edu; Alexander Egyed, Institute of Software Systems Engineering, Johannes Kepler University Linz, Linz, Austria; e-mail: alexander.egyed@jku.at; Rudolf Ramler, Software Competence Center Hagenberg GmbH, Hagenberg, Austria; e-mail: rudolf.ramler@scch.at.

ACM 1557-7392/2025/5-ART142
https://doi.org/10.1145/3708527

addition, based on the analysis of existing literature, we present a detailed discussion of research opportunities in this field. The main challenges are providing tooling support, followed by defining a modernization process and considering better evaluation metrics. The main contribution of our work is to equip practitioners and researchers with knowledge of the current state of contemporary software modernization so that they are aware of practices and challenges to be addressed when deciding to modernize legacy systems.

CCS Concepts: • **Software and its engineering → Software evolution**; **Software architectures**; • **Computer systems organization → Cloud computing**;

Additional Key Words and Phrases: Software Evolution, Software Migration, Re-designing, Re-engineering

ACM Reference format:
Wesley K. G. Assunção, Luciano Marchezan, Lawrence Arkoh, Alexander Egyed, and Rudolf Ramler. 2025. Contemporary Software Modernization: Strategies, Driving Forces, and Research Opportunities. *ACM Trans. Softw. Eng. Methodol.* 34, 5, Article 142 (May 2025), 35 pages.
https://doi.org/10.1145/3708527

1 Introduction

Throughout the life of a software system, its architecture decays, its underlying technologies become obsolete, the user requirements change, or the company's business models evolve—ultimately, causing the software to morph into what we call legacy systems [16]. The vast majority of software currently in operation is long-lived systems that represent many years of competitive knowledge and business value [53]. Although fundamental for the business operations, due to extensive maintenance and obsolete technology, legacy systems are costly to maintain, more exposed to cybersecurity risks, less effective in meeting their intended purpose, and push up costs of digital transformation [15, 66, 84]. For example, the US government spent more than $100 billion in fiscal year 2023 on IT, of which about 80% was used to operate and maintain legacy systems [85]. In addition, the UK government spends £4.7 billion a year on IT in all departments, and £2.3 billion goes on patching up systems, some of which date back 30 years or more [66].

To remain competitive, companies must modernize their legacy systems [10]. Furthermore, with modernization, companies can preserve the hard-earned knowledge acquired through many years of system development [53, 79, 88]. According to Seacord et al. [79] "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." Differently from conventional software evolution (e.g., add new features, enhance performance, or adapt to new requirements), software modernization involves re-engineering[1] or migrating[2] entire legacy systems. Thus, software modernization refers to the conversion, rewriting, redesign, or porting of a legacy system to a modern computer programming language, protocols, or hardware platform [54, 78]. The process of modernizing a legacy system has benefits such as ease of engineering activities, satisfying user needs, achieving new business goals, or reducing costs [79]. Furthermore, modernization is a means to leverage digital transformation [15], as it allows the use of emerging and disruptive technologies such as AI, high-performance computing, cloud computing, the Internet of Things, industry 4.0, robotics, and big data [60].

[1]Software re-engineering is the examination and alteration of a system to reconstitute it in a new form, e.g., adopting a new architecture style [27].
[2]Software migration is transferring an existing software system to a new environment to provide elaborated methods and techniques allowing already established software systems to benefit from the advantages of new technologies, e.g., start using parallel computing [48].

Given the pervasive nature of software today, modernizing legacy systems is essential to provide customers with competitive, innovative, and sustainable products and services and to support companies to benefit from new technologies [20, 41, 53, 80]. In the literature, we can find different modernization strategies [60, 81]. For example, restructuring systems using components, adopting aspect-oriented development, re-engineering system variants into **software product lines (SPLs)**, migrating to microservices, and supporting new hardware (e.g., multi-core and GPU devices). Even the software development process has been modernized, with the adoption of agile methods [62] and DevOps [25]. Additionally, modernization has different driving forces and impacts related to *organizational*, *operational*, and *technological* aspects [88]. For instance, modernization can focus on independence for agile teams, optimize the deployment, ease the inclusion of innovation, facilitate scalability, support new hardware, or explore new market segments [81, 88].

Despite the prevalent discussion of software modernization in gray literature [66, 84, 85], and the pieces of work in the literature [38, 49, 53, 54, 60, 63, 81], there is no work presenting a "big picture" of contemporary[3] software modernization. It is still unclear what are the current modernization strategies, driving forces that trigger the modernizations, and well-defined research agenda with challenges to be addressed in the following years. As pieces of work span across many years and focus on modernizing for different purposes, there is a need for discussing modernization in the context of contemporaneous software development. The studies that try to organize the existing literature have several limitations. They only present an overview of the state of the art [60]; are based on few case studies or a subset of existing literature [49, 63, 81]; are outdated regarding current emerging/disruptive technologies [38, 49, 53, 54]; partially cover the modernization lifecycle, and rarely take into account organizational, operational, and technological aspects [60, 88].

Based on the limitations of existing work, we conducted a study guided by three **research questions (RQs)**. These RQs aim to investigate the existing strategies to modernize legacy systems (RQ1: FROM>To Strategies), what are the common reasons and pains that lead to the modernizing (RQ2: Driving forces), and the open challenges and research opportunities in the field (RQ3: Challenges). To answer these questions, the goal of this work is to describe the current state of the art on software modernization, considering studies in the past 10 years. By performing a **rapid review (RR)** [21, 22], we collect the pieces of work in five **digital libraries (DLs)**, identifying ≈3.5k studies. Guided by the RQs, after a careful analysis of the content of these papers, we selected a final set of 126 used as a source for our analysis.

The results show that eight main strategies (RQ1) for modernizing legacy systems have been explored in the last 10 years. These strategies are cloudification of legacy systems, architecture redesign, moving to a new programming language, targeting reuse optimization, software modernization for new hardware integration, practices to leverage automation, database modernization, and digital transformation. We found 14 distinct driving forces (RQ2) that are considered for starting a modernization process, namely reduce operational costs, promote interoperability among systems, improve performance and scalability, foster innovation, achieve better flexibility and efficiency, reduce complexity, enhance agility, explore new business opportunities, reduce time to market, avoid duplications and inconsistency, leverage reuse opportunities, allow better understanding of the system architecture, provide external services, and improve the quality of data. In addition, based on the analysis of existing literature, we identified 16 challenges for software modernization (RQ3), which allowed us to present a detailed discussion of 27 research opportunities in this field. The main challenges are lack of tooling support, the need for better evaluation metrics, applying cost-benefit analysis to align the modernization with the business goals, maintaining the consistency between legacy and modern systems, and keeping the correctness of the modern system.

[3]By "contemporary," we refer to the context of modern software development, which comprehends technologies, practices, and companies' needs observed in the last 10 years.

The main contributions of this work are:

— *Review of the State of the Art:* We present a comprehensive and contemporaneous review of software modernization in the last 10 years, with the goal of calling attention to advances in research and practice toward dealing with legacy systems. We contextualize the different strategies, together with driving forces for software modernization, and introduce a research agenda with 27 research opportunities to be taken into account in the following years.
— *Support the Decision on the Right Modernization Strategy:* We describe the modernization strategies and their driving forces. Based on specific reasons or pains, some approaches are more appropriate than others. Thus, our work can help practitioners to make informed decisions, avoiding deciding only based on technological "hypes." Choosing the wrong approach can lead to inefficiency and frustration during modernization. Our work motivates the discussion on software modernization and alerts companies about challenges they may incur when choosing a specific modernization strategy.
— *Train the Workforce with Skills for Dealing with Modernization:* Our results reveal several strategies and challenges that must be considered during modernization. Thus, educators can benefit from our work as a basis for designing new courses in academia, with the goal of training the workforce in charge of operationalizing the modernization of legacy systems.

2 Background

Seacord et al. [79] presented software modernization as a remedy to face the legacy system crisis in the early 2000s. They discussed how to keep or add business value through modern technologies, reducing operational costs, and dealing with technical aspects (e.g., allowing better reuse and easier maintenance) [79]. The focus on software modernization is to preserve knowledge represented in legacy systems while employing disruptive and emerging technologies to the benefit of users, companies, and society.

Disruptive and emerging technologies such as ubiquitous embedded computing, connectivity, and flexible value streams are enablers of digital transformation, a technology-driven continuous change process [35, 70]. Modernization is a means to leverage the digital transformation [15, 60]. According to the report, "Shaping the digital transformation in Europe" [37], from the European Commission, by 2030 the cumulative additional GDP contribution of new digital technologies could amount to 2.2 trillion Euros. The same report points, for example, that digital transformation (e.g., online interactions and monitoring, paperless data, workflow automation, decision support, and appropriate patient self-care) in the EU health sector can achieve efficiencies of up to 120 billion Euros. However, a survey revealed a staggering 88% of services companies that have had a digital project fail or reduced in scope due to the cost of making changes to legacy technology, making companies spend twice the industry average on digital transformation [15].

In the literature, we can see the three ways of moving a legacy system toward a modern system: (i) big bang or cold turkey [29], which is the replacement of the legacy system with the modern one at once; (ii) incremental modernization, following a strangler pattern [40], in which parts of the legacy systems are incrementally replaced by modern parts [79, 88]; and (iii) the coexistence, in which legacy and modern parts operate together in one system [75]. Furthermore, there are different types of modernization a company should decide on, which must be based on a portfolio analysis. Figure 1 presents the portfolio analysis quadrant (extended from Seacord et al. [79]) to support the decision among the types of modernization. In addition to the *technical quality* and *business value* dimensions, we introduce *innovation* as an additional dimension that is achieved by new disruptive and emerging technologies (i.e., driving forces for modernization). The five quadrants in these three dimensions are:

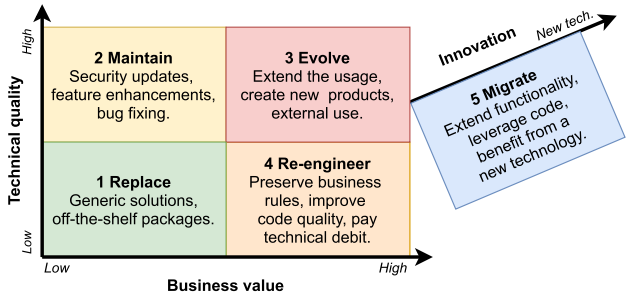


Fig. 1. Extended quadrant of the portfolio analysis for software modernization, adapted from [78].

(1) *Replace*: Legacy systems that have low business value and low technical quality, i.e., accumulated technical debt, should be replaced by new systems, using generic solutions or off-the-shelf systems, instead of undergoing a re-engineering or migration process.
(2) *Maintain*: Systems with high technical quality and low business value should not require modernization effort, but traditional maintenance activities should be used, just to keep them operating and meeting customers' needs.
(3) *Evolve*: High-quality legacies with high business value should be actively evolved using traditional evolutionary development practices for introducing new features, new products, or even serving as a third party for other systems.
(4) *Re-Engineer*: Systems with high business value and low technical quality should be re-engineered to preserve business value, i.e., external quality, and manage the technical debt, i.e., internal quality. This type of modernization can be transparent to the end user.
(5) *Migrate*: When the system has high business value and a company decides to drive innovation with emerging or disruptive technologies, independently of the system's technical quality, a migration to the desired new technologies should take place. This is, for example, the case when companies foster a digitalization initiative.

In the literature, we can find studies on software modernization to retain the business value of legacy systems [60, 81]. For example, restructuring systems using components [26, 36, 55], adoption of aspect-oriented development [5, 43, 74], re-engineering of system variants into SPLs [8, 9, 59, 73], migration to microservices [19, 23, 33, 57, 82, 86, 88], supporting new devices or pieces of hardware (e.g., from single-core to multi-core machines) [68, 72, 76], classical information systems to quantum computing [71, 90], and leveraging the use of AI/ML/Foundation Models [3]. Even the software development process has been modernized, such as the adoption of agile methods [62] and DevOps [18, 25]. Modernization also has different driving forces and impacts related to *organizational*, *operational*, and *technological* aspects. The modernization can focus on the independence of teams, optimizing deployment, adding innovation, facilitating scalability, or exploring new market segments [81, 88].

Despite existing literature, the work on software modernization has several limitations: studies only present an overview of the state of the art [60], are based on few case studies or a subset of existing literature [49, 63, 81], are outdated regarding current emerging/disruptive technologies [38, 49, 53, 54], partially cover the modernization lifecycle, and rarely take into account *organizational*, *operational*, and *technological* aspects [60, 88]. Furthermore, these studies are limited to exploring contemporary needs (e.g., digital transformation [60]). Thus, to have a comprehensive view of the state of the art on software modernization, we conducted a study analyzing the pieces of work published on this topic in the last 10 years. This study is presented in the next section.

3 Study Design

To describe the state of the art and provide a research agenda on software modernization, we adopt an RR protocol [46, 83]. RR is an emerging (considering software engineering) alternative to **systematic literature reviews (SLR)** [56], focusing on more practical problems, while analyzing a comprehensive set of studies [21, 22]. Despite being different from traditional systematic mappings (SMS) and SLR, an RR still enforces a well-planned and executed protocol to obtain the desired results. The first step of this protocol is to define the goal and RQs. Given the aims of our study, the main goal of this study is:

> ***Goal:*** *Review the literature on software modernization for the purpose of identifying the existing strategies, main driving forces to modernize legacy systems, and challenges that still need to be addressed.*

3.1 RQ

The goal of our study originated three RQs, which guided the search strategy, selection of primary sources, and data extraction, as follows:

- *RQ1: (From> To Strategies) What are the contemporaneous strategies to modernize legacy systems?* In our study, we first focus on identifying which modernization strategies have been explored in the literature. To understand the modernization strategies, we consider what was the legacy system and what is the modern system for each study, meaning we investigated the "[From] legacy system [To] target system" strategies explored in the literature. By answering this question, we have an overview of the contemporaneous modernization strategies, since our study focuses on the last 10 years (i.e., 2014–2024).
- *RQ2: (Driving Forces) What are the common driving forces for modernizing a legacy system?* As important as knowing which modernization strategies have been explored in the literature, is to know "why" this modernization took place. In this question, we aim to investigate what are the driving forces that triggered software modernization. By understanding the reasons and pains that lead companies to decide to modernize their legacy systems, we are able to know what are the main limitations of legacy systems.
- *RQ3: (Challenges) What are the challenges and research opportunities in the field of software modernization?* Given the paramount importance of legacy systems for users, companies, and society, in this question we focus on providing a well-defined research agenda on the topic. We explore existing literature to grasp what are the open challenges, observed limitations, and future work. Based on this analysis, we present research opportunities to guide the advancement of knowledge on the topic of software modernization in the following years.

3.2 Search Strategy

For the search of primary sources, we defined the relevant terms, and their synonyms, related to the goal and RQs of our study. These terms and synonyms are used to establish our search string:

> ***Search String****: "(migrat* OR moderniz* OR transform* OR re-engineer* OR restructur*) AND ("legacy software" OR "existing software" OR "legacy system" OR "existing system" OR "legacy program" OR "existing program") AND (practic* OR industr* OR real-world))".*

Table 1. IC/EC to Filter Relevant Studies

IC	EC
IC1. The paper's main contribution must be related to software modernization. IC2. The paper discusses/addresses a practical problem in software modernization.	EC1. The paper has less than six pages. EC2. The paper is not written in English. EC3. The paper was published before the year 2014. EC4. The full text of the paper is not available online. EC5. The paper is a conference version of a journal extended publication.

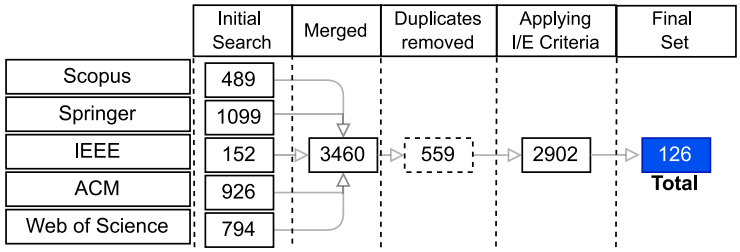


Fig. 2. DLs, steps, and number of retrieved papers for the search and selection of papers.

We then derive a specific search string for each of the used DLs, namely Scopus, Springer, IEEE, ACM, and Web of Science.[4] These specific search strings were defined after executing, analyzing, and refining the base search string (presented above) to the point they returned a satisfactory result. Additionally, when allowed by the DL, we included keywords and filters in the strings that are related to the **inclusion criteria (IC)/exclusion criteria (EC)**, such as to search only papers published in English. Finally, for the screening of the studies returned from each DL, according to the protocol of our RR, we defined these IC and EC to select only relevant papers, as presented in Table 1.

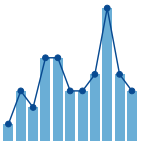
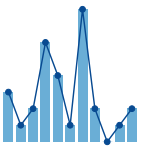
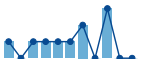
Two of the authors of this study applied the IC/EC to all 2,902 papers. When there was a conflict between these two authors (i.e., one author decided to include and the other to exclude), a third author also applied the criteria, making the final decision about the IC/EC.

3.3 Search and Selection

Figure 2 presents the number of studies retrieved from each DL. The search was carried out in August 2024. After performing the search in the five DLs, a total of 3,460 studies were retrieved. From these studies, 559 were duplicates. We considered a study as a duplicate if this study was exactly the same paper found in different DLs (e.g., Scopus and ACM). After removing the duplicates, 2,902 studies remained for the application of the IC/EC, based on the criteria presented in Table 1. One important criterion is given by EC5, where we excluded papers that were a conference version of a paper later extended to a journal (i.e., we included the journal version of the paper as it contained a more complete version of the contributions). After applying these criteria, 126 studies remained. The complete list of primary sources is presented in Appendix A and also available in our supplementary material [44].

[4]Details about the search execution, including the search strings are available at our online repository [44].

Table 2. Types of Modernization Strategies Observed in the Last 10 Years

Strategies	Total	Primary Sources	Pub. Trends
Cloudification	40	[S1, S8, S9, S11, S15, S20, S21, S22, S25, S26, S27, S29, S31, S32, S39, S41, S44, S50, S54, S56, S63, S67, S68, S70, S71, S72, S73, S74, S88, S89, S92, S107, S111, S113, S117, S119, S120, S122, S124, S126]	
Architecture redesign	30	[S2, S4, S6, S7, S10, S13, S14, S16, S24, S33, S36, S43, S52, S53, S55, S61, S65, S69, S76, S81, S83, S84, S85, S86, S90, S91, S108, S112, S121, S123]	
New programming language	23	[S17, S23, S30, S34, S40, S42, S45, S46, S48, S49, S51, S62, S78, S79, S97, S99, S102, S103, S104, S105, S109, S115, S118]	
Reuse optimization	10	[S57, S58, S59, S60, S77, S80, S82, S87, S94, S98]	
New hardware integration	8	[S5, S18, S37, S38, S47, S66, S96, S100]	
Leverage automation	7	[S3, S12, S28, S64, S101, S106, S116]	
Database modernization	5	[S19, S93, S95, S110, S125]	
Digital transformation	3	[S35, S75, S114]	

The Column Pub. Trends presents the number of studies exploring each strategy in the last 10 years.

3.4 Data Extraction

Based on the guidelines for RR [21, 83], the typical analysis of primary sources employs a descriptive synthesis approach rather than a quantitative meta-analysis. To extract the data required to address the three RQs of our study, we conducted a manual review of the selected papers, aiming to classify and generate a descriptive synthesis for each. The following fields were extracted from each paper: (i) Modernization strategies and origin and target technology or architecture: we extract these strategies from the description of the approach and the intended results; (ii) Description of the driving forces and the reasons to modernize: we extract this information from the motivation and practical problems described in the studies; and (iii) Challenges and main research opportunities: these pieces of information are extracted from limitations and future work described in each paper analyzed.

4 Results and Analysis

This section presents the strategies, driving forces, and challenges for the 126 primary sources, which are reported in Appendix A. Table 2 lists the eight modernization strategies identified, which have been applied across various domains of software engineering, such as moving systems to

Table 3. Driving Forces that Triggered Software Modernization in the Last 10 Years Grouped into Operational, Technical, and Organizational

	Driving Forces	Total	Primary Sources
Operational	Reduce operational costs	62	[S1, S2, S3, S8, S9, S11, S14, S15, S17, S21, S22, S23, S24, S27, S30, S33, S34, S37, S39, S40, S41, S42, S43, S45, S48, S49, S51, S52, S56, S60, S62, S63, S64, S65, S68, S70, S71, S72, S74, S75, S76, S77, S88, S89, S95, S97, S98, S99, S102, S103, S105, S107, S109, S111, S112, S113, S114, S116, S117, S118, S119, S122]
	Better flexibility and efficiency	13	[S10, S16, S23, S30, S42, S52, S65, S86, S93, S99, S109, S115, S119]
	Enhance agility	3	[S4, S63, S113]
Technical	Improve performance and scalability	39	[S1, S5, S8, S9, S11, S18, S20, S22, S26, S27, S31, S39, S41, S42, S44, S47, S50, S55, S57, S63, S66, S67, S73, S74, S81, S85, S88, S89, S92, S96, S100, S107, S113, S115, S119, S122, S124, S125, S126]
	Reduce complexity	22	[S2, S12, S17, S23, S27, S30, S34, S40, S45, S48, S49, S51, S62, S68, S74, S87, S97, S102, S103, S105, S106, S118]
	Architecture understanding	18	[S1, S6, S7, S13, S25, S29, S36, S41, S52, S62, S63, S79, S84, S87, S94, S108, S117, S122]
	Leverage reuse opportunities	16	[S9, S14, S20, S28, S36, S58, S59, S60, S61, S77, S80, S82, S84, S91, S98, S101]
	Avoid duplications and inconsistency	15	[S6, S7, S21, S24, S33, S36, S52, S63, S76, S83, S85, S87, S94, S108, S112]
	Improve data quality	7	[S13, S41, S83, S87, S93, S94, S110]
	Interoperability among systems	6	[S21, S29, S35, S75, S86, S121]
Organizat.	Foster innovation	14	[S10, S13, S15, S19, S53, S54, S62, S78, S90, S92, S104, S110, S123, S125]
	New business opportunities	9	[S4, S17, S38, S43, S58, S63, S80, S120, S122]
	Faster time to market	6	[S20, S68, S87, S89, S94, S120]
	Provide external services	2	[S20, S106]

the cloud, improving the system architecture, supporting digital transformation, focusing on the database, and enabling the integration of new hardware. For the driving forces, shown in Table 3, we organize them into operational, technical, and organizational, with a total of 14 distinct driving forces. Table 4 presents the 16 challenges identified in the set of studies analyzed. We present details of the eight strategies in the following sections. We also discuss the driving forces related to each strategy, namely the factors or goals for conducting such migrations. Despite the successful migration of many projects and the introduction of novel migration procedures, both researchers

Table 4. Categories of Challenges for Software Modernization Research and Practice

Challenges	Total	Primary Sources
Provide tooling support	45	[S1, S2, S3, S4, S6, S16, S17, S23, S24, S26, S29, S30, S34, S38, S42, S43, S47, S50, S52, S53, S55, S56, S61, S65, S67, S69, S71, S77, S78, S79, S80, S81, S83, S84, S87, S89, S91, S97, S103, S104, S105, S108, S120, S123, S125]
Define a modernization process	21	[S1, S15, S16, S17, S33, S40, S41, S42, S49, S52, S60, S67, S70, S72, S76, S82, S94, S97, S102, S108, S114]
Consider better evaluation metrics	20	[S8, S9, S10, S15, S35, S63, S76, S83, S84, S86, S88, S91, S93, S100, S107, S112, S113, S118, S119, S122]
Service identification/legacy decomposition	12	[S1, S8, S15, 115, S22, S41, S50, S73, S92, S113, S124, S124]
Explore automatic computing	11	[S7, S15, S22, S50, S53, S62, S64, S81, S98, S106, S116]
Prepare the legacy data/system	11	[S3, S25, S36, S60, S74, S76, S77, S87, S107, S111, S119]
Migration between different data models	10	[S5, S13, S19, S31, S59, S87, S94, S95, S106, S110]
Correctness of the new architecture	9	[S20, S21, S30, S45, S49, S97, S99, S111, S118]
Apply cost-benefit analysis	8	[S11, S18, S39, S57, S58, S80, S98, S109]
Consistency between legacy and modern	7	[S2, S7, S27, S34, S37, S83, S126]
Predict/optimize performance	7	[S17, S28, S37, S66, S96, S103, S115]
Deal with security concerns	4	[S30, S66, S68, S90]
Understand the legacy architecture	4	[S8, S12, S111, S121]
Hybrid architecture (legacy+modern)	2	[S44, S54]
Correctness of the modern system	1	[S117]
Use digital twins	1	[S75]

and practitioners encountered various challenges, which we present together with their respective research opportunities.

4.1 Cloudification

Cloudification stands out as the most adopted modernization strategy (41 studies). This was expected due to the popularity of cloud providers.[5] For this category, due to many primary sources, we describe three subcategories targeting: (i) the migration of systems to the cloud without referring to any particular architectural style [S26, S27, S31, S39, S42, S44, S54, S56, S67, S68, S89, S117, S126],

[5]Examples of cloud providers are: Google Cloud (https://cloud.google.com/), Microsoft Azure (https://azure.microsoft.com/), and Amazon Web Services (https://aws.amazon.com/).

(ii) the transition of legacy systems to **service-oriented architecture (SOA)** [S1, S25, S29, S70, S71, S72, S120], and (iii) the migration of monolith systems to microservices [S8, S9, S11, S15, S20, S21, S22, S42, S50, S63, S73, S74, S88, S92, S107, S111, S113, S119, S122, S124].

4.1.1 Migration to the Cloud. This strategy for modernization mainly refers to migrating legacy systems operating on-premise to a cloud infrastructure. This process requires moving applications, data, security, and other artifacts to a cloud computing environment. In this subcategory, it is not necessary to change the legacy architecture (i.e., a monolith can be deployed on the cloud).

Driving Forces. The primary motivation to migrate a legacy system from on-premise to a cloud infrastructure is to address *scalability challenges* arising from increasing system demands [S26, S27, S31, S39, S42, S89, S126]. For instance, a study discusses the migration of a real-time geo-social networking application to the cloud due to the growth in data volume, complex search queries, and the absence of real-time features such as instantaneous upload and download of frequently used resources [S27]. Similarly, another paper describes the migration of a geospatial analytics system, initially developed for on-premises infrastructure, to the Siemens MindSphere cloud platform to enable elastic scaling and large-scale data processing [S89]. Another major disadvantage of on-premise infrastructure is the high *operational cost* [S27, S56, S68], primarily due to the expense of maintaining the infrastructure and the need for extensive administrative oversight [S27]. In contrast, cloud providers offer a pay-as-you-go model where costs are based on actual resource usage, and resource management is simplified [S68]. A study, which focused on migration to **function as a service (FaaS)**, highlighted the *reduction in system complexity* associated with cloud migration [S117]. According to the authors, the adoption of FaaS brings benefits such as improved maintainability, testability, and reusability.

Research Opportunities. For this subcategory, we distilled five research opportunities.

Managing Consistency between Legacy and Modern Systems: Among the types of transitions from legacy to modern systems presented in Section 2, the big bang and incremental approaches are explored in the literature, however, little is discussed on how to establish a hybrid environment to allow the development and coexistence of legacy and modern system [S44, S89]. One study suggests future research on scenarios where the legacy and modern systems coexist continuously and in parallel (i.e., a hybrid architecture) for a certain period of time due to missing functionality between target and legacy system [S54]. The authors also recommend exploring potential solutions for situations where functionalities are missing in the new system or when data format incompatibilities arise.

Defining a Process for the Migration: Integrating new solutions into existing systems and the absence of established migration practices are significant challenges. The authors of one paper recommend adopting a multi-criteria decision-making framework for platform selection in the industry [S39]. This idea is further expanded in another paper, where the authors propose a generic model to facilitate cloud migration in terms of cost and quality estimation [S31].

Predicting and Optimizing Performance: One paper acknowledges the fact that pre-warming containers (by using future function call prediction) is a way of addressing the issue of cold starting in FaaS [S117]. A similar issue of cold start was raised in another study, when a company migrated its systems to a serverless architecture (FaaS) [S39].

Providing Tooling Support for Migration to the Cloud: There are several concerns related to the development and enhancement of tools for generating and visualizing architectures to support effective cloud migration planning [S26]. A paper advocates for a tool that abstracts the cloud migration model, offering simple interfaces to incorporate additional migration planning parameters and to track the decision-making process throughout the migration [S89].

4.1.2 Transition to SOA. This modernization strategy primarily focuses on transforming legacy systems into modular and loosely coupled services. These services within the application can be invoked independently, each exposing specific functionality [S72]. In this way, SOA can be used to address some of the inherent challenges of traditional monolithic legacy systems.

Driving Forces. Among the papers in this category, the primary goal for transitioning toward SOA is to *reduce costs* for maintaining the legacy system [S71, S72]. In one study, the authors mention that SOA makes easier the *comprehension of the system's architecture* [S25]. The migration to SOA also leverages *new business opportunities* [S120]. One paper asserts that migrating legacy applications to SOA enhances their robustness and optimizes them to align with business objectives [S70]. A study emphasizes the utilization of relatively independent and reusable services, which effectively address *scalability challenges* in legacy systems [S1]. Other authors discuss the goal of the migration process as integrating core functionalities with other agencies to improve the e-government model, thereby expanding *interoperability among systems* [S71]. They also note that legacy systems, particularly those developed in COBOL, result in increased *operational costs* related to processing power, difficulties in finding affordable programmers skilled in these technologies, and the lack of productive development tools.

Research Opportunities. For this subcategory, we observed two research opportunities.

Exploring Diverse Methodologies for Service Identification: Researchers utilized clustering algorithms to identify services for the target architecture [S25]. However, they note a significant challenge: in tightly coupled codebases with limited modularity, these algorithms may struggle to decompose the system effectively into distinct services. This limitation indicates a need for further research into alternative techniques for service identification that are better suited to legacy systems migrating to SOA [S25]. This perspective is echoed in another study [S1], where the authors propose experimenting with other algorithms as part of their future work. Exploring diverse methodologies for service identification is crucial for improving the accuracy and efficiency of SOA migration processes.

Tooling Support to Detect and Mitigate Antipatterns during the Modernization: In one study, the authors emphasize that while adopting SOA through rapid and cost-effective means may seem appealing, it often results in sub-optimal service quality [S71]. They advocate for robust tooling support to facilitate the migration process, focusing on the correctness of the modern system. For instance, they document a case where a code-first tool, used to generate ASP Web Service interfaces, introduced antipatterns such as Redundant Port Types and Enclosed Data Models. This highlights the need for sophisticated tools that can detect and mitigate such issues during migration. Similarly, a paper discusses the need for the development of automatic packaging techniques to efficiently integrate identified services into the target platform [S1]. This requirement underscores the importance of advanced tooling solutions to streamline the migration and integration processes, enabling a smoother transition to SOA.

4.1.3 Monolith to Microservices. This strategy focuses on splitting a legacy system into small, autonomous, and highly independent services communicating by using lightweight network protocols [S8]. Though microservices is an evolution of SOA [S119], we present our findings on migration to microservices as a separate strategy due to its wide adoption across adopt several industries [S124].

Driving Forces. The main driving forces to migrate from a legacy monolith system to a microservice-based architecture is largely attributed to *scalability issues* [S113, S124], and *faster time to market* enabled by automated cycles with continuous integration and deployments [S9, S20, S74]. We also found that the modernization applied seeks to *improve performanc*e by reducing system latency

[S124], and applying selective scaling to specific resources and application instances [S88]. For example, a study highlighted that the system's high coupling negatively affected performance. When demand increased for a single module, the entire system had to be scaled due to the interconnected nature of the components [S11]. We noted a similar scenario for a bank that needed to attain rapid scalability in order to serve a high number of customers, having a high fluctuating behavior in terms of application usage [S41]. At the operational level, the primary goals of modernization with microservices are to *enhance agility* [S63], *foster innovation* [S15], and *reduce the operation cost* of maintenance [S8, S41], thereby improving *organizational flexibility and efficiency* [S113]. For instance, a paper highlighted a case where different departments within a police intelligence office faced challenges due to a lack of service *interoperability among their systems* [S21]. Additionally, their legacy systems were difficult to maintain, resulting in operational inefficiencies and *data duplication and inconsistencies* issues.

Research Opportunities. For this subcategory, we distilled three research opportunities.

Advancing the Techniques for Monolith Decomposition: Similarly to what was pointed as a challenge for the transition to SOA, for microservices, deciding on how to decompose the legacy systems (i.e., service identification) is a prevalent challenge. The primary step for migrating a legacy system to microservices involves the decomposition of the monolithic architecture [88]. This requires application of dynamic analysis techniques for service identification [S50], allowing engineers to understand how the modern application can be impacted by the network communication among microservices. Strategies for the identification of services in non-object-oriented code-base (e.g., COBOL systems) is also of key concern [S8, S73]. Additionally, automated refactoring to microservices is essential [S22, S50] to reduce modernization costs/effort while maintaining the expected quality. Authors advocate for a framework to support the prioritization of services in the context of incremental migration [S74, S111]. This type of framework would particularly be useful in the migration of mission-critical systems where rapid migration is needed or in the situation of constrained resources [S107]. There is an increasing use of machine learning algorithms to identify microservices within monolithic applications. Specifically, a paper advocates for the use of clustering algorithms [S113], while the implementation of graph neural networks is recommended by other authors [S124]. Other pieces of work call for the exploration of alternative techniques for identifying microservices from monoliths, moving beyond their prior use of static analysis methods and search-based algorithms [S8, S11].

Designing New and Better Evaluation Metrics: There is an emphasis on introducing evaluation metrics for the thorough analysis of target architecture [S21, S111]. These metrics should seek to quantify the level of reuse among multiple services [S8] and provide means of assessing other decomposition methods, which is also related to the previous challenge [S111, S119]. This can particularly enable efficient planning of the migration journey. To continue with, legacy code-bases often exhibit poor quality and contain antipatterns. Thus, some authors emphasize the need to assess the impact these may have on training machine learning algorithms to aid in service identification [S41, S92]. Additionally, one paper highlights concerns regarding the effects that automatically generated services could have on the future operations of the entire architecture [S15]. Furthermore, other pieces of work address the challenges of preventing microservice antipatterns and accumulating technical debt during the modernization process [S15, S20], which also deserves more attention in future research.

Tooling Support for Visualization and Reconfiguration of Microservices: Two papers discuss the need for effective visualization of microservice architectures to bridge the gap between technical and business stakeholders, enabling better alignment and communication [S15, S50]. Similarly, a paper emphasizes the importance of packaging and deploying individual microservice units with

the flexibility for dynamic reconfiguration [S107]. Addressing these challenges requires robust tooling support that can integrate the requirements for both decomposition and adaptability. Thus, developing such tools can significantly enhance the design, deployment, and management of microservices, making this a critical area for research and innovation.

4.2 Architecture Redesign

Given the long lifespan of software in many industries, designing software with foresight is crucial. When the legacy system has a high business value, it is a candidate for modernization, independently of its internal quality. However, understanding and modernizing a legacy system with poor internal quality is a complex task. For example, systems usually evolve in space, adding new features, and time, with features being revised [65], which make their comprehension difficult. For such a situation, using refactoring strategies is a good way to improve the legacy internal quality to face the modernization [17]. Architectural redesign is important for effective software modernization, which is the focus of the studies in this section.

Driving Forces. There is a case of poorly designed software for certain bank's departments that led to *record duplication*, with each department assigning a different key to the same record [S13]. Similarly, a paper highlighted several reasons for architectural redesign, such as the use of multiple servers for data management combined with a rigid design structure, which led to *data duplication and inconsistencies* [S52]. Additionally, another study noted *duplicated business logic* in the codebase, without proper encapsulation, *hindering reusability* [S85]. A different paper used Amazon Simple Queue Service as an example to illustrate the issue of broad coupling of server functions, which *limits reuse* [S84]. Yet, to introduce *interoperability in industrial applications*, architectural refactoring was employed to expose existing functionalities through APIs [S121]. Furthermore, advancements in technology necessitate substantial investment in *enhancing the security* of existing systems. This is discussed in one of the papers under the concept of privacy-oriented software development [S10].

There are several additional examples in this category. A defense industry project had its legacy codebase refactored into a component-oriented architecture to *enable reusability* while building prototypes [S61]. Another interesting case involves introducing multi-tenancy in a legacy application solely through code refactoring, with the goal of *increasing profitability* through the **software as a service (SaaS)** model [S43]. To enhance software design with reactive programming, which *reduces complexity* of computations over data streams, static dataflow analysis-based refactoring was used to transform asynchronous code into reactive code [S55]. One study details a unique case of software refactoring aimed at changing data representation to *improve performance* and *data quality* [S81].

Research Opportunities. Architecture redesign is one of the strategies in which we found several research opportunities, mainly because it is strictly related to refactoring.

Improving General Architecture Recovery: One of the first steps for the redesign is to recover the architecture of the legacy system. Thus, architecture recovery is a prevalent challenge [27]. The authors of a paper propose, as part of their future work, a grounded-theory-based approach for recovering the architectures of large-scale legacy systems [S112]. Other researchers suggest the use of unsupervised learning techniques to assist in architectural refactoring [S76]. Additionally, a study highlights the need of establishing a baseline for evaluating architecture recovery approaches [S112].

Checking the Correctness of the New Architecture: While source-to-source transformations are used to support the migration of architectures, a call was made to additional techniques such as the automatic generation of unit tests to validate the correctness of the transformed code [S2, S123]. There are often negative effects on system performance when extensive refactoring is carried out

[S108]. Thus, it is worth researching how to ensure that the restructured code achieves the intended purpose without incurring significant performance costs. The area of code smell detection is not exempt from considerations during the redesign. It was mentioned as future work to incorporate detection mechanisms for more design flaws that exhibit bad smells during refactoring [S108]. Additionally, other authors intend to utilize previous code changes and class complexity to aid in the detection of code smells in refactored code [S2]. Also, a paper mentioned the opportunity of exploring the use of Large Language Models to test smell detection and removal [S7].

Improving Architectural Refactoring Tools: A paper highlights that there is a lack of tools for architecture-driven modernization in refactoring activities [S24]. As part of their future work, the authors propose that these tools should be improved to incorporate developer feedback, rather than relying solely on code smells and other code characteristics, as not all code smells necessarily trigger refactoring. Additionally, in the scenario of legacy systems with missing architecture documentation, tools that assist in recovering software architectures through direct code inspection would be highly beneficial [S112]. In a move to migrate skeletal implementation patterns in legacy Java code to Java 8 default methods, the authors intend to explore techniques such as Javadoc merging, transforming build scripts for migrations across modules, and using machine learning to disambiguate destination interfaces [S53]. This can be envisioned through proper tooling support and development. In addition to the primary sources, an additional research direction is to leverage Foundation Models (i.e., Large Language Models) in tasks related to code understanding [67] and refactoring [4] during the modernization.

4.3 New Programming Language

This strategy for modernization mainly refers to the switch between programming languages while modernizing legacy systems. The continuous evolving of some programming languages pushes other slow paced maintained languages into the legacy zone (e.g., due to outdated documentation or technology advancements [S34]). Thus, companies are forced to modernize their systems to use new programming language features. We recorded at least eight different instances of language switch from COBOL to JAVA and C#. Others include C++ to JAVA, C to C++, Java to C++ and C++ to Web Assembly. The main motivations for this strategy are discussed below.

Driving Forces. A number of the papers mentioned *reducing complexity* helps in better maintainability of the software as part of their driving forces when switching between different languages [S17, S48, S62, S97, S102, S103, S105]. For example, a group of authors transformed Oracle Forms to Java applications with the aim of decreasing the workload required to understand the structure of Oracle Forms [S97]. The next underlying reason is the *lack of support* for a given language or the underlying technology of a given system. For instance, after the suspension of PACBASE in 2015, there was a need for migrating from PACBASE code to plain COBOL [S23]. With respect to maintainability and lack of maintenance support, a financial institution in Japan re-engineered its COBOL based system to Java [S118]. Also, a group of authors stated that translation of Visual Basic to Python is relevant as financial industries are adopting AI solutions [S62]. Thus, modern technologies facilitate this modernization strategy as they *foster innovation*. Additionally, two other driving forces of migration to new languages are reduced build time [S115] and adoption of new technologies and growth [S78, S104].

Research Opportunities. There are three direction for future work in the modernization with new programming languages.

Supporting the Evaluation of Modern Applications: One study reported significant performance issues during the re-engineering of a legacy system in the financial industry, which left business stakeholders dissatisfied [S17]. Meanwhile, another paper intends to focus on evaluating the

performance of a system after its migration from Google Web Toolkit to the Angular framework [S115]. This work aims to leverage Angular-specific features, such as lazy loading and ahead-of-time compilation, to optimize performance. Thus, research on evaluating target systems is necessary to provide valuable insights for both researchers and practitioners, helping them assess the broader applicability of their methodologies.

Cost-Benefit Analysis during Migration Planning: To avoid delays in decision-making during modernization, a research opportunity is to quantify the cost-benefits of incremental migration [S118]. The authors plan to refine their analysis by incorporating dependencies, such as database access, to enhance accuracy. They also aim to extend their analysis to other target programming languages. Another challenge that needs to be addressed is comparing the complexity and quality of both the legacy and target systems based on a specific modernization strategy [S109].

Tools for Language Transformations: Similar to automatically generated code, there is a need for tools for the automatic generation of unit tests [S123]. Another propose future work which can complement language transformation tools is the fine-tuning of general Large Language Models for specification discovery from legacy systems [S62]. In terms of transformations between the same language, there is the need for enhanced code difference techniques and tools by incorporating semantic and structural characteristics of the code rather than just treating code as text [S23].

4.4 Reuse Optimization

The studies on this strategy focus mainly on SPL engineering, as a way of addressing the issues of opportunistic reuse (a.k.a., copy-and-paste or clone-and-own) [8]. Opportunistic reuse offers low initial costs and simplicity, but results in high maintenance costs due to the need to propagate bug fixes and feature updates across all clones [S59]. This makes the approach less sustainable in the long term. SPL engineering focuses on systematically deriving new systems from existing artifacts rather than developing them from scratch.

Driving Forces. The authors of a paper applied SPLs to a framework consisting of over 20 software artifacts with the primary goal to streamline the development of frequently used functionalities (e.g., security, file management, and business-related activities) and *reduce the time to market* by leveraging reuse opportunities [S87]. Similarly, a case study involving a seven-variant control unit software with over 13,000 lines of code highlighted the presence of two variants that were essentially clones of each other [S98]. This situation underscored the need to modernize the system and adopt a reuse strategy to *avoid duplication and inconsistency*. Another paper mentions the need of *managing features across different versions* of product variants as a driving force to adopt SPLs [S77]. The authors pointed out that locating feature revisions and composing variants with different feature revisions is essential for organizations adopting this strategy, as it *enables better management* and *leverages reuse opportunities* across product lines.

Research Opportunities. There are three research opportunities for the modernization of legacy system with focus on optimizing the reuse among family of software products:

Providing Cost-Benefit Estimation Models: It is usually risky switching to a new development paradigm, such as reuse-oriented software development, as this may lead to incurring severe cost. Hence, there is a call for cost estimations to predict the benefits of alternate development approach, especially in the extractive approach for SPLs [S58]. Similarly, the use of decision models or substantial empirical data to assess the cost and benefits of alternative re-engineering methods is worthy of research [S57].

Improving Variability Modeling and Analysis with Business-Related Concerns: One study highlights the need for inculcating business constraints to aid in defining products that match current market needs [S87]. Another paper, which addressed the need to generate core requirements for reuse,

seeks to include the dependencies and constraints between variable parts to improve the reusability of the core requirements (i.e., features of high value to the business) [S94].

Creating Tools for SPL Extraction: One of the challenges we noticed is related to the difficulty in finding appropriate tools for SPL extraction, which resulted in incomplete and inconsistent feature extractions, difficulties in managing feature dependencies, and issues with feature constraints [S80]. Another paper recommends tool development to facilitate multistaged product configuration, focusing on providing the necessary infrastructure and resources to manage complex configurations [S87].

4.5 New Hardware Integration

Studies on this strategy pertain to the modernization of software in response to changes in underlying hardware. Recently, certain low-level software computation and memory protection logics have been offloaded to new multi-core CPU architectures [S66]. As these multi-core platforms have become more widespread, new programming patterns have also been introduced to ensure efficient utilization of their computational power [S37]. This serves as the foundation for modernizing software based on hardware advancements, facilitating the ability to meet new business requirements [S38].

Driving Forces. In the realm of artificial neural network computation, **computing-in-memory (CIM)** architectures utilizing Non-volatile Memories aim to solve the "memory wall" issue that affects traditional Von Neumann architectures. As a result, legacy systems must be re-engineered to effectively leverage these advanced architectures [S47]. Additionally, the concept of reconfigurable hardware is emerging as a major driver for transitioning existing software to heterogeneous reconfigurable architectures, offering *better flexibility and efficiency* [S18]. The automotive industry is experiencing a shift toward multi-core processors, which necessitates the modernization of legacy software to fully harness the computational power of these processors to *improve performance* [S66]. Moreover, with the development of the **logical execution time (LET)** programming model, this approach has become a key paradigm for migrating legacy single-core applications to multi-core environments [S96].

Research Opportunities. In the analysis of the primary sources of our study, we identified two challenges that should be addressed with further research to support modernization when the goal is to integrate new hardware.

Application of Sophisticate Evaluation Metrics: Identifying potential solutions (migrated versions) is both time-consuming and critical to the migration process. One paper highlights the importance of performing a tradeoff analysis when selecting a solution from a set of potential options. Stakeholders must be cautious, as optimizing for one requirement may negatively impact others [S100]. A related direction for future research is the mechanical verification of whether the modeling constraints are sufficiently strong to ensure the preservation of sequential program semantics after parallelization [S37].

Creating Tools to Support Program Transformation: In the process of designing a compilation tool based on CIM architecture for code transformation, the authors of a paper observed that their original algorithm, particularly in areas like graph processing and genome analysis, often required additional transformations. These transformations were necessary to convert the problem into formats suitable for operations such as Matrix-Vector Multiplication, Matrix-Matrix Multiplication, or bitmap logical operations [S47]. Another paper mentions that while manual optimizations could yield further improvements, they would necessitate significant code rewrites to align with their design methodologies [S18]. However, this would compromise the portability of the software across different devices and platforms, while also significantly increasing the effort required for testing

and verification at every possible deployment stage. Another challenge pointed in one paper is the way dealing with is the optimization of memory allocation in multi-core LET-based system [S96]. A similar challenge raise is about resource conflicts, race conditions, task synchronizations, guaranteed execution time, and freedom from interference [S66]. Given the complexity of the programming paradigms of these new pieces of hardware, new tools are required to assist engineers when transforming the legacy systems to accommodate such hardware.

4.6 Leverage Automation

This migration strategy focuses on easing the development and deployment process through the introduction of DevOps in the maintenance of legacy system [S3]. The time spent on software maintenance could be reduced if manual tasks are automated. Automatic capabilities could be applied to tuning system configurations to optimize system performance and reduce human intervention [S64].

Driving Forces. One paper emphasized the integration of DevOps techniques into legacy system management, noting that adopting DevOps can significantly lead to both *faster time to market* and *reduce operational cost* [S3]. However, as maintenance and operational costs continue to rise even with DevOps practices, another study suggested the adoption of autonomic computing techniques to further reduce these expenses for *better flexibility and efficiency* [S64]. Additionally, we came across an innovative approach in another paper, which proposed applying automatic transformation techniques for cross-platform development [S12]. This method has the potential to simplify the handling of device-specific characteristics, ultimately reducing development cost and complexity [S3, S12, S64, S116].

Research Opportunities. We found out two directions for future work on this strategy:

Taking Advantage of Autonomic Computing Strategies: Autonomic computing systems can self-manage and adapt with minimal human intervention. One paper highlights that incorporating autonomic computing capabilities into existing software presents significant opportunities [S64]. By developing frameworks and techniques for integrating these capabilities in a cost-effective manner, organizations can enhance system reliability, reduce operational overhead, and improve responsiveness to business needs.

Providing Tools for Automation of Development: There is a need for sophisticated refactoring tools that can effectively manage legacy codebases while ensuring alignment with modern practices such as continuous integration/continuous deployment and containerization [S3]. These tools should automate the refactoring process, helping to maintain code quality and streamline deployments, ultimately reducing technical debt.

4.7 Database Modernization

This strategy refers to modernization activities related to databases. Database modernization encompasses the adoption of new database-related technologies and the integration of data from multiple sources to enhance performance and scalability. This is a crucial activity for efficient continuity of business operations, mostly in cases when businesses merge or acquired [S93].

Driving Forces. The main driving forces for the modernization of databases is to optimize data integration strategies, especially for the digitalization of legacy systems, enabling effective communication and data exchange across diverse platforms [S125]. Moreover, we noted that during business acquisitions and mergers, efficient data migration becomes crucial for maintaining operational efficiency and ensuring *interoperability among systems* [S93]. One paper mentioned that the issues associated with horizontally scaling traditional relational databases have also prompted a

shift toward more flexible and scalable alternatives, such as NoSQL databases, which can better accommodate the demands of modern data-intensive applications [S19] for *better flexibility and efficiency* [S125].

Research Opportunities. Among the set of primary sources, we identified two research opportunities for the challenges of this strategy.

Data Migration between Databases: Research into efficient data migration between relational databases and NoSQL systems should be a high priority, especially given the increasing adoption of NoSQL databases. While some authors have addressed aspects of this issue [S19], extensive work is still needed to explore additional techniques and compare them in terms of security, performance, and implementation feasibility [S110].

Preparing Databases for Systems Interoperability: In the context of Industry 4.0, achieving seamless interoperability among different systems requires a unified and consistent database infrastructure. Thus, researchers must focus on creating platforms or infrastructures that support data interoperability [S125]. In this context, it can also become handy to recover models from database schemas, although this has been attempted with the use of model-driven reverse engineering [S95], it is worth extending the previous implementation or introduce novel approaches.

4.8 Digital Transformation

Digital transformation is currently a trend and is receiving great attention around the world. For example, the European Union has the Digital Europe Programme,[6] Australia has the Digital Economy Strategy,[7] in North America there are the Canada Digital Adoption Program[8] and the Digital Strategy[9] of the United States. Despite expected benefits, the digital transformation is hampered by legacy systems [15]. In this context, modernization is a means to leverage the digital transformation [15, 60].

For this strategy, we found studies related to modernization toward Industry 4.0. In the advent of new technologies and new requirements in the field of IT and Operational Technology [S35], industries need to integrate existing components with modern components due to interoperability among modern components. Thus, it is necessary for industries to adopt modern technologies to remain competitive in the long term [S75].

Driving Forces. One paper proposes a layer to integrate to the legacy devices, enabling *interoperability among systems*, mapping field device data into an existing information model [S35]. Another study seeks to *improve performance, integration, and the overall streamlining* of processes, safety, and system interoperability for legacy systems [S75]. The goal is to provide an economical solution by allowing the continued use of existing facility devices while transitioning to Industry 4.0. Also, for another group of authors, modernization of legacy systems is key to *remaining competitive* and find *new business opportunities* [S114].

Research Opportunities. Given that we found only three papers addressing modernization for digital transformation, then we observed mainly two research directions.

Using Digital Twins to Analyze the Modernizations: In one paper, the authors proposed using digital twins to enable the adaptation of legacy systems, intending to enable the simulation of the system's behavior across various operating conditions [S75]. This can give insights into performance optimization without the necessity of physical prototyping or testing.

[6] https://digital-strategy.ec.europa.eu/en/activities/digital-programme
[7] https://digitaleconomy.pmc.gov.au/
[8] https://www.ic.gc.ca/eic/site/152.nsf/eng/home
[9] https://www.state.gov/digital-government-strategy/

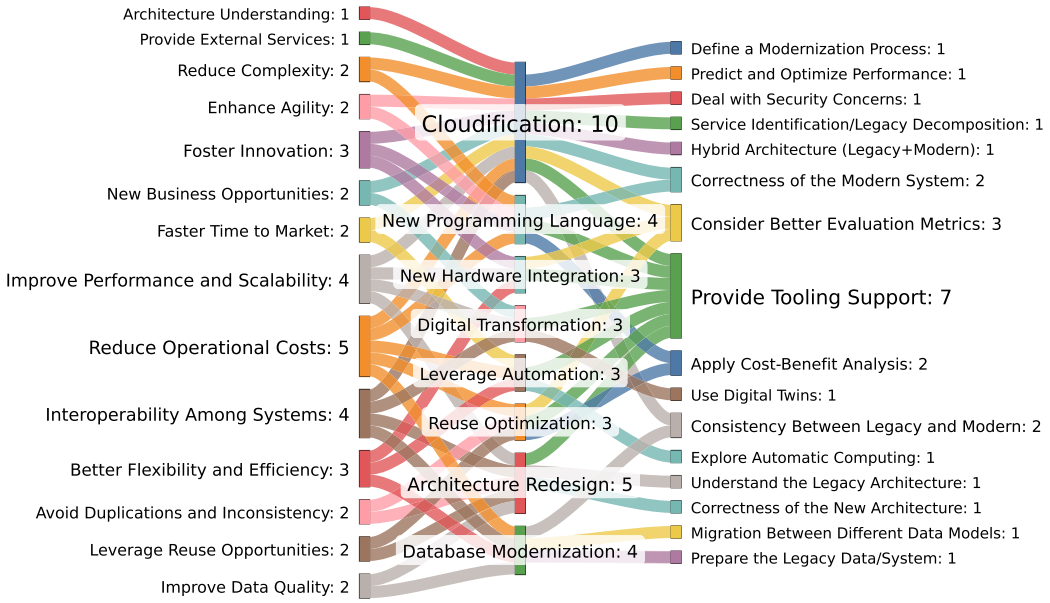


Fig. 3. Relations among modernization driving forces (left), strategies (middle), and challenges (right). The relationships between strategies and challenges define the 27 research opportunities described in Section 4.

Automated Tooling Support for Reconfiguration: A challenge pointed out by a study is that to leverage interoperability, it is necessary to detect changes and reconfigure factory floor devices. Thus, there is a need to further advance tools that allow devices to self-organize themselves using techniques like automated scaling and software defined networks [S35].

5 Discussions

Figure 3 presents a summary of the results with the relations between driving forces and modernization strategies, as well as between modernization strategies and challenges. This figure also allows us to analyze which driving forces and challenges are common for different modernization strategies. Figure 3, together with Tables 2, 3, and 4, and the results presented above; are the basis to answer the three RQs of our study.

RQ1: (From>To Strategies) What are the Contemporaneous Strategies to Modernize Legacy Systems? Our study identified eight modernization strategies (see Table 2) described in existing literature, namely modernization with a focus on: cloudification, architecture redesign, new programming language, reuse optimization, new hardware integration, leverage automation, database modernization, and digital transformation. Cloudification has the most number of studies (41, see Table 2), which we subcategorize into simple transition to the cloud, adoption of SOA, and migration to microservices. According to Figure 3, we can see the decision to embrace cloudification based on several driving forces (10), followed by architecture redesign (five driving forces) and new programming language (four driving forces). All strategies have been explored over the last 10 years (see column pub. Trends in Table 2), with some differences for database modernization and digital transformation. While the former has studies almost every other year, the latter has been more explored in the last year.

RQ2: (Driving Forces) What Are the Common Driving Forces for Modernizing a Legacy System? Our study identified 14 distinct driving forces (see Table 3) among the 126 primary sources. These driving forces are grouped in three main categories. First, seven *Technical* driving forces: architecture understanding, avoid duplications and inconsistency, improve data quality, improve performance and scalability, interoperability among systems, leverage reuse opportunities, and reduce complexity. Second, we observed three *Operational* driving forces: better flexibility and efficiency, enhance agility, and reduce operational costs. Third, there are four *Organizational* driving forces: faster time to market, foster innovation, new business opportunities, and provide external services. Among those, reduce operational costs (related to five strategies), improve performance/scalability (four), and interoperability among systems (four) are the most common across different modernization strategies (see the number of relationships between strategies and driving forces in Figure 3). This is a logical result, as the demand for software increases on a daily basis, and new pieces of software are developed, performance, interoperability, and operational costs are critical concerns.

RQ3: (Challenges) What Are the Challenges and Research Opportunities in the Field of Software Modernization? Despite many papers addressing software modernization, there are several open challenges and research gaps. Based on them, we collected 16 challenges that motivate 27 research opportunities for future work (see the relations between strategies and challenges in Figure 3). By far, the most prevalent opportunity is the development of tooling support (mentioned in 45 papers in Table 4) for seven different types of strategies (see Figure 3). Then, define a modernization process (in 21 studies), consider better evaluation metrics (in 20 studies), explore automatic computing, and prepare the legacy data/system (mentioned in 11 studies each) are the most commonly mentioned challenges. For evaluation metrics, the papers describe the need for considering different sources of information (e.g., databases and documentation). The research opportunity related to the correctness of modern systems (mentioned in nine papers) refers to avoiding transferring problems from the legacy system during modernization, such as antipatterns. Future work on cost-benefit analysis (mentioned in eight papers) should focus on to what extent a certain modernization strategy is beneficial for the company's context.

5.1 Threats to Validity

In this section, we discuss the main threats to validity related to our study and present how we mitigated them based on [7, 21, 87].

Construct Validity. Utilizing an RR protocol [22] can bring different biases to the study due to the possible limitations compared to an SLR [56]. These limitations include a limited number of sources used (DLs), only one reviewer analyzing the papers (selection bias), and a lack of quality assessment questions (quality bias) [21]. To mitigate these threats, when planning the protocol for our RR, we incorporated recommendations from the SLR protocol. For example, RRs commonly use only one source to retrieve the papers. In our case, we considered five distinct DLs (as described in Section 3). This led to a total of 3,460 studies retrieved (a similar number found in SMSs or SLRs). Furthermore, to avoid selection bias, all papers were analyzed by two of the authors, with a third author resolving conflicts when required. Lastly, although we did not use quality assessment metrics we mitigated the problem of low-quality or incomplete studies being analyzed by only considering published work in peer-reviewed venues (e.g., no arXiv-only studies were included) where the number of pages was greater than five for double-column papers and ten for single-column papers. These two criteria helped us avoid including vision, incomplete, or low-quality studies in our result analysis.

Internal Validity. Another potential threat is the limited number of studies analyzed, which may not fully represent the broader field of software modernization. To address this, we mitigated the

issue by selecting and analyzing a substantial number of studies (126 primary sources). Compared to other related reviews (see Section 6), our analysis includes studies from different domains of software modernization (e.g., cloud, microservices, and SPLs), enhancing the comprehensiveness of our review and broadening its scope in comparison to others.

External Validity. One potential external threat relates to the coherence of our results. We based our RR protocol on established guidelines [22], even taking recommendations from SLR guidelines [56]. We argue that this led to satisfactory outcomes in answering our RQs. When analyzing the data, we aimed for a broad perspective that includes both academic and practical viewpoints. Another threat involves incorrectly identifying challenges and research opportunities (RQ3). To mitigate this, we extracted limitations and future work highlighted by authors in each study and examined their coverage across the other studies. We also noted common future work, such as the development of supporting tools for managing the modernization across different modernization strategies.

Conclusion Validity. Researchers are often unaware of their own biases during the analysis and classification of studies, which can negatively impact the results of an RR. Additionally, data extraction inaccuracies pose another threat. To mitigate these threats, two authors independently conducted the selection and data extraction of the studies. In case of discrepancies, a third author was consulted to resolve them. Lastly, the "fishing and error rate" issue could influence the conclusions. This potential threat was mitigated by forming conclusions and answering RQs only after a full analysis of all primary studies.

6 Related Work

There has been a focus on software modernization in the past years, with different methodologies being employed to collect and analyze the state of the art and the practice [28, 32, 47, 88]. In this section, we describe how related work has explored different faces of software modernization from both the literature and industrial perspectives, while still falling short on covering the aspects that we present in our work.

6.1 The State of the Art

Since cloudification is prevalent in the practice of software engineering, the study by Zhao and Zhou [89] investigates the migration of legacy systems to the cloud, addressing both the challenges and benefits of such migration. The authors analyze existing literature and industrial applications, classifying migration methods into three strategies based on cloud service models: infrastructure as a service, platform as a service, and SaaS. Besides the work by Zhao and Zhou, other studies have also focused on migration to the cloud [32, 42, 47, 50, 69]. More recently, Hajlaoui et al. [45] investigated how migration to a cloud platform influences technical and business challenges that arise, particularly for **small and medium-sized enterprises (SMEs)**. Their study highlights the complexity and cost of migrating applications between cloud providers, emphasizing the difficulty of selecting the best migration strategy, even with available guidelines. The work by Abdellatif et al. [2] focuses on service identification approaches for migrating legacy systems to SOA. The authors conducted an SLR analyzing 41 approaches, categorizing them based on inputs, processes, outputs, and usability. This SLR was also validated by industry experts, highlighting that most approaches are still in the early stages.

Still related to cloudification of legacy systems, one of the most extensively explored topics is the migration to microservices [2, 12, 14, 64]. For instance, Wolfart et al. [88] highlight the complexity of the modernization process, emphasizing the lack of comprehensive studies covering real-world scenarios that include organizational, operational, and technical factors. While the roadmap in

their work focuses specifically on microservices, our work presents a more diverse analysis of modernization trends. Another study focusing on microservices maps the entire modernization process, covering both technical and systemic organizational changes [64]. The authors employ a grounded-theory approach, analyzing 19 interviews and 215 Stack Overflow discussions to identify two modes of change, 14 activities, and 53 solution outcomes. They emphasize that successful migration requires both short- and long-term perspectives, addressing technical and business considerations.

Despite the major focus on cloud modernization, other aspects of software are also considered. The work by Fritzsch et al. [39] explores the challenges of modernizing legacy systems in the domain of **cyber-physical systems (CPS)**, focusing on analyzing how microservices and DevOps, widely studied in enterprise systems, apply to CPS. By conducting a RR of 146 scientific papers and validating findings through interviews with nine CPS professionals from Siemens AG, the study identifies the challenges and practices relevant to CPS modernization, comparing them to those in enterprise applications. The findings reveal that, while some CPS-specific differences exist, many challenges and practices are shared across both domains. Similarly, Jatain and Gaur [51] focus on the re-engineering of object-oriented legacy systems, exploring the benefits of component-based systems over traditional object-oriented approaches.

Among the other subdomains of software modernization, re-engineering legacy software to SPL aims at maximizing reuse while minimizing maintenance costs [61]. In this context, the work by Assunção et al. [8] explored how re-engineering can be adopted to modernize legacy systems, considering the process, tools, and strategies used. Other approaches have also explored the processes of modernizing for different contexts, such as considering the implementation phases [13, 81] and how to rank different software modernization solutions based on the predetermined evaluation criteria [52].

In comparison to the aforementioned related work, our study aims to provide an understanding of modernization trends across a wide range of systems, highlighting key challenges that span multiple domains, including enterprise systems, cloud environments, transition between programming languages, new hardware devices, and beyond.

6.2 The State of Practice

Software modernization is a topic closely related to industry. Thus, we also consider the several industrial-focused studies contributing to the topic [14, 58]. Khadka et al. [53] investigate the practitioners' perceptions of legacy systems and their modernization, involving interviews with 26 practitioners and a survey with 198 respondents. The authors conclude that modernization challenges are not solely technical, but also encompass business and organizational dimensions. Similarly, the work by Abdellatif et al. [1] focuses on migrating legacy systems to SOA, with a specific emphasis on the challenges of identifying services within the target application. In a survey with 45 industry practitioners, the study reveals key drivers and methods for service identification, including the frequent use of domain knowledge and source code. Additionally, the study highlights the limited automation of service identification and offers recommendations and best practices for the migration process.

When focusing on migration to the cloud, the work by Aubé and Polacsek [11] investigates high-level requirements driving cloudification, based on insights from a qualitative study involving cloud migration experts. The study identifies 11 key requirements that influence design decisions during migrations, alongside two classifications aimed at guiding the elicitation and analysis of these requirements. For the modernization with microservices [40], the work by Di Francesco et al. [34] investigates the challenges faced by companies migrating to microservices. It reports findings from an empirical study based on interviews and questionnaires with practitioners. Once

again, the study is not limited to technical problems but also organizational challenges during the migration. In the same direction, Carvalho et al. [24] explore the criteria practitioners use for extracting microservices during system migrations, reporting findings from an exploratory survey of 15 specialists. The results highlight the importance of using multiple criteria (e.g., coupling and cohesion) for effective microservice identification. This study also emphasizes a gap between academic methods and practical needs, particularly the inadequacy of current tooling support. Another work investigates best practices, challenges, and solutions in microservice-based application development [86]. Their investigation focuses on "mature" teams with over two years of experience. The results, obtained from interviews and surveys, reveal that practitioners often deviate from standard recommendations, such as splitting services by business capabilities, highlighting the importance of robust infrastructural support.

Although these pieces of work bring several contributions to the understanding of practical aspects, they are limited to a few modernization strategies. In our study, we have not included surveys or interviews with practitioners, but our results can serve as a starting point for understanding current practices in software modernization and for validating them with practitioners in future work.

7 Conclusions

Software modernization is a fundamental activity of software engineering, since requirements inevitably change, technologies advance, and new business models emerge. Despite that, research on this topic has not followed modern software development, and legacy systems still remain a problem. To fill this gap and to spark research on this topic, we present an RR on software modernization in light of contemporary software modernization. We reviewed 126 papers, of which we identified eight contemporaneous strategies, 14 different driving forces, and 27 research opportunities (based on 16 challenges) to be considered.

Our study relied on an RR to understand the existing literature, focusing on the practical concerns of software modernization. Future work can investigate different aspects and perform more comparative and complementary analysis among the primary sources by applying other methodologies such as SLR or systematic mapping. For instance, a future study can focus on identifying the steps performed in the studies to conduct the modernization process, collecting what are the main artifacts used as input and produced as output for different strategies, and exploring how the studies evaluate their proposed methods. Furthermore, research on software modernization based on surveys and interviews with practitioners (as discussed in Section 6) can enrich the understanding of the practices and challenges faced in the industry.

In addition, and based on our experience in the field of software modernization, we also envisage some research directions not found in our RR. For example, in the background, we present five quadrants (i.e., replace, maintain, evolve, re-engineer, or migrate) of the portfolio analysis to support companies in deciding which modernization strategy to adopt. Choosing how to modernize a legacy system is a multi-criteria decision, so companies need solutions to help them address this challenge. To this end, we expect future work to propose recommendation approaches to support decision-making in terms of modernization options. Another direction of research is about non-intrusive migration approaches. Practitioners usually have preferences for using certain technologies, tools, and workflows. Based on that, researchers should propose modernization approaches and tools that take these preferences into account. Non-intrusive approaches are easier to transfer to practice [30]. Finally, the existing literature indicates that some software engineering activities should be performed differently in the context of SMEs [31, 77]. This might also be the case for software modernization [6]. Hence, research needs to be conducted on challenges faced by SMEs when modernizing their legacy systems in order to grow and expand their business.

References

[1] Manel Abdellatif, Geoffrey Hecht, Hafedh Mili, Ghizlane Elboussaidi, Naouel Moha, Anas Shatnawi, Jean Privat, and Yann-Gaël Guéhéneuc. 2018. State of the practice in service identification for SOA migration in industry. In *Service-Oriented Computing*. Claus Pahl, Maja Vukovic, Jianwei Yin, and Qi Yu (Eds.), Springer, Cham, 634–650.

[2] Manel Abdellatif, Anas Shatnawi, Hafedh Mili, Naouel Moha, Ghizlane El Boussaidi, Geoffrey Hecht, Jean Privat, and Yann-Gaël Guéhéneuc. 2021. A taxonomy of service identification approaches for legacy software systems modernization. *Journal of Systems and Software* 173 (2021). DOI: https://doi.org/10.1016/j.jss.2020.110868

[3] Shivali Agarwal, Sridhar Chimalakonda, Saravanan Krishnan, Vini Kanvar, and Samveg Shah. 2024. Tutorial report on legacy software modernization: A journey from non-AI to generative AI approaches. In *17th Innovations in Software Engineering Conference (ISEC '24)*. ACM, Article 19, 3 pages. DOI: https://doi.org/10.1145/3641399.3641434

[4] Eman Abdullah AlOmar, Anushkrishna Venkatakrishnan, Mohamed Wiem Mkaouer, Christian D. Newman, and Ali Ouni. 2024. How to refactor this code? An exploratory study on developer-ChatGPT refactoring conversations. arXiv:2402.06013 Retrieved from https://arxiv.org/abs/2402.06013

[5] Anas M. R. AlSobeh and Aws A. Magableh. 2018. An aspect-oriented with BIP components for better crosscutting concerns modernization in IOT applications. In *CS & IT Conference Proceedings*, Vol. 8, CS & IT Conference Proceedings, 21–31.

[6] B. Althani, S. Khaddaj, and B. Makoond. 2016. A quality assured framework for cloud adaptation and modernization of enterprise applications. In *IEEE International Conference on Computational Science and Engineering (CSE '16) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC '16) and 15th International Symposium on Distributed Computing and Applications for Business Engineering (DCABES '16)*, 634–637. DOI: https://doi.org/10.1109/CSE-EUC-DCABES.2016.251

[7] Apostolos Ampatzoglou, Stamatia Bibi, Paris Avgeriou, Marijn Verbeek, and Alexander Chatzigeorgiou. 2019. Identifying, categorizing and mitigating threats to validity in software engineering secondary studies. *Information and Software Technology* 106 (2019), 201–230.

[8] Wesley K. Assunção, Roberto E. Lopez-Herrejon, Lukas Linsbauer, Silvia R. Vergilio, and Alexander Egyed. 2017. Reengineering legacy applications Into software product lines: A systematic mapping. *Empirical Software Engineering* 22, 6 (Dec. 2017), 2972–3016. DOI: https://doi.org/10.1007/s10664-017-9499-z

[9] Wesley K. G. Assunção, Jacob Krüger, and Willian D. F. Mendonça. 2020. Variability management meets microservices: Six challenges of re-engineering microservice-based webshops. In *24th ACM Conference on Systems and Software Product Line: Volume A*. ACM, Article 22, 6 pages. DOI: https://doi.org/10.1145/3382025.3414942

[10] Wesley K. G. Assunção, Luciano Marchezan, Alexander Egyed, and Rudolf Ramler. 2024. Contemporary software modernization: Perspectives and challenges to deal with legacy systems. In *International Workshop on Software Engineering in 2030*. Retrieved from https://arxiv.org/abs/2407.04017

[11] Antoine Aubé and Thomas Polacsek. 2023. Cloud migration high-level requirements. In *Research Challenges in Information Science: Information Science and the Connected World*. Selmin Nurcan, Andreas L. Opdahl, Haralambos Mouratidis, and Aggeliki Tsohou (Eds.), Springer, 19–34.

[12] Deepali Bajaj, Urmil Bharti, Anita Goel, and S. C. Gupta. 2021. A prescriptive model for migration to microservices based on SDLC artifacts. *Journal of Web Engineering* 20, 3 (2021), 817–852. DOI: https://doi.org/10.13052/jwe1540-9589.20312

[13] Humairath K. M. Abu Bakar, Rozilawati Razali, and Dian Indrayani Jambari. 2019. Implementation phases in modernisation of legacy systems. In *2019 6th International Conference on Research and Innovation in Information Systems (ICRIIS '19)*, 1–6. DOI: https://doi.org/10.1109/ICRIIS48246.2019.9073628

[14] Armin Balalaie, Abbas Heydarnoori, Pooyan Jamshidi, Damian A. Tamburri, and Theo Lynn. 2018. Microservices migration patterns. *Software: Practice and Experience* 48, 11 (2018), 2019–2042. DOI: https://doi.org/10.1002/spe.2608

[15] David Beach. 2019. Legacy systems push up costs of digital transformation. Retrieved from https://www.theglobaltreasurer.com/2018/09/27/legacy-systems-push-up-costs-of-digital-transformation/

[16] K. Bennett. 1995. Legacy systems: Coping with success. *IEEE Software* 12, 1 (Jan. 1995), 19–23. DOI: https://doi.org/10.1109/52.363157

[17] Ana Carla Bibiano, Anderson Uchôa, Wesley K. G. Assunção, Daniel Tenório, Thelma E. Colanzi, Silvia Regina Vergilio, and Alessandro Garcia. 2023. Composite refactoring: Representations, characteristics and effects on software projects. *Information and Software Technology* 156 (2023), 107134.

[18] Nagendra Bommadevara, Andrea Del Miglio, and Steve Jansen. 2018. Cloud adoption to accelerate IT modernization. Digitial McKinsey: Insights.

[19] Georg Buchgeher, Rudolf Ramler, Heinz Stummer, and Hannes Kaufmann. 2021. Adopting microservices for industrial control systems: A five step migration path. In *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA '21)*. IEEE, 1–8.

[20] Coral Calero, Félix O. García, Gabriel Alberto García-Mireles, M. Ángeles Moraga, and Aurora Vizcaíno. 2024. Addressing sustainability-in software challenges. In *International Workshop on Software Engineering in 2030*. Retrieved from https://arxiv.org/abs/2406.07380

[21] Bruno Cartaxo, Gustavo Pinto, Baldoíno Fonseca, Márcio Ribeiro, Pedro Pinheiro, Maria Teresa Baldassarre, and Sérgio Soares. 2019. Software engineering research community viewpoints on rapid reviews. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '19)*, 1–12. DOI: https://doi.org/10.1109/ESEM.2019.8870144

[22] Bruno Cartaxo, Gustavo Pinto, and Sergio Soares. 2020. Rapid reviews in software engineering. In *Contemporary Empirical Methods in Software Engineering*. Felderer Michael, and Travassos Guilherme Horta (Eds.), Springer, Cham, 357–384. DOI: https://doi.org/10.1007/978-3-030-32489-6_13

[23] Luiz Carvalho, Alessandro Garcia, Wesley K. G. Assunção, Rafael de Mello, and Maria Julia de Lima. 2019. Analysis of the criteria adopted in industry to extract microservices. In *7th International Workshop on Conducting Empirical Studies in Industry and 6th International Workshop on Software Engineering Research and Industrial Practice*. IEEE, 22–29.

[24] Luiz Carvalho, Alessandro Garcia, Wesley K. G. Assunção, Rafael de Mello, and Maria Julia de Lima. 2019. Analysis of the criteria adopted in industry to extract microservices. In *IEEE/ACM Joint 7th International Workshop on Conducting Empirical Studies in Industry (CESI '19) and 6th International Workshop on Software Engineering Research and Industrial Practice (SER & IP '19)*, 22–29. DOI: https://doi.org/10.1109/CESSER-IP.2019.00012

[25] R. Cherinka, S. Foote, J. Burgo, and J. Prezzama. 2022. The impact of agile methods and "DevOps" on Day 2+ operations for large enterprises. In *Intelligent Computing*. Kohei Arai (Ed.), Springer, Cham, 1068–1081.

[26] Chia-Chu Chiang and Coskun Bayrak. 2006. Legacy software modernization. In *IEEE International Conference on Systems, Man and Cybernetics*, Vol. 2, 1304–1309. DOI: https://doi.org/10.1109/ICSMC.2006.384895

[27] E. J. Chikofsky and J. H. Cross. 1990. Reverse engineering and design recovery: A taxonomy. *IEEE Software* 7, 1 (1990), 13–17.

[28] Anita Choudhary, Mahesh Govil, Girdhari Singh, Lalit Awasthi, Emmanuel Pilli, and Divya Kapil. 2017. A critical survey of live virtual machine migration techniques. *Journal of Cloud Computing* 6 (Nov. 2017), 23. DOI: https://doi.org/10.1186/s13677-017-0092-1

[29] Comella-Dorda, Wallnau, Seacord, and Robert. 2000. A survey of black-box modernization approaches for information systems. In *International Conference on Software Maintenance*, 173–183. DOI: https://doi.org/10.1109/ICSM.2000.883039

[30] Caio H. Costa, Paulo H. M. Maia, Nabor C. Mendonça, and Lincoln S. Rocha. 2016. Supporting partial database migration to the cloud using non-intrusive software adaptations: An experience report. In *Advances in Service-Oriented and Cloud Computing*. Springer, Cham, 238–248.

[31] Ivonei Freitas da Silva, Paulo Anselmo da Mota Silveira Neto, Pádraig O'Leary, Eduardo Santana de Almeida, and Silvio Romero de Lemos Meira. 2014. Software product line scoping and requirements engineering in a small and medium-sized enterprise: An industrial case study. *Journal of Systems and Software* 88 (2014), 189–206. DOI: https://doi.org/10.1016/j.jss.2013.10.040

[32] Antonio Carlos Marcelino de Paula and Glauco de Figueiredo de Carneiro. 2016. A systematic literature review on Cloud computing adoption and migration. In *Evaluation of Novel Approaches to Software Engineering*. Leszek A. Maciaszek and Joaquim Filipe (Eds.), Springer International Publishing, Cham, 222–243.

[33] Paolo Di Francesco, Patricia Lago, and Ivano Malavolta. 2018. Migrating towards microservice architectures: An industrial survey. In *International Conference on Software Architecture*. IEEE, 29–2909.

[34] Paolo Di Francesco, Patricia Lago, and Ivano Malavolta. 2018. Migrating towards microservice architectures: An industrial survey. In *2018 IEEE International Conference on Software Architecture (ICSA '18)*, 29–2909. DOI: https://doi.org/10.1109/ICSA.2018.00012

[35] Christof Ebert and Carlos Henrique C. Duarte. 2018. Digital transformation. *IEEE Software* 35, 4 (2018), 16–21.

[36] Bassey Asuquo Ekanem and Evans Woherem. 2016. Dealing with components reusability issues as cutting-edge applications turn legacy. In *SAI Computing Conference (SAI '16)*. IEEE, 1190–1198. DOI: https://doi.org/10.1109/SAI.2016.7556129

[37] European Commission. 2020. Shaping the digital transformation in Europe. Retrieved from https://www.ospi.es/export/sites/ospi/documents/documentos/Sstudy_Shaping_the_digital_transformation_in_Europe_Final_report_202009.pdf

[38] Timothy C. Fanelli, Scott C. Simons, and Sean Banerjee. 2016. A systematic framework for modernizing legacy application systems. In *23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER '16)*, Vol. 1, IEEE, 678–682.

[39] Jonas Fritzsch, Justus Bogner, Markus Haug, Ana Cristina Franco da Silva, Carolin Rubner, Matthias Saft, Horst Sauer, and Stefan Wagner. 2023. Adopting microservices and DevOps in the cyber-physical systems domain: A rapid review and case study. *Software: Practice and Experience* 53, 3 (2023), 790–810.

[40] Jonas Fritzsch, Justus Bogner, Stefan Wagner, and Alfred Zimmermann. 2019. Microservices migration in industry: Intentions, strategies, and challenges. In *IEEE International Conference on Software Maintenance and Evolution (ICSME '19)*. IEEE, 481–490.

[41] Lidia Fuentes. 2024. Engineering software for next-generation networks in a sustainable Way. In *International Workshop on Software Engineering in 2030*. Retrieved from https://riuma.uma.es/xmlui/handle/10630/31573

[42] Mahdi Fahmideh Gholami, Farhad Daneshgar, Ghassan Beydoun, and Fethi Rabhi. 2017. Challenges in migrating legacy software systems to the cloud — An empirical study. *Information Systems* 67 (2017), 100–113. DOI: https://doi.org/10.1016/j.is.2017.03.008

[43] Noopur Goel (Ed.). 2015. Legacy systems towards aspect-oriented systems. In *Achieving Enterprise Agility through Innovative Software Development*. IGI Global, 262–286.

[44] Wesley Klewerton Guez Assunção, Luciano Marchezan, Lawrence Arkoh, Alexander Egyed, and Rudolf Ramler. 2024. *Contemporary Software Modernization: Strategies, Driving Forces, and Research Opportunities - Supplementary Material*. DOI: https://doi.org/10.5281/zenodo.13866356

[45] Jaleleddine Hajlaoui, Zied Trifa, and Zaki Brahmi. 2022. Model based migration of cloud systems: Review and roadmap. In *Computational Science and Its Applications (ICCSA '22)*. Osvaldo Gervasi, Beniamino Murgante, Eligius M. T. Hendrix, David Taniar, and Bernady O. Apduhan (Eds.), Springer, Cham, 249–264.

[46] Mauricio Hidalgo, Hernán Astudillo, and Laura M. Castro. 2024. Challenges to use role playing in software engineering education: A rapid review. In *Applied Informatics*. Hector Florez and Marcelo Leon (Eds.), Springer Nature Switzerland, Cham, 245–260.

[47] Melanie Holloway, Ronny Hans, Amr Rizk, Marvin Dickhaus, Bastian Emondts, and Ralf Steinmetz. 2017. Cloud adoption in the spotlight – Empirical insights from German IT experts. In *America's Conference on Information Systems: A Tradition of Innovation (AMCIS '17)*, 1–10.

[48] Anca Daniela Ionita, Marin Litoiu, and Grace Lewis. 2012. *Migrating Legacy Applications: Challenges in Service Oriented Architecture and Cloud Computing Environments* (1st. ed.). IGI Global.

[49] Alexandru F. Iosif-Lazar, Ahmad Salim Al-Sibahi, Aleksandar S. Dimovski, Juha Erik Savolainen, Krzysztof Sierszecki, and Andrzej Wasowski. 2015. Experiences from designing and validating a software modernization transformation (E). In *30th IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 597–607.

[50] Pooyan Jamshidi, Claus Pahl, Samuel Chinenyeze, and Xiaodong Liu. 2015. Cloud migration patterns: A multi-cloud service architecture perspective. In *12th International Conference on Service-Oriented Computing (ICSOC '15) 2014 Workshops*. F. Toumani, B. Pernici, D. Grigori, D. Benslimane, J. Mendling, N. B. HadjAlouane, B. Blake, O. Perrin, I. Saleh, and S. Bhiri (Eds.), Lecture Notes in Computer Science, Vol. 8954, 6–19. DOI: https://doi.org/10.1007/978-3-319-22885-3_2

[51] Aman Jatain and Deepti Gaur. 2015. Reengineering techniques for object oriented legacy systems. *International Journal of Software Engineering and its Applications* 9, 1 (2015), 35–51. DOI: https://doi.org/10.14257/ijseia.2015.9.1.04

[52] Nazean Jomhari, Nurul Aswani Ahmad Alias, Adi Aslah Abdul Ellah, Aws A. Magableh, and Ezlika Mohd Ghazali. 2024. A multi-criteria decision-making for legacy system modernization with FUCOM-WSM approach. *IEEE Access* 12 (2024), 48608–48619. DOI: https://doi.org/10.1109/ACCESS.2024.3383917

[53] Ravi Khadka, Belfrit V. Batlajery, Amir M. Saeidi, Slinger Jansen, and Jurriaan Hage. 2014. How do professionals perceive legacy systems and software modernization? In *36th International Conference on Software Engineering*, 36–47.

[54] Ravi Khadka, Prajan Shrestha, Bart Klein, Amir Saeidi, Jurriaan Hage, Slinger Jansen, Edwin van Dis, and Magiel Bruntink. 2015. Does software modernization deliver what it aimed for? A post modernization analysis of five software modernization case studies. In *International Conference on Software Maintenance and Evolution (ICSME '15)*. IEEE, 477–486.

[55] Alireza Khalilipour, Moharram Challenger, Mehmet Onat, Hale Gezgen, and Geylani Kardas. 2021. Refactoring legacy software for layer separation. *International Journal of Software Engineering and Knowledge Engineering* 31, 2 (2021), 217–247.

[56] Barbara Kitchenham, O. Pearl Brereton, David Budgen, Mark Turner, John Bailey, and Stephen Linkman. 2009. Systematic literature reviews in software engineering–A systematic literature review. *Information and Software Technology* 51, 1 (2009), 7–15.

[57] Holger Knoche and Wilhelm Hasselbring. 2018. Using microservices for legacy software modernization. *IEEE Software* 35, 3 (2018), 44–49. DOI: https://doi.org/10.1109/MS.2018.2141035

[58] Holger Knoche and Wilhelm Hasselbring. 2019. Drivers and barriers for microservice adoption – A survey among professionals in Germany. *Enterprise Modelling and Information Systems Architectures* 14 (2019). DOI: https://doi.org/10.18417/emisa.14.1

[59] Jacob Krüger, Wardah Mahmood, and Thorsten Berger. 2020. Promote-pl: A round-trip engineering process model for adopting and evolving product lines. In *24th ACM Conference on Systems and Software Product Line (SPLC '20)*. ACM, Article 2, 12 pages. DOI: https://doi.org/10.1145/3382025.3414970

[60] Pablo Luiz Leon and Flávio Eduardo Aoki Horita. 2021. On the modernization of systems for supporting digital transformation: A research agenda. In *XVII Brazilian Symposium on Information Systems*, 1–8.

[61] Luciano Marchezan, Elder Rodrigues, Wesley Klewerton Guez Assunção, Maicon Bernardino, Fábio Paulo Basso, and João Carbonell. 2022. Software product line scoping: A systematic literature review. *Journal of Systems and Software* 186 (2022), 111189. DOI: https://doi.org/10.1016/j.jss.2021.111189

[62] Robert Cecil Martin. 2003. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall PTR.

[63] Abir M'baya, Jannik Laval, and Nejib Moalla. 2017. An assessment conceptual framework for the modernization of legacy systems. In *11th International Conference on Software, Knowledge, Information Management and Applications*. IEEE, 1–11.

[64] Hamdy Michael Ayas, Philipp Leitner, and Regina Hebig. 2023. An empirical study of the systemic and technical migration towards microservices. *Empirical Software Engineering*. 28, 4 (May 2023). DOI: https://doi.org/10.1007/s10664-023-10308-9

[65] Gabriela K. Michelon, Wesley K. G. Assunção, David Obermann, Lukas Linsbauer, Paul Grünbacher, and Alexander Egyed. 2021. The life cycle of features in highly-configurable software systems evolving in space and time. In *20th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences*. ACM, 2–15.

[66] Richard Morris. 2021. Keeping old computers going costs government$2.3bn a year, says report. Retrieved from https://www.bbc.com/news/uk-politics-58085316

[67] Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. 2024. Using an llm to help with code understanding. In *IEEE/ACM 46th International Conference on Software Engineering (ICSE '24)*. IEEE, 881–881.

[68] C. D. Norton, C. Zuffada, OV Kalashnikova, and VK Decyk. 2008. Challenges in modernizing legacy scientific software. In *AGU Fall Meeting Abstracts*, Vol. 2008, IN11A–1016.

[69] Hongyu Pei Breivold. 2020. Towards factories of the future: Migration of industrial legacy automation systems in the Cloud computing and Internet-of-things context. *Enterprise Information Systems* 14, 4 (2020), 542–562. DOI: https://doi.org/10.1080/17517575.2018.1556814

[70] Dmitry Plekhanov, Henrik Franke, and Torbjørn H. Netland. 2023. Digital transformation: A review and research agenda. *European Management Journal* 41, 6 (2023), 821–844.

[71] Ricardo Pérez-Castillo, Manuel A. Serrano, and Mario Piattini. 2021. Software modernization to embrace quantum technology. *Advances in Engineering Software* 151 (2021), 102933. DOI: https://doi.org/10.1016/j.advengsoft.2020.102933

[72] Vinay T. R. and Ajeet A. Chikkamannur. 2016. A methodology for migration of software from single-core to multi-core machine. In *2016 International Conference on Computation System and Information Technology for Sustainable Solutions (CSITSS '16)*, 367–369. DOI: https://doi.org/10.1109/CSITSS.2016.7779388

[73] Jonas Åkesson, Sebastian Nilsson, Jacob Krüger, and Thorsten Berger. 2019. Migrating the Android apo-games Into an annotation-based software product line. In *23rd International Systems and Software Product Line Conference (SPLC '19)*. ACM, 103–107. DOI: https://doi.org/10.1145/3336294.3342362

[74] SAM Rizvi, Zeba Khanam, and Jamia Millia Islamia. 2010. A comparative study of using object oriented approach and aspect oriented approach for the evolution of legacy system. *International Journal of Computer Applications* 975 (2010), 8887.

[75] Paul Robertson. 1997. Integrating legacy systems with modern corporate applications. *Communications of the ACM* 40, 5 (May 1997), 39–46. DOI: https://doi.org/10.1145/253769.253785

[76] Shaik Mohammed Salman, Alessandro V. Papadopoulos, Saad Mubeen, and Thomas Nolte. 2021. A systematic methodology to migrate complex real-time software systems to multi-core platforms. *Journal of Systems Architecture* 117 (2021), 102087. DOI: https://doi.org/10.1016/j.sysarc.2021.102087

[77] Mary-Luz Sánchez-Gordón, Ricardo Colomo-Palacios, Antonio de Amescua Seco, and Rory V. O'Connor. 2016. *The Route to Software Process Improvement in Small- and Medium-Sized Enterprises*. Springer, Cham, 109–136. DOI: https://doi.org/10.1007/978-3-319-31545-4_7

[78] Robert Seacord, Santiago Comella-Dorda, Grace Lewis, Patrick Place, and Daniel Plakosh. 2001. *Legacy System Modernization Strategies*. Technical Report CMU/SEI-2001-TR-025. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. Retrieved from http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=5729

[79] Robert C. Seacord, Daniel Plakosh, and Grace A. Lewis. 2003. *Modernizing Legacy Systems: Software Technologies, Engineering Process and Business Practices*. Addison-Wesley.

[80] Hatef Shamshiri, Ashok Tripathi, Shola Oyedeji, and Jari Porras. 2024. Exploring the experiences of experts: Sustainability in agile software development – insights from the finnish software industry. In *International Workshop on Software Engineering in 2030*. Retrieved from https://arxiv.org/abs/2407.06978

[81] Stefan Strobl, Mario Bernhart, and Thomas Grechenig. 2020. Towards a topology for legacy system migration. In *IEEE/ACM 42nd International Conference on Software Engineering Workshops*, 586–594.

[82] Davide Taibi, Valentina Lenarduzzi, and Claus Pahl. 2017. Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation. *IEEE Cloud Computing* 4, 5 (2017), 22–32.

[83] Andrea C. Tricco, Etienne Langlois, Sharon E. Straus, and World Health Organization. 2017. *Rapid Reviews to Strengthen Health Policy and Systems: A Practical Guide*. World Health Organization.
[84] U.S. Government Accountability Office. 2019. Information technology: Agencies need to develop modernization plans for critical legacy systems. Retrieved from https://www.gao.gov/products/gao-19-471
[85] U.S. Government Accountability Office. 2023. Information technology: Agencies need to continue addressing critical legacy systems. Retrieved from https://www.gao.gov/products/gao-23-106821
[86] Yingying Wang, Harshavardhan Kadiyala, and Julia Rubin. 2021. Promises and challenges of microservices: An exploratory study. *Empirical Software Engineering* 26, 4 (May 2021). DOI: https://doi.org/10.1007/s10664-020-09910-y
[87] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in Software Engineering*. Vol. 1, Springer Science & Business Media.
[88] Daniele Wolfart, Wesley K. G. Assunção, Ivonei F. da Silva, Diogo C. P. Domingos, Ederson Schmeing, Guilherme L. Donin Villaca, and Diogo do N. Paza. 2021. Modernizing legacy systems with microservices: A roadmap. In *25th Evaluation and Assessment in Software Engineering (EASE '21)*. ACM, 149–159.
[89] Jun-Feng Zhao and Jian-Tao Zhou. 2014. Strategies and methods for cloud migration. *International Journal of Automation and Computing* 11, 2 (2014), 143–152. DOI: https://doi.org/10.1007/s11633-014-0776-7
[90] Xudong Zhao, Xiaolong Xu, Lianyong Qi, Xiaoyu Xia, Muhammad Bilal, Wenwen Gong, and Huaizhen Kou. 2024. Unraveling quantum computing system architectures: An extensive survey of cutting-edge paradigms. *Information and Software Technology* 167 (2024), 107380.

A Primary Sources

Primary References

[S1] Manel Abdellatif, Rafik Tighilt, Naouel Moha, Hafedh Mili, Ghizlane El Boussaidi, Jean Privat, and Yann-Gaël Guéhéneuc. 2020. A type-sensitive service identification approach for legacy-to-SOA migration. In *Service-Oriented Computing*. Eleanna Kafeza, Boualem Benatallah, Fabio Martinelli, Hakim Hacid, Athman Bouguettaya, and Hamid Motahari (Eds.), Springer, Cham, 476–491.
[S2] Vahid Alizadeh, Marouane Kessentini, Mohamed Wiem Mkaouer, Mel Ocinneide, Ali Ouni, and Yuanfang Cai. 2020. An interactive and dynamic search-based approach to software refactoring recommendations. *IEEE Transactions on Software Engineering* 46, 9 (Sep 2020), 932–961.
[S3] Álax Alves and Carla Rocha. 2023. Assuring the evolvability of legacy systems in devops transformation/adoption: Insights of an experience report. In *Agile Methods*. Carla Rocha, Celio Santana Júnior, Fernando De Sá, and Tiago Silva da Silva (Eds.), Springer, Cham, 32–53.
[S4] Kijin An and Eli Tilevich. 2020. Client insourcing: Bringing ops in-house for seamless re-engineering of full-stack JavaScript applications. In *ACM Web Conference 2020 (WWW '20)*. ACM, New York, NY, 179–189.
[S5] Kevin Angstadt, Jean-Baptiste Jeannin, and Westley Weimer. 2020. Accelerating legacy string kernels via bounded automata learning. In *25th International Conference on Architectural Support for Programming Languages and Operating Systems*, 235–249.
[S6] Nicolas Anquetil, Anne Etien, Gaelle Andreo, and Stéphane Ducasse. 2019. Decomposing God classes at siemens. In *IEEE International Conference on Software Maintenance and Evolution (ICSME '19)*. IEEE, 169–180.
[S7] Manoel Aranda, Naelson Oliveira, Elvys Soares, Marcio Ribeiro, Davi Romao, Ullyanne Patriota, Rohit Gheyi, Emerson Souza, and Ivan Machado. 2024. A catalog of transformations to remove smells from natural language tests. In *28th International Conference on Evaluation and Assessment in Software Engineering (EASE '24)*, 7–16.
[S8] Wesley K. G. Assunção, Thelma Elita Colanzi, Luiz Carvalho, Alessandro Garcia, Juliana Alves Pereira, Maria Julia de Lima, and Carlos Lucena. 2022. Analysis of a many-objective optimization approach for identifying microservices from legacy systems. *Empirical Software Engineering* 27, 2 (2022), 1–31.
[S9] Armin Balalaie, Abbas Heydarnoori, and Pooyan Jamshidi. 2016. Migrating to cloud-native architectures using microservices: An experience report. In *Workshops of Advances in Service-Oriented and Cloud Computing (ESOCC '16)*. A Celesti and P Leitner (Eds.), Communications in Computer and Information Science, Vol. 567, Springer, 201–215.
[S10] Maria Teresa Baldassarre, Vita Santa Barletta, Danilo Caivano, and Michele Scalera. 2020. Integrating security and privacy in software development. *Software Quality Journal* 28, 3 (2020), 987–1018.
[S11] Cesar Batista, Bruno Proença, Everton Cavalcante, Thais Batista, Felipe Morais, and Henrique Medeiros. 2022. Towards a multi-tenant microservice architecture: An industrial experience. In *2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC '22)*. IEEE, 553–562.
[S12] Juliano Zanuzzio Blanco and Daniel Lucrédio. 2021. A holistic approach for cross-platform software development. *Journal of Systems and Software* 179 (2021), 110985.

[S13] Matthias Book, Simon Grapenthin, and Volker Gruhn. 2014. Value-based migration of legacy data structures. In *Software Quality. Model-Based Approaches for Advanced Software and Systems Engineering*. Dietmar Winkler, Stefan Biffl, and Johannes Bergsmann (Eds.), Springer, Cham, 115–134.

[S14] Kiana Busch, Norman Christopher Böwing, Simon Spinner, Qais Noorshams, and Michael Grötzner. 2021. An experience report on modernizing i/o configuration software. In *Software Architecture*. Stefan Biffl, Elena Navarro, Welf Löwe, Marjan Sirjani, Raffaela Mirandola, and Danny Weyns (Eds.), Springer, Cham, 260–276.

[S15] Luiz Carvalho, Thelma Elita Colanzi, Wesley K. G. Assunção, Alessandro Garcia, Juliana Alves Pereira, Marcos Kalinowski, Rafael Maiani de Mello, Maria Julia de Lima, and Carlos Lucena. 2024. On the usefulness of automatically generated microservice architectures. *IEEE Transactions on Software Engineering* 50, 3 (2024), 651–667.

[S16] Wanli Chang, Ran Wei, Shuai Zhao, Andy Wellings, Jim Woodcock, and Alan Burns. 2020. Development automation of real-time Java: Model-driven transformation and synthesis. *ACM Transactions on Embedded Computing Systems* 19, 5 (2020).

[S17] Prabhakar Cherukupalli and Y. Raghu Reddy. 2015. Reengineering enterprise wide legacy BFSI systems: Industrial case study. In *8th India Software Engineering Conference*, 40–49.

[S18] Cătălin Bogdan Ciobanu, Giulio Stramondo, Ana Lucia Varbanescu, Andreas Brokalakis, Antonis Nikitakis, Lorenzo Di Tucci, Marco Rabozzi, Luca Stornaiuolo, Marco Santambrogio, Grigorios Chrysos, et al. 2018. EXTRA: An open platform for reconfigurable architectures. In *ACM18th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS '18)*. ACM, New York, NY, 220–229.

[S19] Caio H. Costa, Paulo H. M. Maia, Nabor C. Mendonça, and Lincoln S. Rocha. 2016. Supporting partial database migration to the Cloud using Non-intrusive software adaptations: An experience report. In *Advances in Service-Oriented and Cloud Computing*, Antonio Celesti and Philipp Leitner (Eds.). Springer, Cham, 238–248.

[S20] Carlos Eduardo da Silva, Yan de Lima Justino, and Eiji Adachi. 2022. SPReaD: Service-oriented process for reengineering and DevOps: Developing microservices for a Brazilian State department of taxation. *Service-Oriented Computing & Applications* 16, 1 (Mar. 2022), 1–16.

[S21] Murilo Góes de Almeida and Edna Dias Canedo. 2022. The adoption of microservices architecture as a natural consequence of legacy system migration at police intelligence department. In *Computational Science and Its Applications (ICCSA '22)*. Osvaldo Gervasi, Beniamino Murgante, Eligius M. T. Hendrix, David Taniar, and Bernady O. Apduhan (Eds.), Springer, Cham, 354–369.

[S22] Mohammadhadi Dehghani, Shekoufeh Kolahdouz Rahimi, Massimo Tisi, and Dalila Tamzalit. 2022. Facilitating the migration to the microservice architecture via model-driven reverse engineering and reinforcement learning. *Software and Systems Modeling* 21 (Jun. 2022), 1–19.

[S23] Céline Deknop, Johan Fabry, Kim Mens, and Vadim Zaytsev. 2020. Improving a software modernisation process by differencing migration logs. In *Product-Focused Software Process Improvement*. Maurizio Morisio, Marco Torchiano, and Andreas Jedlitschka (Eds.), Springer, Cham, 270–286.

[S24] Rafael S. Durelli, Matheus C. Viana, André de S. Landi, Vinicius H. S. Durelli, Marcio E. Delamaro, and Valter V. de Camargo. 2017. Improving the structure of KDM instances via refactorings: An experimental study using KDM-RE. In *ACMXXXI Brazilian Symposium on Software Engineering*. ACM, 174–183.

[S25] Ural Erdemir and Feza Buzluca. 2014. A learning-based module extraction method for object-oriented systems. *Journal of Systems and Software* 97 (2014), 156–177.

[S26] Mahdi Fahmideh and Ghassan Beydoun. 2018. Reusing empirical knowledge during cloud computing adoption. *Journal of Systems and Software* 138 (2018), 124–157.

[S27] Mahdi Fahmideh, John Grundy, Ghassan Beydoun, Didar Zowghi, Willy Susilo, and Davoud Mougouei. 2022. A model-driven approach to reengineering processes in cloud computing. *Information and Software Technology* 144 (2022), 106795.

[S28] Michalis Famelis, Levi Lúcio, Gehan Selim, Alessio Di Sandro, Rick Salay, Marsha Chechik, James R. Cordy, Juergen Dingel, and Hans Vangheluwe. 2015. Migrating automotive product lines: A case study. In *8th International Conference on Theory and Practice of Model Transformations*. D. Kolovos and M. Wimmer (Eds.), Lecture Notes in Computer Science, Vol. 9152, Springer, 82–97.

[S29] Everton Mateus Fernandes and Thiago Schumacher Barcelos. 2019. Identifying success factors in a legacy systems reengineering project using agile methods. In *Agile Methods*. Paulo Meirelles, Maria Augusta Nelson, and Carla Rocha (Eds.), Springer, Cham, 101–110.

[S30] Carlos Javier Fernández Candel, Jesús García Molina, Francisco Javier Bermúdez Ruiz, Jose Ramón Hoyos Barceló, Diego Sevilla Ruiz, and Benito José Cuesta Viera. 2019. Developing a model-driven reengineering approach for migrating PL/SQL triggers to Java: A practical experience. *Journal of Systems and Software* 151 (2019), 38–64.

[S31] Frank Fowley, Divyaa Manimaran Elango, Hany Magar, and Claus Pahl. 2017. Software system migration to cloud-native architectures for SME-sized software vendors. In *Theory and Practice of Computer Science (SOFSEM '17)*.

Bernhard Steffen, Christel Baier, Mark van den Brand, Johann Eder, Mike Hinchey, and Tiziana Margaria (Eds.), Springer, Cham, 498–509.

[S32] Frank Fowley, Divyaa Manimaran Elango, Hany Magar, and Claus Pahl. 2018. The benefits of using experimental exploration for cloud migration analysis and planning. In *Cloud Computing and Service Science*. Donald Ferguson, Víctor Méndez Muñoz, Jorge Cardoso, Markus Helfert, and Claus Pahl (Eds.), Springer, Cham, 157–176.

[S33] Kelly Garcés, Rubby Casallas, Camilo Álvarez, Edgar Sandoval, Alejandro Salamanca, Fredy Viera, Fabián Melo, and Juan Manuel Soto. 2018. White-box modernization of legacy applications: The Oracle forms case study. *Computer Standards & Interfaces* 57 (2018), 110–122.

[S34] Laura García-Borgoñón, Miguel Angel Barcelona, Armando J. Egea, German Reyes, Alejandro Sainz-de-la maza, and Adolfo González-Uzabal. 2023. Lessons learned in model-based reverse engineering of large legacy systems. *35th International Conference on Advanced Information Systems Engineering (CAiSE)*. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 13901, 330–344.

[S35] Omid Givehchi, Klaus Landsdorf, Pieter Simoens, and Armando Walter Colombo. 2017. Interoperability for industrial cyber-physical systems: An approach for legacy systems. *IEEE Transactions on Industrial Informatics* 13, 6 (2017), 3370–3378.

[S36] Brice Govin, Nicolas Anquetil, Anne Etien, Stephane Ducasse, and Arnaud Monegier. 2017. How can we help software rearchitecting efforts? Study of an industrial case. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME '17)*. IEEE, 509–518.

[S37] Reiner Hähnle, Asmae Heydari Tabar, Arya Mazaheri, Mohammad Norouzi, Dominic Steinhöfel, and Felix Wolf. 2020. Safer parallelization. In *Leveraging Applications of Formal Methods, Verification and Validation: Engineering Principles*. Tiziana Margaria and Bernhard Steffen (Eds.), Springer, Cham, 117–137.

[S38] Daniel Hallmans, Kristan Sandström, Thomas Nolte, and Stig Larsson. 2015. A method and industrial case: Replacement of an FPGA component in a legacy control system. In *IEEE 13th International Conference on Industrial Informatics (INDIN '15)*. IEEE, 208–214.

[S39] Muhammad Hamza, Muhammad Azeem Akbar, and Kari Smolander. 2024. The Journey to serverless migration: An empirical analysis of intentions, strategies, and challenges. In *Product-Focused Software Process Improvement*. Regine Kadgien, Andreas Jedlitschka, Andrea Janes, Valentina Lenarduzzi, and Xiaozhou Li (Eds.), Springer Nature Switzerland, Cham, 100–115.

[S40] Tomomi Hatano and Akihiko Matsuo. 2017. Removing code clones from industrial systems using compiler directives. In *IEEE/ACM 25th International Conference on Program Comprehension (ICPC '17)*. IEEE, 336–345.

[S41] Hasan Emre Hayretci and Fatma Başak Aydemir. 2021. A multi case study on legacy system migration in the banking industry. *33rd International Conference on Advanced Information Systems Engineering (CAiSE)*. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 12751, 536–550.

[S42] Sebastian Heil, Valentin Siegert, and Martin Gaedke. 2018. ReWaMP: Rapid Web migration prototyping leveraging WebAssembly. In *Web Engineering*. Tommi Mikkonen, Ralf Klamma, and Juan Hernández (Eds.), Springer, Cham, 84–92.

[S43] Uwe Hohenstein and Preeti Koka. 2017. Enabling legacy applications for multi-tenancy without reengineering. In *Software Technologies*. Enrique Cabello, Jorge Cardoso, André Ludwig, Leszek A. Maciaszek, and Marten van Sinderen (Eds.), Springer, Cham, 284–308.

[S44] Abdul R. Hummaida, Norman W. Paton, and Rizos Sakellariou. 2023. A hierarchical decentralized architecture to enable adaptive scalable virtual machine migration. *Concurrency and Computation: Practice and Experience* 35, 2 (2023), e7487.

[S45] Alexandru F. Iosif-Lazar, Ahmad Salim Al-Sibahi, Aleksandar S. Dimovski, Juha Erik Savolainen, Krzysztof Sierszecki, and Andrzej Wasowski. 2015. Experiences from designing and validating a software modernization transformation (E). In *30th IEEE/ACM International Conference on Automated Software Engineering (ASE '15)*. IEEE, 597–607.

[S46] Luis Jiménez-Navajas, Ricardo Pérez-Castillo, and Mario Piattini. 2020. Reverse engineering of quantum programs toward KDM models. In *Quality of Information and Communications Technology*. Martin Shepperd, Fernando Brito e Abreu, Alberto Rodrigues da Silva, and Ricardo Pérez-Castillo (Eds.), Springer, Cham, 249–262.

[S47] Hai Jin, Bo Lei, Haikun Liu, Xiaofei Liao, Zhuohui Duan, Chencheng Ye, and Yu Zhang. 2023. A compilation Tool for computation offloading in ReRAM-based CIM architectures. *ACM Transactions on Architecture and Code Optimization* 20, 4 (Oct. 2023), 1–25.

[S48] Ivan Jovanovikj, Gregor Engels, Anthony Anjorin, and Stefan Sauer. 2018. Model-driven test case migration: The test case reengineering horseshoe model. In *Information Systems in the Big Data Era*. Jan Mendling and Haralambos Mouratidis (Eds.), Springer, Cham, 133–147.

[S49] Paulo Afonso Parreira Júnior, Rosângela Dellosso Penteado, Matheus Carvalho Viana, Rafael Serapilha Durelli, Valter Vieira de Camargo, and Heitor Augustus Xavier Costa. 2014. Reengineering of object-oriented software Into aspect-oriented ones supported by class models. In *Enterprise Information Systems*. Slimane Hammoudi, José Cordeiro, Leszek A. Maciaszek, and Joaquim Filipe (Eds.), Springer, Cham, 296–313.

[S50] Manabu Kamimura, Keisuke Yano, Tomomi Hatano, and Akihiko Matsuo. 2018. Extracting candidates of microservices from monolithic application code. In *25th Asia-Pacific Software Engineering Conference (APSEC '18)*. IEEE, 571–580.

[S51] Nader Kesserwan, Rachida Dssouli, and Jamal Bentahar. 2018. Modernization of legacy software tests to model-driven testing. *2nd International Conference on Emerging Technologies for Developing Countries (AFRICATEK)*. Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, 140–156.

[S52] Musawwer Khan, Islam Ali, Waqar Mehmood, Wasif Nisar, Waqar Aslam, Muhammad Shafiq, and Jin-Ghoo Choi. 2021. CMMI compliant modernization framework to transform legacy systems. *Intelligent Automation and Soft Computing* 27, 2 (2021), 311–331.

[S53] Raffi Khatchadourian. 2017. Automated refactoring of legacy Java software to enumerated types. *Automated Software Engineering* 24, 4 (2017), 757–787.

[S54] Juha Kinnunen. 2021. Leaving the lights on? Exploring cloud ERP migrations and IS discontinuance. In *2021 IEEE 12th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON '21)*, 0058–0064.

[S55] Mirko Köhler and Guido Salvaneschi. 2020. Automated refactoring to reactive programming. In *ACM 34th IEEE/ACM International Conference on Automated Software Engineering (ASE '19)*. IEEE Press, 835–846.

[S56] Harald Kornmayer and Abdallah Salama. 2017. AQUASI-an automated quality assurance application platform for SMEs in handcraft industries. In *2017 IEEE International Conference on Cognitive Computing (ICCC '17)*. IEEE, 80–87.

[S57] Jacob Krüger and Thorsten Berger. 2020. Activities and costs of Re-engineering cloned variants Into an integrated platform. In *14th International Working Conference on Variability Modelling of Software-Intensive Systems*, 1–10.

[S58] Jacob Krüger, Wolfram Fenske, Jens Meinicke, Thomas Leich, and Gunter Saake. 2016. Extracting software product lines: A cost estimation perspective. In *20th International Systems and Software Product Line Conference*, 354–361.

[S59] Jacob Krüger, Louis Nell, Wolfram Fenske, Gunter Saake, and Thomas Leich. 2017. Finding lost features in cloned systems. In *ACM 21st International Systems and Software Product Line Conference - Volume B (SPLC '17)*. ACM, New York, NY, 65–72.

[S60] Jacob Krüger, Marcus Pinnecke, Andy Kenner, Christopher Kruczek, Fabian Benduhn, Thomas Leich, and Gunter Saake. 2018. Composing annotations without regret? Practical experiences using FeatureC. *Software: Practice and Experience* 48, 3 (2018), 402–427.

[S61] Pierre Laborde, Steven Costiou, Éric Le Pors, and Alain Plantec. 2020. 15 years of reuse experience in evolutionary prototyping for the defense industry. In *Reuse in Emerging Software Engineering Practices*. Sihem Ben Sassi, Stéphane Ducasse, and Hafedh Mili (Eds.), Springer, Cham, 87–99.

[S62] Kevin Lano, Howard Haughton, Ziwen Yuan, and Hessa Abdulrahman Alfraihi. 2024. Agile model-driven re-engineering. *Innovations in Systems and Software Engineering* 20 (Jun. 2024), 1–26.

[S63] Neil Lapuz, Paul Clarke, and Yalemisew Abgaz. 2021. Digital transformation and the role of dynamic tooling in extracting microservices from existing software systems. In *Systems, Software and Services Process Improvement*. Murat Yilmaz, Paul Clarke, Richard Messnarz, and Michael Reiner (Eds.), Springer, Cham, 301–315.

[S64] Heng Li, Tse-Hsun Chen, Ahmed E. Hassan, Mohamed Nasser, and Parminder Flora. 2018. Adopting autonomic computing capabilities in existing large-scale systems: An industrial experience report. In *40th International Conference on Software Engineering: Software Engineering in Practice*, 1–10.

[S65] Marcos López-Sanz, Valeria de Castro, Esperanza Marcos, and Jorge Moratalla. 2017. Modernization of information systems at Red.Es: An approach based on Gap analysis and ADM. In *Service-Oriented Computing*. Michael Maximilien, Antonio Vallecillo, Jianmin Wang, and Marc Oriol (Eds.), Springer, Cham, 490–498.

[S66] Georg Macher, Andrea Höller, Eric Armengaud, and Christian Kreiner. 2015. Automotive embedded software: Migration challenges to multi-core computing platforms. In *IEEE 13th International Conference on Industrial Informatics (INDIN '15)*. IEEE, 1386–1393.

[S67] Muhammad Ghufron Mahfudhi and Teresa Galvão Dias. 2016. Specifying modernization Into service-oriented SaaS system in a case of public transport document generator. In *Exploring Services Science*. Theodor Borangiu, Monica Dragoicea, and Henriqueta Nóvoa (Eds.), Springer, Cham, 590–603.

[S68] Luis Márquez, David G. Rosado, Haralambos Mouratidis, Daniel Mellado, and Eduardo Fernández-Medina. 2015. A framework for secure migration processes of legacy systems to the cloud. In *Advanced Information Systems Engineering Workshops*. Anne Persson and Janis Stirna (Eds.), Springer, Cham, 507–517.

[S69] Atif Mashkoor, Felix Kossak, Miklós Biró, and Alexander Egyed. 2018. Model-driven re-engineering of a pressure sensing system: An experience report. In *Modelling Foundations and Applications*. Alfonso Pierantonio and Salvador Trujillo (Eds.), Springer, Cham, 264–278.

[S70] Ayman Massoud. 2015. A new framework for quality-based SOA migration. In *International Conference on Software Engineering Research and Practice (SERP '15),* 196.

[S71] Cristian Mateos, Marco Crasso, Juan M. Rodriguez, Alejandro Zunino, and Marcelo Campo. 2015. Measuring the impact of the approach to migration in the quality of Web service interfaces. *Enterprise Information Systems* 9, 1 (2015), 58–85.

[S72] Cristian Mateos, Alejandro Zunino, Sanjay Misra, Diego Anabalon, and Andres Flores. 2017. Migration from COBOL to SOA: Measuring the impact on web services interfaces complexity. In *Information and Software Technologies*. Robertas Damaševičius and Vilma Mikašytė (Eds.), Springer, Cham, 266–279.

[S73] Genc Mazlami, Jürgen Cito, and Philipp Leitner. 2017. Extraction of microservices from monolithic software architectures. In *IEEE International Conference on Web Services (ICWS '17)*. IEEE, 524–531.

[S74] Manuel Mazzara, Nicola Dragoni, Antonio Bucchiarone, Alberto Giaretta, Stephan T. Larsen, and Schahram Dustdar. 2021. Microservices: Migration of a mission critical system. *IEEE Transactions on Services Computing* 14, 5 (2021), 1464–1477.

[S75] Rafael Da Silva Mendonça, Marenice Melo De Carvalho, Guido Soprano Machado, Claudia Sabrina Monteiro Da Silva, Renan Landau Paiva De Medeiros, and Vicente Ferreira De Lucena. 2023. Development of a novel methodology to retrofit legacy systems in the context of Industry 4.0. *IEEE Access* 11 (2023), 123223–123235.

[S76] Fanyi Meng, Ying Wang, Chun Yong Chong, Hai Yu, and Zhiliang Zhu. 2024. Evolution-aware constraint derivation approach for software remodularization. *ACM Transactions on Software Engineering and Methodology* 33 (Jul. 2024), 1–43.

[S77] Gabriela Karoline Michelon, David Obermann, Wesley K. G. Assunção, Lukas Linsbauer, Paul Grünbacher, Stefan Fischer, Roberto E. Lopez-Herrejon, and Alexander Egyed. 2022. Evolving software system families in space and Time with feature revisions. *Empirical Softw. Engg.* 27, 5 (Sep. 2022).

[S78] Arjan J. Mooij, Gernot Eggen, Jozef Hooman, and Hans van Wezep. 2015. Cost-effective industrial software rejuvenation using domain-specific models. In *Theory and Practice of Model Transformations*. Dimitris Kolovos and Manuel Wimmer (Eds.), Springer, Cham, 66–81.

[S79] Arjan J. Mooij, Mabel M. Joy, Gernot Eggen, P. Janson, and A. Rădulescu. 2016. Industrial software rejuvenation using open-source parsers. In *9th International Conference on Theory and Practice of Model Transformations*. P. VanGorp and G. Engels (Eds.), Lecture Notes in Computer Science, Vol. 9765, Springer, 157–172.

[S80] Rodrigo André Ferreira Moreira, Wesley K. G. Assunção, Jabier Martinez, and Eduardo Figueiredo. 2022. Open-source software product line extraction processes: The ArgoUML-SPL and phaser cases. *Empirical Software Engineering* 27, 4 (2022), 85.

[S81] Krishna Narasimhan, Christoph Reichenbach, and Julia Lawall. 2017. Interactive data representation migration: Exploiting program dependence to aid program transformation. In *ACM2017 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation (PEPM '17)*. ACM, New York, NY, 47–58.

[S82] Michael Nieke, Adrian Hoff, Ina Schaefer, and Christoph Seidl. 2022. Experiences with constructing and evolving a Software product line with delta-oriented programming. In *ACM16th International Working Conference on Variability Modelling of Software-Intensive Systems (VaMoS '22)*. ACM, New York, NY, 1–9.

[S83] Ali Ouni, Marouane Kessentini, Houari Sahraoui, Katsuro Inoue, and Kalyanmoy Deb. 2016. Multi-criteria code refactoring using search-based software engineering: An industrial case study. *ACM Transactions on Software Engineering and Methodology* 25, 3 (Jun 2016.), 1–53.

[S84] Ali Ouni, Hanzhang Wang, Marouane Kessentini, Salah Bouktif, and Katsuro Inoue. 2018. A hybrid approach for improving the design quality of Web service interfaces. *ACM Transactions on Internet Technology* 19, 1 (Dec. 2018), 1–24.

[S85] Ozan Özkan, Önder Babur, and Mark Brand. 2023. Refactoring with domain-driven design in an industrial context: An action research report. *Empirical Software Engineering* 28 (Jun. 2023), 1–36.

[S86] Fatih Öztürk, Erdem Sarili, Hasan Sözer, and Bariş Aktemur. 2014. Effort estimation for architectural refactoring to introduce module isolation. In *Software Architecture*. Paris Avgeriou and Uwe Zdun (Eds.), Springer, Cham, 300–307.

[S87] Carlos Parra, Diego Joya, Leonardo Giral, and Alvaro Infante. 2014. An SOA approach for automating software product line adoption. In *29th Annual ACM Symposium on Applied Computing*, 1231–1238.

[S88] Ilaria Pigazzini, Francesca Arcelli Fontana, and Andrea Maggioni. 2019. Tool support for the migration to microservice architecture: An industrial case study. In *Software Architecture*. Tomas Bures, Laurence Duchien, and Paola Inverardi (Eds.), Springer, Cham, 247–263.

[S89] Konstantinos Plakidas, Daniel Schall, and Uwe Zdun. 2018. Model-based support for decision-making in architecture evolution of complex software systems. In *12th European Conference on Software Architecture: Companion Proceedings*, 1–7.

[S90] Peter Priller, Andreas Aldrian, and Thomas Ebner. 2014. Case study: From legacy to connectivity migrating industrial devices Into the world of smart services. In *2014 IEEE Emerging Technology and Factory Automation (ETFA '14)*. IEEE, 1–8.

[S91] Amit Rathee and Jitender Kumar Chhabra. 2020. Mining reusable software components from object-oriented source code using discrete PSO and modeling them as Java beans. *Information Systems Frontiers* 22, 6 (2020), 1519–1537.

[S92] Tanisha Rathod, Christina Terese Joseph, and John Paul Martin. 2023. Improving industry 4.0 readiness: Monolith application refactoring using graph attention networks. In *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing Workshops (CCGridW '23)*. IEEE, 223–230.

[S93] Maryam Razavian and Patricia Lago. 2014. A lean and mean strategy: A data migration industrial study. *Journal of Software: Evolution and Process* 26, 2 (2014), 141–171.

[S94] Iris Reinhartz-Berger and Mark Kemelman. 2020. Extracting core requirements for software product lines. *Requirements Engineering* 25, 1 (2020), 47–65.

[S95] André Reis and Alberto Rodrigues da Silva. 2017. XIS-reverse: A model-driven reverse engineering approach for legacy information systems. In *International Conference on Model-Driven Engineering and Software Development*, Vol. 2, SCITEPRESS, 196–207.

[S96] Stefan Resmerita, Andreas Naderlinger, and Stefan Lukesch. 2017. Efficient realization of logical execution times in legacy embedded software. In *15th ACM-IEEE International Conference on Formal Methods and Models for System Design*. ACM & IEEE, 37–46.

[S97] Cristo Rodríguez, Kelly Garcés, Jordi Cabot, Rubby Casallas, Fabián Melo, Daniel Escobar, and Alejandro Salamanca. 2021. Model-based assisted migration of Oracle forms applications: The overall process in an industrial setting. *Software - Practice and Experience* 51, 8 (2021), 1641–1675.

[S98] Kamil Rosiak, Oliver Urbaniak, Alexander Schlie, Christoph Seidl, and Ina Schaefer. 2019. Analyzing variability in 25 years of industrial legacy software: An experience report. In *23rd International Systems and Software Product Line Conference-Volume B*, 65–72.

[S99] Fco Javier Bermúdez Ruiz, ÓScar Sánchez Ramón, and Jesús García Molina. 2017. A tool to support the definition and enactment of model-driven migration processes. *Journal of Systems and Software* 128 (2017), 106–129.

[S100] Shaik Mohammed Salman, Alessandro V. Papadopoulos, Saad Mubeen, and Thomas Nolte. 2021. A systematic methodology to migrate complex real-time software systems to multi-core platforms. *Journal of Systems Architecture* 117 (2021), 102087.

[S101] Jesus Sanchez Cuadrado, Esther Guerra, and Juan de Lara. 2014. Reverse engineering of model transformations for reusability. D DiRuscio and D Varro (Eds.), In *Theory and Practice of Model Transformations (ICMT '14)*, Lecture Notes in Computer Science, Vol. 8568, 186–201.

[S102] Bruno Santos, Daniel San Martín, Raphael Honda, and Valter Vieira de Camargo. 2019. Concern metrics for modularity-oriented modernizations. In *Quality of Information and Communications Technology*. Mario Piattini, Paulo Rupino da Cunha, Ignacio García Rodríguez de Guzmán, and Ricardo Pérez-Castillo (Eds.), Springer, Cham, 225–238.

[S103] Markus Schnappinger and Jonathan Streit. 2021. Efficient platform migration of a mainframe legacy system using custom transpilation. In *IEEE International Conference on Software Maintenance and Evolution (ICSME '21)*. IEEE, 545–554.

[S104] Mathijs Schuts, Jozef Hooman, and Paul Tielemans. 2018. Industrial experience with the migration of legacy models using a DSL. In *Real World Domain Specific Languages Workshop (RWDSL '18)*, 1–10.

[S105] Mathijs T. W. Schuts, Rodin T. A. Aarssen, Paul M. Tielemans, and Jurgen J. Vinju. 2022. Large-scale semi-automated migration of legacy C/C++ Test code. *Software: Practice and Experience* 52, 7 (2022), 1543–1580.

[S106] Gehan M. K. Selim, Shige Wang, James R. Cordy, and Juergen Dingel. 2015. Model transformations for migrating legacy deployment models in the automotive industry. *Software & Systems Modeling* 14 (2015), 365–381.

[S107] Anfel Selmadji, Abdelhak-Djamel Seriai, Hinde Lilia Bouziane, Christophe Dony, and Rahina Oumarou Mahamane. 2018. Re-architecting OO software into microservices. In *Service-Oriented and Cloud Computing*. Kyriakos Kritikos, Pierluigi Plebani, and Flavio de Paoli (Eds.), Springer, Cham, 65–73.

[S108] Mohsin Shaikh and Chan-Gun Lee. 2016. Aspect oriented re-engineering of legacy software using cross-cutting concern characterization and significant code smells detection. *International Journal of Software Engineering and Knowledge Engineering* 26, 3 (2016), 513–536.

[S109] Harry Sneed and Chris Verhoef. 2019. Re-implementing a legacy system. *Journal of Systems and Software* 155 (2019), 162–184.

[S110] K. Subramani, Bugra Caskurlu, and Utku Umur Acikalin. 2020. Security-aware database migration planning. In *Algorithmic Aspects of Cloud Computing*. Ivona Brandic, Thiago A. L. Genez, Ilia Pietri, and Rizos Sakellariou (Eds.), Springer, Cham, 103–121.

[S111] Davide Taibi and Kari Systä. 2020. A decomposition and metric-based evaluation framework for microservices. In *Cloud Computing and Services Science*. Donald Ferguson, Víctor Méndez Muñnoz, Claus Pahl, and Markus Helfert (Eds.), Springer, Cham, 133–149.

[S112] Damian A. Tamburri and Rick Kazman. 2018. General methods for software architecture recovery: A potential approach and its evaluation. *Empirical Software Engineering* 23, 3 (Jun. 2018), 1457–1489.

[S113] Imen Trabelsi, Manel Abdellatif, Abdalgader Abubaker, Naouel Moha, Sébastien Mosser, Samira Ebrahimi-Kahou, and Yann-Gaël Guéhéneuc. 2023. From legacy to microservices: A type-based approach for microservices identification using machine learning and semantic analysis. *Journal of Software: Evolution and Process* 35, 10 (2023), 4213–4242.

[S114] André Luiz Alcântara Castilho Venâncio, Eduardo de Freitas Rocha Loures, Fernando Deschamps, Ricardo Alexandre Diogo, Alysson Felipe Lumikoski, and Neri dos Santos. 2021. Digital transformation framework for adequacy of maintenance systems for industry 4.0. *Communications in Computer and Information Science* 1407 CCIS (2021), 280–292.

[S115] Benoit Verhaeghe, Anas Shatnawi, Abderrahmane Seriai, Anne Etien, Nicolas Anquetil, Mustapha Derras, and Stephane Ducasse. 2022. A hybrid architecture for the incremental migration of a web front-end. In *17th International Conference on Software Technologies (ICSOFT '22)*. H. G. Fill, M. VanSinderen, and L. Maciaszek (Eds.), INSTICC, 101–110.

[S116] Kenny Wehling, David Wille, Christoph Seidl, and Ina Schaefer. 2018. Reducing variability of technically related software systems in large-scale IT landscapes. In *28th Annual International Conference on Computer Science and Software Engineering (CASCON '18)*. IBM Corp., 224–235.

[S117] Hendrik M. Würz, Michel Krämer, Marvin Kaster, and Arjan Kuijper. 2024. Migrating monolithic applications to function as a service. *Software: Practice and Experience* 54, 2 (2024), 149–167.

[S118] Kazuki Yokoi, Eunjong Choi, Norihiro Yoshida, Joji Okada, and Yoshiki Higo. 2023. Cost-benefit analysis for modernizing a large-scale industrial system. In *30th Asia-Pacific Software Engineering Conference (APSEC '23)*. IEEE, 441–449.

[S119] Pascal Zaragoza, Abdelhak-Djamel Seriai, Abderrahmane Seriai, Anas Shatnawi, Hinde-Lilia Bouziane, and Mustapha Derras. 2022. Materializing microservice-oriented architecture from monolithic object-oriented source code. In *Software Technologies*. Hans-Georg Fill, Marten van Sinderen, and Leszek A. Maciaszek (Eds.), Springer, Cham, 143–168.

[S120] Labib M. Zawra, Hala A. Mansour, and Nagy W. Messiha. 2019. Migration of legacy industrial automation systems in the context of industry 4.0-A comparative study. In *International Conference on Fourth Industrial Revolution (ICFIR '19)*. IEEE, 1–7.

[S121] Su Zhang, Hua-Qian Cai, Yun Ma, Tian-Yue Fan, Ying Zhang, and Gang Huang. 2020. Smartpipe: Towards interoperability of industrial applications via computational reflection. *Journal of Computer Science and Technology* 35 (2020), 161–178.

[S122] Yukun Zhang, Bo Liu, Liyun Dai, Kang Chen, and Xuelian Cao. 2020. Automated microservice identification in legacy systems with functional and non-functional metrics. In *IEEE International Conference on Software Architecture (ICSA '20)*. IEEE, 135–145.

[S123] Bo Zhao, Zhen Li, Ali Jannesari, Felix Wolf, and Weiguo Wu. 2015. Dependence-based code transformation for coarse-grained parallelism. In *International Workshop on Code Optimisation for Multi and Many Cores*, 1–10.

[S124] Teng Zhong, Yinglei Teng, Shijun Ma, Jiaxuan Chen, and Sicong Yu. 2023. A microservices identification method based on spectral clustering for industrial legacy systems. In *IEEE Globecom Workshops (GC Wkshps)*. IEEE, 1331–1337.

[S125] Aaron Zielstorff, Dirk Schöttke, Antonius Hohenhövel, Thomas Kämpfe, Stephan Schäfer, and Frank Schnicke. 2023. Harmonizing heterogeneity: A novel architecture for legacy system integration with digital twins in industry 4.0. *Communications in Computer and Information Science* 1886 CCIS (2023), 68–87.

[S126] Olaf Zimmermann. 2017. Architectural refactoring for the cloud: A decision-centric view on cloud migration. *Computing* 99, 2 (2017), 129–145.

Received 5 April 2024; revised 1 October 2024; accepted 15 November 2024