IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 23 April 2024, accepted 23 May 2024, date of publication 13 June 2024, date of current version 21 June 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3414377

EXPOSITION

Explaining the Business-Technological Age of Legacy Information Systems

SEBASTIAN ROSENKRANZ[1,2], DANIEL STAEGEMANN[1], MATTHIAS VOLK[1], AND KLAUS TUROWSKI[1]

[1]Magdeburg Research and Competence Cluster, Otto von Guericke University Magdeburg, 39106 Magdeburg, Germany
[2]Volkswagen Group, 38440 Wolfsburg, Germany

Corresponding authors: Sebastian Rosenkranz (sebastian3.rosenkranz@ovgu.de), Daniel Staegemann (daniel.staegemann@ovgu.de), Matthias Volk (matthias.volk@ovgu.de), and Klaus Turowski (klaus.turowski@ovgu.de)

ABSTRACT Aged information systems are commonly referred to as legacy information systems or just merely as legacy systems. Typically, these are mission-critical systems developed years ago that significantly resist evolution. Many organizations have lived with these systems for a long time and consider them a burden because they hold back their businesses and cause unreasonably high evolution and operating costs. However, they also cannot easily be discarded and replaced by modern solutions. Despite their relevance, no universally accepted concept exists that explains these systems or the mechanisms behind them. Accordingly, the properties and challenges associated with legacy systems vary significantly depending on their respective author and intentions. This needs to be improved, and this is also where our research comes in. To this end, we first describe the causes of information systems' aging, typical symptoms by which legacy systems are often recognized, and the consequences of using outdated solutions. Based on this empirical point of view, we introduce a holistic model to explain legacy systems objectively as well as the concept of *business-technological age* to distinguish between chronological age and actual obsolescence of a deployed system. This approach provides practitioners and researchers with a foundation for stringently explaining the hitherto fuzzy legacy phenomenon, promoting a common understanding of legacy systems, and simplifying communication by employing specific concepts. Practitioners can use this approach to better understand their stock application systems in terms of aging and improve their decision-making regarding evolution.

INDEX TERMS Legacy systems, legacy information systems, legacy software, legacy hardware, software aging, software erosion, software evolution, literature review.

I. INTRODUCTION

Legacy systems are widespread and are typically defined as *"large software systems that we don't know how to cope with but that are vital to our organization"* [1]. A vast number of studies show that the information system inventory of organizations in some Western countries ranges between around 39% [2] to around 50% [3], [4]. Therefore, a significant part of the IS in Western industrialized countries should be considered outdated. However, these systems are not only a widespread economic problem, but also a problem for society as a whole, as legacy systems are also a significant problem for public authorities [5], [6], [7], [8]. Finally, the legacy inventory is growing (and hence the associated problems) as past and current digitization initiatives are generating tomorrow's legacies on a large scale.

Legacy systems, once modern IS, evolve over many years, usually decades, into massive black boxes that are almost impossible to understand and that cannot be efficiently adapted to ever-changing requirements. As a result, they become a burden and an obstacle to the implementation of business strategies, [9], [10] since increasing amounts of resources must be devoted to their operation and further evolution[1] as such systems remain in service [5], [8], [12], [13], [14, p. 590]. Replacing them with a modern and economical

The associate editor coordinating the review of this manuscript and approving it for publication was Huiyan Zhang.

[1]The term 'evolution' is understood as a range of activities from small enhancements to complete re-implementation of information systems [11].

solution seems obvious given the significant effort involved. However, they usually cannot be easily discarded or replaced. Even a major restructuring is often not a trivial matter [1, 14, p. 590f, 15–17]. This kind of predicament for organizations is often referred to as *legacy dilemma.* Either they continue to evolve and operate their legacy systems despite the high effort and slow evolution, or they incur high costs and risks to replace them [1], [5], [14, p. 590f], [15], [18], and [19, p. 3].

However, research on legacy issues is by no means new. Indeed, research on software and system aging has been ongoing since the 1970s [20, p. 2f]. Over the past 50 years, much knowledge has been gained regarding outdated systems. Nevertheless, there is no single, commonly accepted definition and no comprehensive theory of legacy systems, only a variety of phenomena related to them [12], [21], [22, p. 33]. No approach provides a holistic understanding of these systems. Instead, legacy systems are usually only described in a fragmented or one-sided way, and only certain aspects that are relevant to the intentions of the respective author are highlighted. Accordingly, the understanding of these systems varies considerably [9], [21], [23]. However, a precise and objective comprehension of the state of IS is vital for executives, policymakers, and researchers in business, government, and academia, so that they can make better decisions. This is the subject of this article and we propose the following research questions (RQ):

RQ-1: How can the business-technological age of information systems be explained in general?

RQ-2: What are the factors that lead to the aging of IS?

RQ-3: Is the lifetime of IS related to obsolescence?

RQ-4: What are the symptoms of legacy systems?

This article is structured as follows: In section II, the underlying scientific approach of this paper is explained in detail. Section III explains the empirical characteristics attributed to legacy systems and the causes that contribute to the emergence of legacy systems. Section IV presents a theoretical model that integrates the apparent empirical properties into a consistent approach and thus resolves the apparent contradictions. This model makes it possible to explain the nature of legacy systems theoretically. It helps on the one hand in communication and on the other hand in the design and evolution of information systems by clearly identifying the drivers that lead to the development of legacy systems. The developed and presented contents are then discussed and the limits as well as the scientific and practical added value are highlighted.

II. METHODOLOGY

To address the RQs, we took a design science research (DSR) [24] oriented approach. Fig. 1 shows the six steps of the design science process: problem identification and motivation, objectives of the solution, design and development, demonstration, evaluation, and communication.

The first research activity consists of the problem definition; thus, the description of the problem field and business needs. Furthermore, the additional benefit of a solution to the addressed business need is justified, which should motivate the research task. In the second step, the objectives of the artifact are expressed in quantitative or qualitative terms. The third and fourth steps are to build the artifact according to the requirements and perform the demonstration. The preliminary outcome must be evaluated against the objectives that are initially raised. These tasks may be run several times to improve the artifact with each iteration until an appropriate level of quality is achieved. After a positive evaluation, the final artifact is presented to the community to initiate discourse, contribute the results to the knowledge base, and allow practitioners to gain an advantage for their organization [25]. The remainder of this section describes the implementation of the aforementioned research steps in this work.

Following this DSR approach, we define the detrimental effects of evolving and operating legacy systems as the relevant problem field. The associated evolutionary process, the system context (typically an organization) of the legacy system, and the current state of technology are included in this assessment. Such a holistic view is necessary because a legacy system is closely intertwined with the associated evolutionary engineering process and the organization it serves. The legacy phenomenon can only be explained accurately when all these aspects are considered [26].

We understand legacy systems as socio-technical systems and those as an instance of the Human Task Technology System Theory [27, p. 17ff]. Accordingly, legacy systems are aged IS. Fig. 2 visualizes this theory and its system elements: *human*, *task*, and *technology*. Since we consider legacy systems as IS,[2] this paper focuses on the underlying basic sciences of business information systems, thus, on information technology and business management, and less on the psychological, social, legal, and other aspects that also influence such systems [30, p. 19ff].

In the context of IS, we regard information as an immaterial business resource detached from its material foundations. According to Wiener: *"Information is information, neither matter nor energy"* [31, p. 166], we abstract from physical foundations, although information is always bound to a carrier of material, energy, or a combination of both [32, p. 95]. Consequently, we neglect aging from an energetic point of view, such as the excessive power consumption of old hardware, or material wear and tear, such as occurs in the use of hard disks or electrolytic capacitors. Therefore, we will occasionally touch on these aspects, but will not go into great depth.

In considering the business need of the our research approach, we have chosen the following research

[2]Sometimes, especially in older literature, also referred to as 'business information technology systems' [28], or 'computerized information systems.' [29].

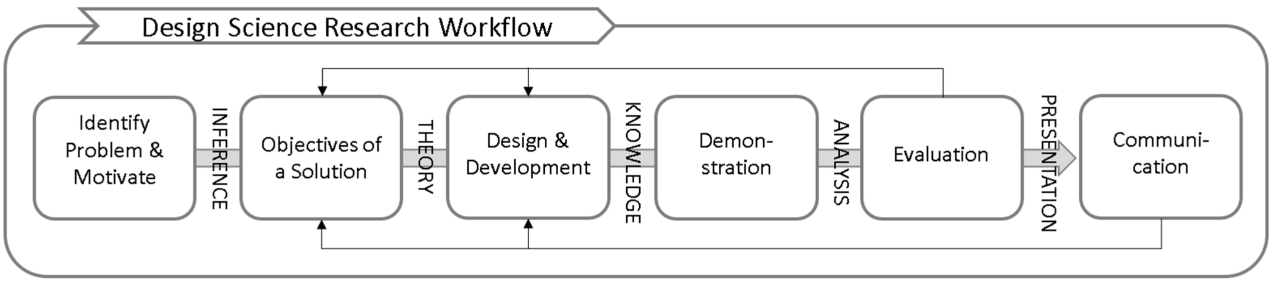


FIGURE 1. The design science research workflow consists of six research tasks that build on each other.

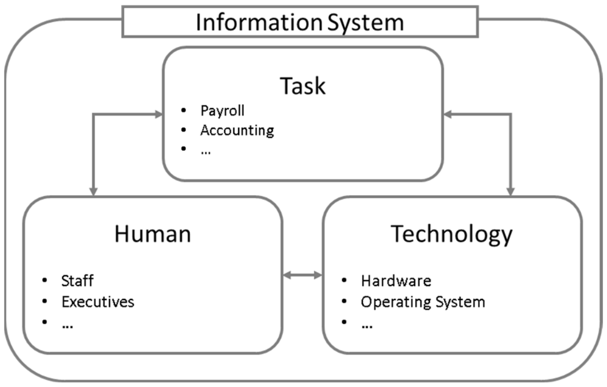


FIGURE 2. An information system can be explained by the human task technology systems theory.

goal – *enable people who are developing new or evolving existing IS to understand the impact of obsolescence on those systems, including its causes and consequences, to make better decisions.* Therefore, we seek an easy-to-understand model that objectively explains IS and their obsolescence. In doing so, we strictly separate and implement the following aspects in the model: (1) the factors that cause or influence obsolescence, (2) the characteristics and symptoms that make the state of obsolescence objectively assessable, and (3) the consequences that result from using such solutions.

After addressing the problem field and objectives of the artifact, the third step in the DSR workflow is to construct the artifact that addresses the business need. For this purpose, we have conducted an extensive review of the literature to understand the subject of this research and to identify the empirical characteristics of legacies. The result of our literature review, which consists of the empirically proven characteristics and statements related to legacy systems, is presented in section three.

In section four, we elaborate on this artifact, which is a model that explains legacy systems and their aging drivers. After introducing the model and the legacy status explained by the model, we perform a demonstration by mapping the assessed empirical legacy system properties to the elements of the proposed model, based on a deductive approach. The subsequent evaluation was performed using argumentative-deductive analysis [33]. This provides answers to the previously stated research questions and discusses the practical utility of the artifact. Finally, section five concludes the work and discusses possible limitations as well as potential avenues for further research.

Finally, the document at hand is a descriptive (also known as positive or non-normative) research paper and, accordingly, no normative statements will be made in the process, such as defining thresholds or the extent to which certain actions should be taken. Since this is a descriptive research paper, the focus is on the acquisition of knowledge, which means the description and subsequent explanation of the object of study. The former was done through an extensive review of the literature and the latter through the proposed artifact.

III. LITERATURE REVIEW

It is generally recommended to provide a detailed description of a problem beforehand to effectively explain it later [27, p. 75ff]. Therefore, understanding legacy systems first requires a thorough consideration of their empirical nature, an in-depth knowledge of the causes of obsolescence, and the impact of their use on the organizations they serve. With these considerations in mind, we have reviewed the literature to build a sound empirical description of legacy systems. The following section presents the parameters, time intervals, and inclusion and exclusion criteria used in the literature review to make the process objectively understandable [34]. At the end of the third section, the results of the literature review are discussed and summarized.

A. LITERATURE REVIEW PROCESS

A reproducible documentation of the literature search is important to meet scientific rigor [34]. Accordingly, we describe the literature review in detail below. We conducted a structured literature review, as recommended by Levy et al. [35], and applied it to IEEE Xplore and ScienceDirect. IEEE Xplore was selected as it covers technical areas very well. ScienceDirect was added to extend the scope and to include any peripheral areas not yet covered. An initial query was performed with the following terms:[3]

- ‘legacy system’ and ‘legacy systems’

[3]The quotation marks are part of the keywords.

TABLE 1. Query and number of documents found.

Data base	Key word	Field	Hits
IEEE Xplore	legacy system	Document Title	82
	legacy systems	Document Title	187
	legacy information system	Document Title	3
	legacy information systems	Document Title	15
ScienceDirect	legacy system	'Title, abstract or author-specified keywords'	312
	legacy systems	'Title, abstract or author-specified keywords'	312, as above
	legacy information system	'Title, abstract or author-specified keywords'	17
	legacy information system	'Title, abstract or author-specified keywords'	17, as above
	Total number of documents found		616

- 'legacy information system' and 'legacy information systems'

This paper is about legacy information systems. Since both terms, legacy system and legacy information system, are often used synonymously in the literature, both expressions were chosen for the literature search. As summarized in Table 1, the search used IEEE Xplore's 'Document Title' field, resulting in 287 documents. In ScienceDirect, the search was conducted using the field 'Title, abstract or author-specified keywords' leading to another 329 hits. The query considered all papers published up to the end of 2023.

In the second phase, we first evaluated articles based on the abstract, the introduction, and (where applicable) the conclusion. We considered articles that dealt specifically with symptoms of legacy information systems, such as architecture degeneration, software erosion, obsolete hardware, or obsolete programming languages or engineering models. If an article describes the problems of legacy systems in detail, it was included. In contrast, articles were excluded in which legacy systems were merely used as a value-neutral reference to an earlier legacy system without addressing its characteristics. Articles that do not deal with legacy information systems, but with legacy systems of other domains, such as power grids, airplanes or highly integrated real-time systems were also excluded. Short papers were not explicitly excluded. In addition, the primary sources of the included articles were reviewed and considered if these articles were not found in the initial search. Moreover, a forward search was conducted to identify and include more recent literature and evidence where appropriate. The forward search was conducted with the help of Google Scholar[4] and Semantic Scholar.[5] The backward and forward searches have naturally added other literature databases and a broad amount of articles, so we believe that the knowledge base is largely covered, as required for high-quality literature reviews [36]. In addition to this, only using official publications at conferences, in journals and books guarantees a minimum level of quality through the review process associated with the publication. Therefore, only such quality-assured papers were considered. To summarize, the Fig. 3 visualizes the search process, Table 2 lists the Inclusion and Exclusion criteria.

The articles found and included in the way described above provide detailed characteristics about outdated systems.

[4]Available at: *https://scholar.google.de*.

[5]Available at: *https://www.sematicscholar.org*.

TABLE 2. Inclusion and exclusion criteria.

Inclusion Criteria	Exclusion Criteria
Published at conferences, in journals, or book chapters.	Not written in English or German.
Legacy systems are discussed in detail.	Legacy system is only used as a synonym for a previous system or inventory system.
Articles that can be assigned to the field of (business) information systems or business informatics.	Papers related to network engineering, cyber-physical systems, medicine, aircraft industry, and the grid were excluded.

B. LITERATURE REVIEW RESULTS

In the following Fig. 4, an overview according to Cooper's taxonomy is given to characterize the literature review [37]. The following TABLE 3 lists all articles found and included. Papers that refer exclusively to the methodology of this paper are not listed in this table. Some additional references that are not part of the literature review were also not listed in the table, but merely added as footnotes. The abbreviation 'FW' used in the 'Source' column stands for forward search and 'BW' for backward search. In addition, the number of citations for each paper was determined in March 2024 using Google Scholar. These can be found in the 'Citations' column.

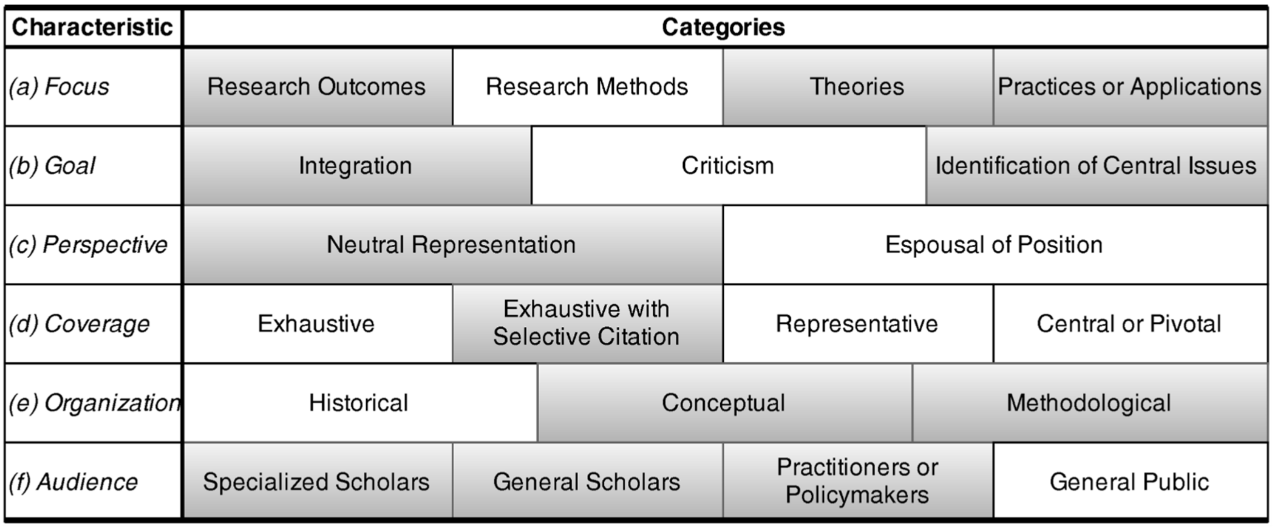
C. LEGACY SYSTEM

In the following sections, legacy systems are presented on the basis of their empirically perceptible characteristics and those identified through the literature review, focusing on applied information technology and the business model embodied by the legacy system. The structure follows the previously given elements of IS: technology, human, and business (cf. Fig. 2). The initial section 'An historical Perspective' begins with a short introduction and time-related characteristics of those systems. The section is summarized by a conclusion.

1) A HISTORICAL PERSPECTIVE

Organizations rarely embark on legacy system development [61]; rather, the original developers typically used the best solutions, technologies, and architectures available at that time [1], [26], [53], [58], [61], [70], [75]. Over time and after many changes, these modern information systems then turn into legacy systems [1].

"no one write [sic] a legacy system; rather they write a nice and pretty system, which latterly becomes old and ugly" [26].



Characteristic	Categories			
(a) Focus	Research Outcomes	Research Methods	Theories	Practices or Applications
(b) Goal	Integration	Criticism	Identification of Central Issues	
(c) Perspective	Neutral Representation	Espousal of Position		
(d) Coverage	Exhaustive	Exhaustive with Selective Citation	Representative	Central or Pivotal
(e) Organization	Historical	Conceptual	Methodological	
(f) Audience	Specialized Scholars	General Scholars	Practitioners or Policymakers	General Public

FIGURE 3. Adaption of the literature review by cooper.

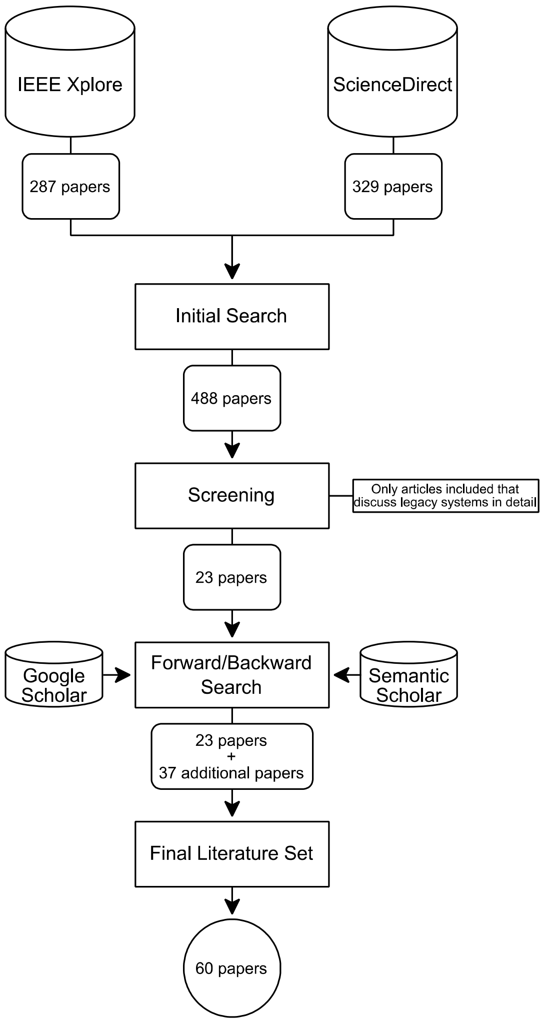


FIGURE 4. Visualization of the search process.

The following is a look at the temporal dimension and its connection to legacy systems.

a: A BRIEF INTRODUCTION AND HISTORY OF LEGACY SYSTEMS

A legacy system is *"Old code on old hardware"* [78], thus, *"an aggregate package of software and hardware solutions whose languages, standards, codes, and technologies belong to a prior generation or era of innovation"* [64]. Others refer to a legacy system when it has reached the serving stage of its lifecycle and a successor system is already on the horizon [68]. An often-quoted definition introduces legacy systems as *"large software systems that we don't know how to cope with but that are vital to our organization"* [1]. Another report states: *"In many ways, these [legacy] information systems are to an enterprise what a brain is to the higher species – a complex, poorly understood mass upon which the organism relies for its very existence"* [11]. Typically, a legacy system is understood as a large, old piece of software that is difficult to maintain, but still in operation and critical to the organization it serves and to that organization [48]. Because of the demands, costs, and risks involved in changing them, a typical legacy system falls short of what is needed by the business department and the larger the gap, the more outdated the information system [26].

The first generation of such systems deployed in the 1960-1970s are characterized by applications developed in machine languages, such as Assembly, or more rarely, COBOL[6] or FORTRAN,[7] running on monolithic, centralized hardware, typically mainframes. The data is frequently stored in files and the communication is batch-based in most cases. The following 2nd generation legacy information systems (short: LIS) introduced in the late 1970 – 1980s come along with a degree of modularity and improvements in data storage, thus, data base systems and data base management systems. Improvements in hardware performance

[6]COBOL – COmmon Business-Oriented Language.
[7]FORTRAN – FORmula TRANslation.

TABLE 3. Included papers.

Refs	Title	Year	Type	Source	Citations
[38]	Dilemmas in a general theory of planning	1973	Journal	FW	25386
[39]	Programs, life cycles, and laws of software evolution	1980	Journal	BW	1899
[40]	A Framework for Reverse Engineering DoD Legacy Information Systems	1993	Conference	IEEE Xplore	23
[41]	Issues and approaches for migration/cohabitation between legacy and new systems	1993	Conference	BW	11
[18]	A Strategic Attempt to Management of Legacy Information Systems	1994	Conference	IEEE Xplore	2
[42]	Legacy or liability-the will to change	1994	Conference	IEEE Xplore	-
[1]	Legacy systems: coping with success	1994	Journal	IEEE Xplore	543
[5]	Software aging	1994	Conference	BW	1474
[43]	Cash cow in the tar pit: reengineering a legacy system	1996	Journal	IEEE Xplore	88
[44]	A Survey of Research into Legacy System Migration	1997	Tech. Note	FW	47
[45]	An Overview of Legacy Information System Migration	1997	Conference	IEEE Xplore	61
[46]	How evolution of information systems may fail: many improvements adding up to negative effects	1997	Journal	BW	16
[47]	Implications of Distributed Object Technology for Reengineering	1997	Tech. Note	BW	63
[48]	Improving Legacy Systems Maintainability	1997	Journal	BW	26
[28]	Knowledge for Software Maintenance	1997	Journal	BW	16
[49]	Legacy Asset Management	1997	Journal	BW	14
[50]	Metrics and laws of software evolution-the nineties view	1997	Conference	BW	975
[15]	A Method for Assessing Legacy Systems for Evolution	1998	Conference	IEEE Xplore	123
[6]	Software's future: managing evolution	1998	Journal	FW	157
[51]	Throwing off the shackles of a legacy system	1998	Journal	IEEE Xplore	9
[52]	Decision model for legacy systems	1999	Journal	BW	90
[53]	Focus Issue on Legacy Information Systems and Business Process Change: A Business Perspective of Legacy Information Systems	1999	Journal	FW	59
[54]	Legacy information systems: issues and directions	1999	Journal	IEEE Xplore	586
[29]	Legacy systems: assets or liabilities?: a language action perspective on respecting and reflecting negotiated business relationships in information systems	1999	Journal	BW	4
[10]	Technical Opinion: Viewpoints on legacy systems	1999	Journal	BW	51
[11]	A Survey of Legacy System Modernization Approaches	2000	Tech. Note	FW	145
[55]	Co-Evolution and an Enabling Infrastructure: A Solution to Legacy?	2000	Book chapter	FW	35
[56]	It's Not just about Old Software: A Wider View of Legacy Systems	2000	Book chapter	BW	10
[21]	LACE frameworks and technique-identifying the legacy status of a business information system from the perspectives of its causes and effects	2000	Conference	IEEE Xplore	12
[57]	Leveraging legacy system dollars for e-business	2000	Journal	IEEE Xplore	628
[58]	Legacy System Anti-Patterns and a Pattern-Oriented Migration Response	2000	Book chapter	BW	19
[59]	Software maintenance and evolution: a roadmap	2000	Conference	FW	1104
[60]	A decisional framework for legacy system management	2001	Conference	IEEE Xplore	61
[16]	Ageing of a data-intensive legacy system: symptoms and remedies	2001	Journal	FW	87
[26]	Global Perspectives on Legacy Systems	2002	Book chapter	FW	7
[61]	More Legacy System Patterns	2002	Book chapter	FW	7
[62]	Alternative Theory of Legacy Information Systems	2003	Conference	FW	12
[63]	Reworking with a Legacy Financial Accounting System: Lessons from a Pharma Company	2005	Journal	FW	11
[64]	Improving Legacy-System Sustainability: A Systematic Approach	2012	Journal	IEEE Xplore	34
[65]	A preliminary review of legacy information systems evaluation models	2013	Conference	IEEE Xplore	12
[66]	Migrating a large scale legacy application to SOA: Challenges and lessons learned	2013	Conference	FW	49

TABLE 3. ***(Continued.)*** **Included papers.**

[23]	Towards a Framework to Assess Legacy Systems	2013	Conference	IEEE Xplore	18
[67]	A Framework to Assess Legacy Software Systems	2014	Journal	FW	12
[9]	How do professionals perceive legacy systems and software modernization?	2014	Conference	FW	138
[68]	Software evolution and maintenance	2014	Conference	FW	121
[69]	Does software modernization deliver what it aimed for? A post modernization analysis of five software modernization case studies	2015	Conference	BW	27
[7]	The legacy problem in government agencies	2015	Conference	FW	21
[70]	Understanding Legacy Architecture Patterns	2015	Conference	FW	4
[71]	Why Replacing Legacy Systems Is So Hard in Global Software Development: An Information Infrastructure Perspective	2015	Conference	FW	42
[72]	A Systematic Framework for Modernizing Legacy Application Systems	2016	Conference	IEEE Xplore	14
[73]	Analysis of Legacy System in Software Application Development: A Comparative Study	2016	Conference	FW	36
[74]	Co-Existence of the 'Technical Debt' and 'Software Legacy' Concepts	2016	Conference	FW	4
[75]	Dave Thomas on Innovating Legacy Systems	2016	Journal	IEEE Xplore	9
[8]	Dragging Government Legacy Systems Out of the Shadows	2016	Journal	IEEE Xplore	4
[76]	Managing legacy system costs: A case study of a meta-assessment model to identify solutions in a large financial services company	2016	Journal	Science Direct	48
[77]	Success factors in managing legacy system evolution	2016	Conference	IEEE Xplore	13
[12]	An assessment conceptual framework for the modernization of legacy systems	2017	Conference	IEEE Xplore	11
[13]	Legacy system and ways of its evolution	2017	Conference	IEEE Xplore	16
[17]	Embracing Agile Development Principles in an Organization using The Legacy System: The Case of Bank XYZ in Indonesia	2020	Conference	IEEE Xplore	2
[78]	Redefining Legacy: A Technical Debt Perspective	2020	Conference	FW	7

led to smaller servers, such as IBM's AS/400. COBOL and some 4G languages became very popular and were widely used. Online transactions were introduced as an alternative to batch-based communication. The 3rd generation arose starting in the late 1980s and featured new architectures, especially the client-server architecture. The character-based user interfaces on small screens were replaced by graphical user interfaces together with the newly invented computer mouse. Computerization continued to increase. ERP systems were introduced and networking with other companies via the emerging Word Wide Web progressed rapidly. Former individual IT applications that operated in isolation became more and more globally integrated [53].

However, it must be acknowledged that the IS and associated technology from the early 2000s are now reaching the end of their life cycle. Cloud computing, quantum computing, and other emerging technologies are revolutionizing the field and rendering their predecessors obsolete. The history of legacy systems does not end there, of course, but is driven by technological advances [1]. We will address the relationship between time, technology, and legacy systems later in this article.

b: SOMETHING FROM THE PAST – THE AGE OF LEGACY SYSTEMS

Strictly speaking, the term 'legacy' has neither a negative nor a positive connotation, but merely refers to *"something being handed down by a predecessor"* [62]. A legacy, then, is something that is left behind after a certain event (typically, the death of a person). In the context of legacy systems, these systems typically outlast the original people involved, such as business analysts, IS architects, or software developers, who retire, change companies, or leave the field entirely. So this definition of legacy implies a certain age of a system [48].

Generally, LIS have been around for a long time. Thus they were created many years ago and are still in operational use. A typical legacy system is old [1], [9], [15], [23], [60], [65], [75]. In this context, the term 'old' refers to a time frame of at least 3-5 years. However, usually, the term legacy implies an age of decades and is not used for newer or younger systems [40], [42]. Accordingly, IS that have been around for ten or more years are typically referred to as legacy systems [9], [23], [60], [79, p. 3]. Further, in the literature, some outdated systems are said to be 20, 30, or even 40 years old [13], [17], [40], [43], [48], [51], [53] and are possibly even older today. Remarkably, some of them have not been maintained for decades [72]. Legacy systems grow with the companies they serve. In this sense, legacy systems are typically not found in young startups, but rather in large, older companies and government agencies [78].

However, just because a system is labeled 'old' it does not necessarily mean that it is outdated [53]. Of course, it is possible for an old system to still do its work perfectly, despite its age, simply because it has been kept up to date through consistent evolution. Conversely, IS can age prematurely if not handled properly; therefore, legacy systems do not necessarily have to be particularly rich in years [60]. Yet, the characteristics of outdated IS usually only become noticeable after many years or decades, when a certain threshold of perception has been exceeded. Conversely, this means that every information system is (to a certain extent) already a legacy system, even if the legacy symptoms are not yet perceived as disruptive[8] [62]. Thus, there is no consensus on how old these systems need to be to reach the obsolescence threshold [48].

c: FREQUENTLY MODIFIED OVER MANY YEARS

"Not only systems and their implementations experience an ongoing evolution, the same applies for numerous businesses and the environments they are operating in" [81]. These businesses (and therefore also the information services used) are in a constant state of flux due to internal and external factors. Internal drivers for change are typically in the hands of the organization itself, such as new business opportunities, the introduction of innovative products or market entry in new countries with different legal frameworks, currencies, digitization and cloud strategies, corporate mergers, and cost-cutting initiatives. Internal drivers also include modifying IS to correct inaccurate specifications and faulty implementations. Moreover, IS itself, as part of the reality, considered as E-type systems,[9] drives the change. They are inherently prone to change and *"evolution is an intrinsic, feedback driven, property"* of them [39]. This means that new knowledge is gained through the creation of the information system and improvements are derived from it, which, in turn, are placed as requirements on the IS and thus - without external specifications - contribute to its further evolution [39]. In addition to internal factors, there are external factors to which an organization and the IS deployed must adapt and over which the organization usually has little or no control. These are, for instance, changes in the law, deregulation, taxation changes, globalization, competitive forces, changing customer needs, and new emerging technologies, such as the Internet, new standards, or ubiquitously available mobile connectivity. The Y2K problem and the introduction of the Euro in Europe are other known examples of external threads [6], [10], [23], [42], [53], [56], [61].

Everything – society, laws, available technologies, regulations, requirements, and organizations – is constantly changing. Consequently, the IS in operation must also constantly be adapted to the ever-changing requirements [6], [10], [42], [55], [61], [81]. If IS are not successfully and continuously aligned with ever-changing needs, they will become progressively useless.[10] At a certain point, they are called legacy systems. To some extent, a legacy system ensures the survival of an enterprise and thus reflects the evolution of the operating organization that runs the system. Consequently, it is possible to trace the evolution of an enterprise from its information systems [9], [18], [53]. Furthermore, some authors state, that such systems have *stood the test of time* [1], [10], [42].

2) A TECHNOLOGICAL PERSPECTIVE

A LIS can be, inter alia, viewed from a technical standpoint. The following sections present empirical statements that can be assigned to this technological dimension.

a: USE OF NUMEROUS OUTDATED TECHNOLOGIES

In legacy systems, the lack of standards, hence the existence of non-standard technologies is common. Some examples include self-developed database systems, programming languages, or home-grown memory management [40], [43]. Some features, such as concurrency or the aforementioned memory management, were not well-known in the early days of information technology. Accordingly, these features (today, an integral part of operating systems (OS), or as COTS) were often not available as standard features in the past and had to be implemented in an ad hoc manner [1].

Outdated storage technologies, such as file-based data storage that was used from the 1960s to the 1970s, are also common. This generation is generally batch-based and runs on mainframes [17], [69], [82]. Subsequent technologies, like IBM's IMS[11] and early DB2 databases, should also be considered legacy from today's perspective [69]. Besides outdated storage technologies, old programming languages such as Assembly or early versions of third-generation languages, like COBOL, CORAL,[12] or FORTRAN-66, PL/I,[13] Visual Basic, and IBMs RPG[14] [1], [9], [63], [69], [79, p. 3], are characteristic of legacy systems. Character-based user interfaces are also clearly considered deprecated; moreover, they are associated with limited usability [61], [69].

Deprecated hardware can be encountered in legacy systems, at least in certain cases [42]. Similar to legacy software, this kind of hardware can be recognized by many symptoms; particularly, upgrades may be difficult or impossible owing to missing interfaces. At some point, hardware manufacturers will cease production and then ultimately support for these products. Then, at the latest, obtaining spare parts is often deemed challenging, expensive, or even impossible. Operating outdated hardware is often considered

[8]Some authors exaggerate the situation and postulate that the code becomes a legacy the moment it is coded [80].

[9]E-Programs are applications that concern evolutionary activities (such as human or societal activities).

[10]This insight is also known as Lehman's first law of software evolution [50]. Accordingly, such systems have evolved successfully over a long period of time [5], [43], [51], [60] and contain many modifications, or customizations in the case of commercial off-the-shelf (COTS) [82], that reflect the ever-changing requirements [15].

[11]IMS – Information Management System.

[12]CORAL – Computer On-line Real-time Applications Language.

[13]PL/I – Programming Language One; also PL/1.

[14]RPB – Report Program Generator.

challenging as well. Moreover, it frequently does not provide sufficient processing power, main memory, mass storage, or network capacity, leading to poor performance [45]. The installed OS – which must be compatible with the hardware operated – may be as outdated as the hardware. Often, however, only the current version of the OS was not installed [76].

A further characteristic of legacy systems is old and outdated architectures [23], [67], [82]. For example, IBM's IMS, which was designed for early mainframes such as the AS/400, is a good illustration of such outdated architectures [17], [53]. In this context, the term 'legacy system' is often used as a synonym for the IBM mainframe architecture [78]. Nevertheless, even the newer client-server architecture is partly labeled as legacy [23], [67]. Some authors see an outdated architecture that no longer complies with the current standards as the defining characteristic of legacy systems [23].

Legacy systems with legacy technology need to be developed using CASE[15] tools and methodologies that are at the same level of technology as the legacy system. Consequently, applied tools and engineering methods are generally old and likely to be outdated as well. In extreme cases, this can mean that no CASE tools, IDEs,[16] compilers, or linkers are used [15]. It can also be a challenge to obtain these old tools or keep them operational [64].

New technologies have emerged and influenced IS during their evolution. New elements with current technologies may be added to older parts, sometimes called the core, whereby the core is often described as stable, but highly intertwined with more unstable peripherals [82]. Consequently, the modules interfacing the legacy core are often younger, as they have been built after and around it [55]. Over time, old and deprecated parts of the legacy system may have been replaced by newer and (from today's perspective, already outdated again) technologies. As a result, legacy parts with old technologies are often tightly interwoven with younger system elements that use newer technologies [42], [55]. Thus, a typical legacy system usually consists of many modules from different times with different technologies [58]. In some cases, as an example for high technological entropy, the application data is stored in both files and in a relational database, and a core that was implemented in an early 3rd generation programming language is operated in tandem with much more modern web technologies [82]. Other examples report, that different COBOL dialects are used within an IT application, some purchased (COTS) and some developed in-house. The infrastructure consists of various technologies, thus, of different hardware and OS [66]. This heterogeneity can then, for instance, be reflected in the use of ten different programming languages and more than 30 different frameworks [78]. Finally, it must be stated that not all legacy patterns need to be fully present at all times. In reality, it is more likely that only some patterns will be found in a particular system and these vary across different systems [61].

[15]CASE – Computer-Aided Software Engineering.
[16]IDE – Integrated Desktop Environment.

b: ERODED, DAMAGED ARCHITECTURES

As we will show later, IS are often changed with poor engineering. As a consequence, ad hoc changes and workarounds can be observed in operational practice, which lead to rapid degeneration of the architecture and often compromise conceptual integrity [47], [49]. Moreover, over time, customizing a solution to meet every unusual change request (over-customizing) can lead to an extremely layered architecture that ends up being de facto unmanageable [55]. Often, layers upon layers build on top of the original architecture and convolute in this way the whole structure [61]. Original architectural decisions and assumptions in the problem domain may be overridden by such changes, rendering the architecture inappropriate [61]. As a result, the architecture of LIS is often considered degenerated or eroded [5]. Finally, such eroded and inflexible architectures occur accidentally through *waves of hacking*, meaning perpetual unsolid engineering [58].

As described above, legacy systems often consist of various technologies, heterogeneous quality conditions, software packages, and platforms, making them very complex. This situation, where different programming languages or technologies are used within a single IS, is sometimes termed as 'Tower of Babel' and describes the issue that components with different technologies are less or not at all interoperable. In these situations, glue code and bidirectional connections between affected modules can often be observed as a consequence. This, in turn, contributes to an architecture that is hard to maintain [58].

If we turn away from the architecture to lower abstraction levels, we must state that another immediate consequence of recurring changes and unsustainable evolution is increased entropy. This is especially true if the architecture is not designed for maintenance and no reengineering has been done. Under such circumstances, poorly structured, cluttered, and bloated code is common and often referred to as 'spaghetti code' or 'code smells' [39], [46], [50], [61]. The so-called 'code decay' also refers to increasing entropy and must be considered as another synonym for the same or very similar concepts as the previous ones [59]. In addition, programs without any source code, test cases that are no longer required, or dead program code, which is no longer accessible via the call graph, can be observed in reality. The same applies to (temp) files, reports, etc. that are no longer needed but have not yet been removed from the product [16]. However, redundancy applies not only to the program code but also to the functional scope. The same functions are frequently implemented multiple times owing to a lack of integration [40], [43]. Thus, legacy systems should be seen as *"functional Silos that contain redundant, fragmented and inconsistently defined functionality"* [82] and data [40].

If we look at more modern, less monolithic, and more distributed legacy systems, it must be stated, that the counterpart to 'spaghetti code' at the systems level is an incomprehensive web of application systems [46]. As Bennett correctly predicted decades ago, the legacy problem will take root at the infrastructure level in parallel with the growth of distributed systems [1]. Spaghetti integration replaces the spaghetti code of monolithic systems [49], [66]. Redundancies can develop not only within a legacy system but also across an information infrastructure. Isolated, non-integrated legacy systems should then be seen as functional silos whose function is implemented multiple times in the information infrastructure, therefore redundantly; then, over the course of decades, complex, fragmented landscapes (complex jigsaw puzzles) of fragile, distributed applications that are highly interconnected emerge. Such landscapes can then consist of hundreds to thousands of linked IT applications [78], [79, p. 10].

If reengineering is neglected, constant changes are accompanied by increased entropy and this is above all also related to the geographical dispersion of an organization. Thus, entropy usually increases as a result of an expansion of a company or its IS, since this expansion into additional countries and regions is accompanied by complexity drivers, such as new currencies, laws, business rules, or languages [55]. This is especially true in the case of a so-called ocean of isolated or poorly integrated systems or as a consequence of a series of mergers with competitors who deployed information systems with very similar functionality [29]. Moreover, as a consequence of poor legacy knowledge, which we will discuss in the following sections, the risk is high that a function is unnecessarily re-implemented, simply because no one knows that it has already been implemented [16].

As a result of continued poor evolution, structures on a large (such as architectures) and small scale (such as classes and methods) lose stringency, decompose, or erode. They lose their structural integrity and have become inadvertently inflexible [5], [9], [62] *"So whatever structure originally existed has long since disappeared"* [1]. It should also be mentioned that eroded architectures on a large and small scale are also expressed using other terms such as poor quality, pollution, and code-smells. These terms should be considered synonyms for increased entropy in most cases. Some authors refer to this issue as software erosion [70].

c: MONOLITHIC AND CHALLENGING TO INTEGRATE

In the early days of computerization, information and communication technologies were expensive, and few routine tasks could be supported economically or automated by IT. There was little need for enterprise-wide or cross-company integrated IS as long as there were no or few other systems in an organization. Consequently, in the 1960s and 70s, monolithic systems were developed deliberately and the information infrastructure typically consisted of many autonomously operating and isolated application systems [18], [40]. These early legacy systems with their very limited hardware performance (compared to the computing power available today) usually come with highly optimized architectures and programs [48].

A monolithic system is difficult to evolve as it hinders the reuse of functions and modules. In this case, legacy systems may suffer from tight coupling, where unknown external applications directly call internal modules and functions of other applications. The applications affected suffer from reduced adaptability, reusability, and maintainability; changes to them are often extremely error-prone [58]. For example, due to the underlying architecture, such as CICS[17] transactions in COBOL on a mainframe, the implementation of a function-sharing mechanism may not be feasible, which means that a considerable number of components have to be implemented redundantly [70]. Thus, a monolithic architecture encourages redundancies.

The interfaces of LIS, if they exist at all, are typically implemented in old technologies that are often incompatible with more modern systems. Integration must be performed using ad hoc mechanisms (glue-code is required, for example). Consequently, the technical integration of legacy systems is complex [23]. In addition to technical integration, semantic integration can also be problematic if existing interfaces are incomprehensible and cannot be correctly understood owing to missing documentation, unclear signatures, and the use of non-standardized data types, for example. For these technical and semantical reasons, *integration of legacy systems is frequently described as demanding or impossible* [18], [40], [45].

Moreover, these difficulties not only apply to horizontal but also to vertical integration, which means the connection to reporting and management information systems [53]. We will return to this problem later.

d: RUNTIME AND MEMORY-OPTIMIZED PROGRAMS

Given the limited hardware capacity in the early days of information technology, the most important design features at the time were memory optimization and low computational cost [48]. Software optimization was required to get the application running at all. To achieve this goal, architectures and programs had to be trimmed for runtime efficiency at the costs of clarity, evolvability, and structure [1], [70]. The constraints posed by the limited hardware resources available were undoubtedly considered when making decisions about the architecture [70]. Under such circumstances, implementing ideal architectures was not always possible. Consequently, inadequate structures or redundancies need not necessarily emerge during evolution due to poor engineering. Rather, they were often intentionally implemented to offset insufficient hardware [53].

Whether architectural flaws emerged accidentally or implemented intentionally can often be recognized by occurrences such as the use of variable aliasing, single, global data structures, and hard-coded conditions instead of variables. Also, purely numeric names are seen, as these can be processed faster [43], or the user interface layer may be closely intertwined with the business layer and may contain constraints that would be better located in the database [61]. A degenerated architecture may also manifest itself in data access via back doors, or hard-coded, non-customizable

[17]CICS – the acronym stands for <u>C</u>ustomer <u>I</u>nformation <u>C</u>ontrol <u>S</u>ystem.

functions [9]. Somewhat generalized, partially intentionally implemented poorly layered monolithic systems can be found, where a strict separation of concerns between the user interface, data, and business layers is missing [9], [40], [61], [82].

However, there are cases documented where the initial architecture decisions do not fit the programming language used. This seems to be particularly the case with COBOL, where the object-oriented design already known was taken as a basis by the architects but this was not supported by COBOL (which was still in widespread use at the time) [70]. Thus, there are some IS with modern architecture, but improper technologies that do not match the architecture.

e: DOCUMENTATION DEFICIENCIES

The documentation of legacy systems is frequently deemed insufficient. Therefore, either no or few reliable formal models are available, and the description of system elements and functions significantly deviates from actual system implementation or behavior. This statement refers to both the business and technical aspects of IS [5], [23], [43], [79, p. 4, p. 10]. For example, the business process associated with a program is often undocumented, which means that the purpose or business logic of the program can no longer be deduced from the documentation [16], [40]. Older IS are often considered to be long outdated and users are faced with applications that do not yet meet their business needs. In this situation, users tend to create workarounds by using the system in ways that are not intended. For instance, control commands or work instructions are hidden in comment fields, or the users simply do parts of their jobs outside the systems. The problem here is that these alternative business processes are typically not documented or communicated. As a result, entropy increases while comprehensibility at the organizational level decreases [46]. The problem here is, if the documentation is imprecise, out of sync, or inscrutable, it immediately loses a large part of its usefulness [1], [42]. Further, the documentation, if it exists at all, is rather often seen as an aide-mémoire in which the details, the minutiae of the business, the design, and the implementation of the application are documented [28].

With regard to the technical components of IS, there is often no suitable documentation of the data structures, meaning the implemented data model, especially in connection with flat-file data storage [44]. It can be roughly assumed that only half of all legacy systems have a completed data dictionary, while the other half is poorly documented. Furthermore, the documentation primarily focuses on development rather than maintenance personnel's need to utilize and expand upon the documented knowledge [28]. In some cases, there is no legacy system documentation at all [63]. Often, the code and the implementation are the last 'documentation' of the actual business model or domain (tacit) knowledge [58]. Various reasons are cited for the inadequate documentation, which we discuss in detail in the following section (human perspective).

However, in addition, it also may lead to the assignment of external staff not familiar with the system and who cannot match their knowledge to the code. This, in addition, may contribute to inaccuracy [5]. However, the quality of the documentation, which must be considered good for new IS, deteriorates in most cases over time, or more precisely, with the number of changes made to it. This is due to changes to the product (and in particular hotfixes, which focus on restoring the functionality of the system), which are often not documented or only very superficially.[18] Accordingly, the lack or degradation of documentation is reported to be directly proportional to the number of conducted changes and the size of legacy IS [26].

Documentation deficiencies must be seen as a precise indicator to identify legacy systems [1]. However, caution is required: It is important not to make sweeping generalizations at this point. In some cases, the quality of the documentation is described as exceptional [17]. Generally, the quality of documentation often varies considerably between enterprises, IS within a given organization, and even across modules within a single information system. It ranges from virtually no documentation to very good, high-quality documentation. In summary, the overall quality of documentation must be classified as out-of-date in most cases [28]. Moreover, reading and processing the documentation require compatible hardware and software, which may be difficult to obtain [1], [42].

f: LARGE IN TERMS OF SIZE, DATA, AND FUNCTIONALITY

As described before, the steady adjustment of applications to new requirements is widely known. Accordingly, legacy systems are typically modified and extended over a long period of time and become quite large; also well known as Lehman's 6th law of software evolution [50]. In particular, this can translate to several million lines of code and thousands of data structures [5], [40], [72], [79, p. 69]. Although Bennett mentions the size of legacy systems as a typical characteristic, the primary argument here is that large programs are considerably more difficult to comprehend and modify than smaller ones [1].

However, not only the size of the application program but also the volume of business data generally increases over time. Substantial amounts of data can accumulate over years of operation [1], [41]. For example, a telephone service provider typically handles and stores all transactions for its customers daily, generating huge volumes of data records over time that are retained for several years [79, p. 88]. Ultimately, a major part of legacy stock should be regarded as so-called *very large business applications*, meaning business applications that are of strategic importance for the success of an organization [84] These very large business applications

[18]It should be noted that software engineers tend to dislike writing documentation and also dislike getting involved in other people's convoluted code [83].

are typically found in a variety of business areas, such as accounting, logistics, sales or marketing [85].

g: DATA INCONSISTENCIES

Business rules are sometimes scattered across the IT application, meaning that they are implemented in different locations. Business rules sometimes also have to be implemented completely redundantly, especially in the case of isolated, standalone IT applications. Under these circumstances, if they need to be changed, it is difficult to find all the places where the rules have been implemented and to change them accordingly. However, if business rules are not changed consistently, inconsistent business decisions and data inconsistencies result. Moreover, business data is often reported to contain many redundancies. This may be, for instance, in the case of various distributed data stores where a central database is missing. The data must then be replicated, which can lead to further technical transfer errors and inconsistencies creeping in [41]. If interfaces are available, they are often haphazard, and data inconsistencies may occur between applications [79, p. 3]. Today's databases use a variety of mechanisms to ensure data consistency, including the use of constraints. With unconstrained storage in files instead of databases, ensuring the integrity of business data is much more difficult and is partly not achieved [51]. To summarize, the data of legacy systems is often scattered across different systems inconsistently defined with respect to each other, and is often kept redundant by replication and is thus inconsistent [40], [70].

h: LIMITED REMAINING VENDOR SUPPORT

As far as support is concerned, it is sometimes only available to a limited extent or is no longer available at all. In such a scenario, bug fixes are no longer provided, support (such as hotlines) is no longer offered, no new product releases are available, and no more licenses are granted. However, this usually only affects parts of the system, such as a specific module, library, or parts of the hardware. If customized packages, standard software, or COTS have been purchased, the remaining support for them may be limited or has been discontinued by the vendor, possibly long ago [9], [23], [43], [53].

i: WEAK PORTABILITY

Legacy systems are characterized, to some extent, by the fact that they make direct calls to the OS. Such use of OS functions, which also includes the use of OS-specific utilities, leads to a strong coupling between an application and the OS and, at the same time, to a decrease in portability. As a result, legacy systems suffer from weak portability unless appropriate technologies such as middleware systems or Java technology are used [58].

However, the legacy patterns present in a legacy system do not always have to have a negative impact. If, for example, the application was hard-coded for a certain OS, it is difficult to port it to another OS. However, if all users of this application exclusively rely on this certain OS, then weak portability is de facto present but does not come to bear, as long as no other OS is used [61].

3) A HUMAN PERSPECTIVE

In the following sections, we will turn to the legacy systems characteristics that can be viewed from a human standpoint. For this purpose, we look at the domain experts who use legacy systems to fulfill their operational tasks, the IT operations personnel who are responsible for the smooth functioning of the operated applications, and the management staff.

a: MISSING LEGACY EXPERTS, MISSING KNOWLEDGE

Regarding legacy systems, a lack of knowledge is one of the biggest problems: *"…the big problem is that you can't find people to understand them [legacy system] and understand the technology"* [9]. There are various reasons for this lack of knowledge. First, engineers prefer dealing with new, exciting technologies instead of old systems with their obsolete technologies. As a result, it is difficult to find experienced personnel who are familiar with, or willing to learn, technologies that are sometimes 20 or 30 years old or sometimes even older [1]. Software engineers, for example, with knowledge in assembly language (abbreviated as ASM) or third-generation languages, such as COBOL, are difficult to attract for maintenance tasks of these old technologies [53]. The lack of knowledge also applies to the scarcity of knowledge about the required toolchain (with required editors, compiler, linker, debugger, and so on) and associated development paradigms [72]. For others, maintaining legacy systems is seen as an end to their careers, and also looks bad on a resume. It can be a career risk to be involved in maintaining legacy systems and their associated old technologies that will eventually become obsolete [61]. Instead, developers would rather acquire new knowledge through the use of cutting-edge technologies. However, this tendency (hence the desire to use cutting-edge technologies prematurely) contributes to technological obsolescence on the one hand and staff turnover on the other. Moreover, after a short period of time, proficiency in older technologies may diminish, thus increasing the risk of becoming obsolete [78]. Unfortunately, education contributes to the scarcity of qualified staff. Students want to learn the latest technologies and avoid older ones, perhaps because they do not see the industry's need for them, and universities train them accordingly to meet the demand. Higher education focuses almost exclusively on new knowledge and technologies [28], [55], [75]. However, this training does not meet industry needs for older technologies, leaving a skills gap in the labor market [26]. Moreover, there is a danger that information technology as a science will forget its own heritage if the old technologies no longer appear in the curricula [26]. Finally, it is difficult to acquire the necessary knowledge, thus, to find personnel who are experienced in old technologies, as they are often not freely available on the labor market. Seasoned people are scarce and due to regular supply and demand mechanics are usually also very expensive [4], [28], [55], [78].

The obvious reason for the lack of knowledge is that experienced legacy experts change their position within a company, leave the company, or exit the labor market and retire [53], [56]. In some cases, it is reported, that the original development team has left the company, leaving *no one with original knowledge* to replace the legacy system [63]. Such a loss of the original development team may, for example, occur in the context of company mergers or acquisitions, if the development team was not retained or otherwise relocated within the company [74]. Knowledge about IS may also be lost through geographical changes of an organization. If, for example, support is centralized, thus, moved from the branch, a location, or a country to the headquarters, the sometimes very specialized local knowledge, such as special customer requirements, and location-specific business processes, is usually lost as the knowledge cannot be completely transferred to the employees at the headquarters [55]. Poor knowledge management combined with neglected documentation, etc. can lead to a situation where business or domain experts no longer know their own business processes and rules and the rationale behind them. They have to ask IT experts about the business model implemented in the application, who have to extract this knowledge from the code or documentation [28].

The legacy knowledge is typically not documented very well, as the documentation task is often neglected. In addition, tacit knowledge cannot be documented very well. Of course, old documentation is also written using the terminology, or taxonomy, in use at the time the documentation was created and the reader must be familiar with this taxonomy, otherwise, he will not be able to understand the documentation properly.

However, even if the documentation is sophisticated and can be understood by the reader, it is nearly impossible to gain a holistic understanding of how a system works from the documentation alone[19] [28]. As mentioned before, the legacy systems themselves (such as the code, the personnel involved, test plans, and cases) are often the last reliable documentation available [86]. Nevertheless, it must be assumed that not all knowledge can be extracted and re-documented from the system alone without a knowledge carrier. When these legacy experts leave the company, parts of their knowledge are lost to the organization forever [16].

To ensure effective and efficient legacy system maintenance, it is not enough for staff to possess business (or domain) *or* IT knowledge – legacy experts need knowledge of both areas (domain and IT), either *externally* at suppliers or *internally* at the company's own IT department [62]. In other words, the maintenance staff responsible must not only be familiar with the design and implementation of the IT application, but also understand the business model embodied in the application, know the purpose the information system serves, and be aware of the users' expectations. Moreover, the staff should use, understand, and commit to a common vocabulary [28]. This need for combined or broader knowledge is also addressed by the concept of co-evolution. Weak co-evolution leads to poor, ineffective communication and misunderstandings, resulting in suboptimal IS and, thus, legacy systems [61].

In a broader sense, the required knowledge may be present but poorly available in the company. This is due to the fact that expert knowledge is usually highly fragmented and widely scattered throughout a company, held by many employees from different areas and domains. Even after several years of working on such systems, the knowledge that many employees acquire is usually incomplete, limited to their domain and therefore fragmented. In addition, experienced staff is often hardly available, as they usually work to capacity in their projects and with their respective outdated systems [9], [43], [65]. It was also found that in-depth knowledge is often limited to just a few employees. These employees then spend a large part of their time advising other employees [28]. Consequently, there is typically a lack of a profound, end-to-end or holistic understanding of internal implementation or system behavior [1], [9], [42], [45]. *"Very few people (if any) understand the system's internal data structures and logical functions"* [51].

Finally, when legacy systems are used, it can be assumed that the associated IS organization is also operating on an outdated level. For example, if mainframe technology is used, aspects such as personnel, processes, CASE tools, or support structures are mainframe-oriented and thus at the same technological level as the legacy system itself. If the legacy system is considered outdated, the associated IS organization must also be considered outdated [49].

b: HARD TO UNDERSTAND, CHALLENGING TO MAINTAIN, RESISTING EVOLUTION

Understanding a program requires up to 50% of the total change effort and leads to a mental model of the application [49]. However, the time-consuming familiarization process to gain the required domain and IT knowledge cannot be avoided because the artifact must first be understood before profound and consistent changes can be made [9], [53]. Consequently, a significant amount of effort must be invested in building an in-depth understanding of the application. The lack of this prior knowledge can seriously jeopardize business operations [28]. In the case of legacy systems, business processes and rules are often undocumented. However, to change something, it must generally be understood beforehand. An unknown or incompletely understood business model is an obstacle to development because it is difficult to reconcile business changes with necessary IT changes [61]. Typically, a holistic or end-to-end

[19] It should also be noted that individuals tend to rely on their colleagues' incomplete and inaccurate knowledge rather than reading available documentation. The documentation, if it exists at all, is rather to be seen as an aide-mémoire in which the details, the minutiae of the business, the design and implementation of the application are documented [28].

understanding is lacking due to fragmented, scattered knowledge. In such a scenario, business analysts cannot mentally penetrate the existing processes and rules and the impact of change requests on the business, resulting in suboptimal business processes and IS [28]. It is worth noting that conducting a time-consuming analysis and building a mental model does not (directly) provide any added value to the product [49].

Legacy systems are typically characterized by a great variety of partially outdated technologies, lack of experts, imprecise or missing documentation, 'code smells' and 'spaghetti code', eroded or optimized structures, and thus, increased entropy. As a result, these systems are often considered confusing and challenging to understand [1], [42], [45], [48], [53]. For example, bizarre restrictions might occur in old programs, like 64 kByte segments and 80-character-wide datasets, the purpose of which becomes clear only with corresponding knowledge of the hardware history. Without legacy experts, no one can explain why these issues were implemented in this way. In addition, legacy technology may be less comprehensible in some cases; for example, if an outdated technology only allows short identifiers (e.g., 8 or 16 bits wide) [43], and as a result, many cryptic abbreviations and barely self-explanatory identifiers are present [16]. Especially in older technologies, such as in Assembly or BASIS, computed or parameter-driven 'goto statements' are common but hardly understandable [42]. Optimized structures trimmed for memory and runtime together with missing knowledge make it very demanding to understand and maintain such systems [48]. Moreover, systems that prioritize runtime efficiency pose challenges not just for comprehension, but particularly for testing due to their convoluted and disordered structures [1]. In this case, it is difficult to identify all the parts of the system that are affected by a requirement and to implement those changes [1]. In addition, due to the high entropy that legacy systems suffer from, it is even more challenging to find and consistently address all the parts of the code that need to be changed [53].

Of course, a weak lexical meaning may also result from doubtful or thoughtless naming of things, like variables or classes [16]. However, it is also possible that completely senseless and unsubstantiated rules are applied in connection with legacy systems, such as 'don't use C++ templates' or 'use max. 6-character long' variable names, because these can be processed more efficiently. Nevertheless, such *"unsubstantiated folklore"* can lead to unnecessarily convoluted code that is unnecessarily hard to understand and maintain. In any case, such structures typically do not promote maintainability [61]. Here one can also refer to so-called guard code, which is also often used in problem-domain-oriented code to catch invalid program calls. Ultimately, however, these mechanisms also lead to code that is characterized by poor maintainability, adaptability, comprehensibility, and reusability [58].

IS continues to evolve over the years and increase in size and complexity to become massive black boxes that are partly almost impossible to understand[20] [50], [51]. Less colloquial, a legacy system must at least partly be considered a so-called wicked problem [38], which means that *"The process of solving the problem is identical with the process of understanding its nature"* [56]. This means that understanding legacy systems is the main prerequisite to solving problems and changing systems. Therefore, changes to an existing, degraded system are estimated to take four times longer than new development [49]. Accordingly, they significantly resist modification and evolution [p. 3]79.

Developers need to understand the code they are modifying. Insufficient knowledge (about the programming language, data structures, underlying business culture, domain knowledge, principles, assumptions, etc.) causes code decay. This can lead to a tipping point, where the mental capacity of the development team can no longer keep track of the changes and their implications, and evolvability reaches a dead end [5], [68]. Lack of knowledge in both areas (business, IT) leads to expensive and delayed work with inferior results [28]. Moreover, a lack of understanding of the rationale behind the code – the implemented business model and the former or underlying requirements – leads to a considerable degree of uncertainty and consequently to resistance to changes [58]. In addition, change cannot be fully anticipated in the case of structures that have eroded and are no longer comprehensible. In some cases, it is simply beyond the intellectual capacity of the people involved. In this case, it is understandable that changes are more likely to be rejected because the consequences cannot be properly assessed. Human resistance to change arises, which then contributes to low maintainability and adaptability [61]. Finally, it should be mentioned that the implementation of the legacy system and the business model contained therein is often the only reliable documentation. Therefore, the maintenance of these systems is usually based on the only reliable source, namely the code [1]. Missing knowledge is a burden for solid engineering.

Legacy systems mostly suffer from eroded and monolithic architectures. Such architectures do not allow new functions to be added simply using plug-ins. Instead, new functions must be implemented in the core at great expense [23]. Hardware often does not have sufficient computing, memory, or data-transfer capacity to support new functions. To implement and operate new features, further accompanying measures such as an upgrade of the hardware may have to be implemented. If necessary, new features should be discontinued if the required resources cannot be upgraded [53].

If all places to be changed are not found or not modified correctly, defects are inevitable. Under such circumstances, even small changes can lead to serious failures, which in turn encourage the emergence of subsequent failures. Accordingly, legacies are sometimes described as *inflexible* and

[20]Some have compared understanding legacy code to archaeology and understanding ancient artifacts from long-dead civilizations [80].

brittle and their maintenance does not meet the required agility, as it is time-consuming, expensive, and partially risky [1], [5], [15], [42], [45], [51], [60], [p. 4]79. The code is becoming increasingly vulnerable to new bugs [82]. *"Even in the early years of the industry, observers were able to document situations in which each error corrected introduced (on average) more than one error"* [5].

c: CARELESS CHANGES

To change a program sustainably, the engineer in charge must first understand the architectures, principles, and inner dependencies and then integrate the required changes into the system. People who do not understand an architecture cannot consistently modify it. This results in unsustainable changes. Legacy symptoms increase in accordance with the number of unsustainably implemented changes [60]. Such poor engineering is often accompanied by code cloning, which unnecessarily increases the size and complexity of the application [61]. The existing functions are copied, modified, and added to the program. *"…the easiest way to add a feature, is to add new code"* [5]. As a result, the code becomes increasingly redundant and ultimately the entire system becomes unnecessarily bloated [11]. In addition, ad hoc changes lead to the rapid deterioration of the conceptual integrity described above [47], [49].

In the case of legacy systems, the time available to conduct changes was often said to be too short. Shortage of time leads to shortcuts, workarounds, and compromises, resulting in unsound solutions. Developers have to skip parts of their engineering tasks or take shortcuts to achieve their goals. Thus, only having little time contributes to decay [55]. Moreover, if the motivation is lacking to deal with other peoples' source code, and when competencies regarding software quality and quality assurance are missing, careless changes and ad hoc solutions can be observed [5], [55], [82]. In particular, the creation of extensive documentation is expensive and time-consuming and often needs to be reduced or omitted to increase the short-term productivity of software engineers [28]. In conclusion, legacy systems are often modified in unsustainable ways [61].

The documentation quality of legacy systems is typically inadequate. The employees responsible do not document their changes sufficiently or at all for a variety of reasons (e.g., lack of time, motivation, lack of quality assurance). The effort and expense of creating sophisticated documentation is often considered unnecessary as the existing generation of maintenance staff already has the required knowledge; this being said, how documentation is handled varies from company to company [28].

Technologies that enable continuous development over a long period of time have been in use since the 1970s. Paradigms such as 'design for change' based on 'information hiding,' 'abstraction,' 'separation of concerns,' 'object orientation,' and others were already known at the time. However, these principles were often not consistently implemented. In such cases, the code is often perfectly written in terms of functionality, but not designed for good changeability [5]. Apart from that, some authors believe that the longevity of outdated systems or the applied technologies was often not anticipated [62] and therefore wrong assumptions, suboptimal design decisions, inappropriate technology choices, relatively inflexible architectural decisions, etc. were made [55], [64]. In addition, in 20, 30, or more years of operation, IS will face many and perhaps fundamental changes. Despite the principles mentioned above, it is difficult, if not impossible, to anticipate these possible changes, some of which are decades into the future, and to design appropriate architectures for them [6].

d: PERSONAL OR CULTURAL RESISTANCE

People are individuals with their own personal likes and dislikes, which are also reflected in their behavior in organizations. Personal attitudes can lead to employees insisting on familiar technologies instead of switching to new and possibly better technologies for the company. This can lead to employees completely rejecting new solutions and training [55]. Conversely, younger workers are often very interested in the latest technologies and tend to ignore existing solutions using older ones, even if they are adequate for the company. In addition, younger workers may pursue their own agenda, which may be influenced by personal career decisions. And here, the latest technologies and cutting edge projects are mostly preferred, as they seem to be better for one's own career [55].

Another noteworthy human factor is that projects may also fail due to employee resistance or a corporate culture that is not very innovation-friendly, especially when old assumptions and management approaches have to be questioned [9], [53]. Replacement systems may be rejected by employees as they have to invest a lot of time and energy in learning the new system and changing their ways of working [1]. In addition, it is possible that managers stick to a legacy system they developed themselves due to their personal attachment to it. This biased behavior can negatively impact decision-making [42].

Other human issues such as the fear of losing a monopoly on knowledge or one's job may also hamper modernization projects [12]. In conclusion, at a macro-level, the corporate culture can be either supportive or averse to change [55]. From an overall business perspective, such cultural resistance may lead to suboptimal decisions [55], and this in turn contributes to the emergence of legacy systems.

e: ONE-SIDED FEATURE EXTENSION, NEGLECTED REENGINEERING OR MODERNIZATION

To keep maintainability and to avoid the degradation of architectures, modifications must be performed sustainably [53]. In contrast, information systems age prematurely if they are not handled with care, which means that workarounds are implemented, documentation is poorly maintained and regular reengineering measures are omitted [13], [47], [48], [49], [75]. This finding coincides with Lehman's second law, which postulates an increasing complexity over time when adequate countermeasures are not implemented [39], [50].

Available resources can be invested in reengineering measures, functional enhancement, or a combination thereof. Reengineering measures reduce the effort of future changes, whereas functional enhancements immediately satisfy the business unit's most urgent requirements, but at the same time increase entropy. Given the finite remaining lifetime and restricted scope for action due to limited maintainability, the challenge of managing these systems is to find appropriate and cost-effective solutions to realize new requirements. In this area of conflict, a careful balance must be established between functional enhancement and reengineering measures. Inadequate or omitted remedial action leads to characteristically high entropy of legacy systems [1], [47], [60].

The early replacement of a legacy system is characterized by unpredictability. For instance, management typically does not know whether a new technology will succeed in the marketplace or quickly fade into obscurity. In addition, the expertise required for new technologies is difficult to find and therefore expensive. Consequently, early adoption of new technologies is risky, which explains why managers tend to rely on established technologies, which in turn means that existing systems remain in operation until new technologies become established and qualified personnel are trained and available [64]. Finally, replacing a legacy system always entails loss, as well as financial and intellectual investment. Reasons like investment and protection of know-how drive an excessively extended operating period, which promotes the development of legacy systems [64].

Far-reaching changes, such as the replacement of a previous in-house development by COTS, or the switch to new technologies, such as a new programming language, can be accompanied by the release of mostly long-standing and highly experienced personnel. These legacy experts who are affected can attempt to defend themselves against such changes and the associated loss of their relevance and not disclose their knowledge. Thus, profound changes are potentially associated with conflicts between the business or IT departments and management [64]. In poorly performing projects, the people involved must also have the courage to communicate the poor state of affairs. However, project conditions may be such that the participants fear for their jobs or funding for their poorly performing projects, and they therefore gloss over results or conceal negative facts. This in turn can lead to serious consequential errors, as management is unable to initiate appropriate corrective measures due to embellished reports [8].

Monaghan [78] describes a clear conflict between management and their business point of view and the IT development departments with their more technical view of a legacy system. The latter would often prefer to use the latest fancy information technologies, while management sees little benefit in this. When the budget is tight, it is likely that information technology updates will be delayed in order to prioritize business functions, thus leading to IT obsolescence. Management is faced with the difficult task that such purely technical changes must also pay off financially and remain within the budget. Yet, it is often impossible to prove the return on investment of IT updates. In some companies, this leads to IT updates being repeatedly postponed and the IT application becoming deprecated over time from an IT point of view [55]. With this in mind, Bennett and Monaghan believe that the primary cause of legacy systems is past misguided decisions or inaction on the part of management, which failed to demand necessary reengineering in a timely and sufficient manner [1], [78], and not so much due to technical reasons. To address such weakness, the goal must be for organizations to anticipate aging using risk assessments or scenario analysis tools [56]. Young-Gul Kim argues in a similar vein, attributing the causes of decay to developers and managers. The former contributes to the decay through a lack of solid engineering. However, the blame lies mainly with a management who has failed to provide the necessary preconditions (for instance CASE tools, training and education, sufficient development time, and engineering guidelines) required for solid engineering [48]. In this context, Bennett [1] claims that most legacy systems have never been re-engineered. He refers to this situation as a double punch that leads to legacy systems. Firstly, changes cause entropy and secondly, reengineering measures are omitted. Finally, the state of the legacy depends on changes and treatments. Without proper treatment, a system gets sick, thus increasing the legacy status [47].

f: POOR USER EXPERIENCE

Legacy systems often have an outdated user interface that makes working different and can be limiting. For instance, some legacy IT applications (e.g., mainframe) are characterized by character-based input screens, where input is possible only with the keyboard, but not with a computer mouse. From today's perspective, such an old-fashioned user interface can result in a poor user experience. Moreover, legacy systems often suffer from poor performance and slow response times, which, in turn, lead to so-called waiting stress [61], [69]. These extended wait times can have a negative effect on the users' perception of the legacy system [46].

g: AGING WORKFORCE

When older employees retire, they take with them the knowledge they have accumulated over a lifetime. It is the accumulated knowledge of the business processes and rules and the information technology used that forms the basis of the IS used in the organization. Particularly, this knowledge includes lore about the evolution of these elements. People with many years of experience know (often only subconsciously) why processes and IT evolved the way they did. This knowledge is no longer available to the organization when they leave unless it has been captured beforehand [7], [66], [69].

The acquisition of the knowledge required for evolution is not trivial or quick, even with proper prior training. It usually takes at least six months for new employees to become productive. However, it takes about two years before these employees can effectively and efficiently perform their maintenance tasks [28].

h: OPERATIONAL COMPLEXITY

Legacy systems are not only large in terms of size and functionality but often also in terms of the number of users. In this context, large typically means thousands of users. It has also been reported that the number of operational users working with a system often increases over time [23], [41], [82].

The operational complexity of legacy systems increases because the number of social interactions and interfaces is greatly increased. This is roughly comparable to the large number of technical interfaces due to a high level of coupling between IT components [55].

Another antipattern of legacy systems can be seen in the criterion that changes to the application can only ever be put into production in conjunction with downtime. New classes, for example, modified functions or changes to the user interface always require the server to be shut down to implement the changes. This always results in undesired downtime, especially for 24/7 applications, which must be handled accordingly [61].

i: COMMUNICATION PROBLEMS

Communication between agents (such as software developers or business analysts) also plays into the formation of legacy systems. Initially, both groups speak slightly different languages and have different perspectives. Business users think that developers deal mainly with technology. The business users deny developers a deeper understanding of business issues. In contrast, the developers take the view that the business users are not able to name and prioritize the requirements clearly, understandably, and consistently; therefore, business users are perceived as fickle. In addition, the IT domain complains that the deliveries and the associated effort and time are not perceived and appreciated by the business users. These communication deficits ultimately lead to a tense working atmosphere, but above all to deliveries that do not fully meet the expectations of the business users. Co-evolution, a prerequisite for the successful evolution of IS, is disturbed and promotes the formation of legacy systems [55].

j: MANAGERIAL DISCONTINUITY, MANAGEMENT DIFFICULTIES

As discussed later, legacy systems can only be replaced under challenging conditions and at high cost. Consequently, managers responsible may delay essential measures to avoid endangering their careers in the event of failure. Moreover, if managers only deal with a legacy system for a short time, it is conceivable that they do not want to invest effort and energy in improvement. Without the courage or power to get appropriate projects off the ground, the legacy system will continue to operate [18]. A rapid turnover of executives can contribute to the problem of legacy issues. If they are only in charge for a short time and can only report short-term successes, there is little motivation to undertake a long-term modernization project [55]. Short-term and accountancy-minded managers are also more interested in low-cost patches. Reengineering measures are often not initiated by the managers responsible, possibly because they do not know that they have to order these measures themselves. Reengineering measures are only considered when the consequences are seen in their budgets [42]. It should also be said that developers generally do not make major optimizations on their own; especially not in legacy systems, where changes are risky and costly. Improvements and optimizations to the code must therefore explicitly be instructed by management or at least communicated as a desired, independently perceptible option for action [42]. Managing legacy systems is also challenging due to frequent changes in the business model and in information technology [49]. Legacy systems, therefore, pose a particular challenge, not only for administrators but especially for management.

4) A BUSINESS, ORGANIZATIONAL PERSPECTIVE

After examining legacy systems from both a technical and human perspective, we will now address the business aspects of these systems in this section.

a: UNCLEAR OWNERSHIP, POOR BUSINESS MODEL

In legacy systems, it could be that the responsibility for a module or the business model bound to it does not exist. Such unclear responsibility can then lead to missing or contradictory decisions, missing budgeting, and thus paralysis of the evolution. While the responsibility for a business function, such as lending, can usually be reassigned relatively easily in retrospect, this task is much more difficult for shared components because they have to serve multiple customers. Here, conflicts can, inter alia, arise with regard to pro-rata payment or conflicting requirements [55].

b: A HUGE INVESTMENT

Having been developed for many years, legacy systems have typically been funded by large budgets. This alone makes legacy systems a significant investment [54]. It can be assumed that these investments have already been depreciated and are earning a correspondingly high profit (or otherwise have a positive impact on the company's income statement, as the depreciation for these systems no longer needs to be taken into account). Shutting down such systems is, therefore, associated with the loss of future simple revenues, which explains why efforts may be made to operate such systems as long as possible. Conversely, premature shutdown involves the loss of future profits and possibly a massive loss if the system has not yet been depreciated. This can endanger the existence of a company [64].

Employees have dealt with evolution over decades and have incorporated many enhancements and problem solutions into such systems [48]. Accordingly, legacy systems incorporate years of in-depth business knowledge and have withstood the test of time by constantly managing their evolution and becoming robust in hardware, software, and functionality over the years or decades of business modeling, bug fixing, and operation [18], [42], [48], [82]. Through years of debugging and sharpening functionality, hard-earned reliability, and (tacit) knowledge have been built up that is documented in this quality only in the existing system, making a greenfield approach for a successor system almost impossible or at least very difficult. This stable business model, together with the tacit knowledge behind it, is the real value of a legacy system, and any replacement threatens it [29], [58].

c: SOURCE OF BUSINESS KNOWLEDGE

Over many years of development and operation, legacy systems have accumulated a significant amount of business knowledge such as business processes and rules, algorithms, policies, expertise, and solved problems embedded in programs and databases with their tables, dependencies, and constraints [1], [23], [41], [43], [45], [48], [51], [53]. Another valuable part of a legacy system is the data itself [26]. This can be, for example, product data in the form of material master data, parts lists, customer data, or transaction data from a bank. The loss of any of these types of data could result in significant adverse effects. IS provide a database for business intelligence and should therefore be regarded as a foundation for consolidated information about an organization [54].

Consequently, outdated systems should be seen as a valuable source of business knowledge essential for understanding the evolution of an organization. Legacy systems can be considered as a corporate knowledge repository [48]. Moreover, legacy systems are usually the only reliable source of business knowledge [86], as mostly insufficient effort is invested in securing it in the documentation. However, even if much business knowledge is encoded in an application, it cannot easily be recovered. Only legacy experts can read, understand, and extract the often wicked legacy code and the business model contained therein [43]. Consequently, replacing a legacy system entails the risk that a significant part of the knowledge stored in the legacy system will be lost forever [54].

d: LEGACY SYSTEMS PERFORM CRUCIAL TASKS AND ARE BUSINESS-CRITICAL

Legacy systems are often referred to as the backbone or core service of a company's data flow, which is essential for consolidated information supply [9], [17], [51], [54]. They typically handle or support crucial tasks, such as payroll management or financial transactions, with highly customized functions [9], [23], [40], [45], [82]. These systems fulfill essential tasks for the area in which they are used and thus offer high business value by generating considerable revenue or indirectly contributing to profitable company results through efficient process handling. LIS are frequently considered a *substantial investment* and *valuable asset* [9], [17], [23], [45], [51], [53], [54], [60], [65], [82]. Legacy systems can remain a necessary, valuable asset even after they are no longer operational. For example, statutory requirements sometimes dictate very long retention periods for data and documents, especially for security and tax-relevant documents. In this context, it can be an important task of a legacy system to enable access to this legacy data. Thus, it is valuable because it fulfills a legal requirement [42].

Even when they become functionally inadequate, these systems provide a solution for managing and governing the daily operations of organizations through proven business processes and rules that have been developed, operationalized, and improved over decades [1], [7], [15], [18], [42], [65]. A failure mostly means the collapse of an organization's day-to-day business or some primary corporate functions with severe consequences [54]. In particular, this applies if the downstream systems are insufficiently decoupled, and an interruption also leads to the collapse of the downstream systems. If one of these systems goes down, a large number of others follow [51], [77]. These IT applications and the accompanying personnel then constitute a crucial asset because they can be used to govern the company's day-to-day business [49].

Finally, *"By definition a legacy system is business critical. A system that is old and obsolete and is not business critical would never reach the status of legacy"* [9]. Therefore, information systems that are not critical to the business in which they are used are not considered for further evolution [15]. Accordingly, typical legacy systems and their essential functions are more likely to be found in the back office and less in front-office applications [9], [23], [67].

e: PERFORMANCE PROBLEMS AND REDUCED RELIABILITY

A characteristic of old systems may be the occurrence of performance and memory problems when they no longer meet the increasing demand. This can be due to insufficient processing power, memory or network limitations, or poor software runtime efficiency [9], [43], [46], [48], [77], [p. 3f]79. In many cases, the reason for these problems is the growth of the application. In this case, it simply takes longer to load a vast program or accumulated business data into the main memory than for small programs or data sets. Poor design, which may manifest, for instance, in high latency, exponential program complexity, memory leaks, or bottlenecks, can also slow down performance and waste hardware resources [5].

Moreover, these systems contain essential operational information regarding a company's daily business and the abovementioned issues do not only apply to the operational tasks but also to the generation of reports [23]. Accordingly, the latter is hindered because these systems are often insufficiently integrated or contain inconsistent business data. The compiled information can often not be sufficiently linked; therefore, the content is fragmented and may contain contradictory information. Under such circumstances, reports can only be created with great effort, meaning manual reworking [53]. If the reports are time-consuming to produce and potentially contain erroneous or contradictory information, management cannot gain an efficient overview of a company's overall situation, which is detrimental to good decision-making.

As a consequence of the over-used hardware, errors can emerge. Accordingly, legacy systems are often, but by no means always, characterized by an increased error rate [5], [77]. Some outdated systems are characterized by many business disruptions, where the day-to-day business is hampered by constant disruptions [40], [77]. None of this is to say that all legacy systems are slow. Many legacy systems are reported in the literature to process transactions very reliably, robustly, and efficiently, which would be difficult to sustain in a successor [66], [82].

f: ORGANIZATIONS ARE HELD BACK, FURTHER BUSINESS DEVELOPMENT IS IMPEDED

A company must be able to adapt itself and its corporate vision to changing environments to remain competitive, which requires flexible IS [9], [23], [53]. However, legacy systems tend to increasingly resist change. Over time, these systems can no longer be adapted to the needs of the business units they serve with the necessary flexibility and cost. Business departments demand changes with growing frequency and urgency [55, 1, p. 20]. This situation is often accompanied by a lack of functionality in the legacy system needed to exploit the business opportunity and long backlogs reflecting the unmet demand [48], [57], [76]. Moreover, the IT department cannot deploy the latest technology due to maintainability problems, which in turn hinders an organization from benefiting from the latest technologies. By lacking changeability, legacy systems lead to inefficient and rigid organizations; they hinder or paralyze the evolution of organizations in the long term [5], [42], [43], [46], [53], [82]. *"Legacy systems are particularly important since they constrain rather than support the ability of the organization to respond to changing environment conditions or to adopt new strategies"* [10].

As required functions are not deployed, new business areas cannot be developed, process optimization has to be postponed and discontinued, or regulatory requirements cannot be met. At a certain point, the costs for the continued evolution and operation of these systems are no longer justifiable [5], [15], [40], [51], [60], [82]. This increased entropy stands in the way of the required flexibility. It must, therefore, be considered a bottleneck in the evolution of legacy systems [66]. Early versions of legacy systems tend to be developed with a high degree of adaptability, which they lose over time as structures erode and, accordingly, become increasingly petrified [61]. These mature IS tend to grow disgracefully by continued *"waves of hacking"* and eventually become immutable, meaning they suffer from outdated patterns and can no longer adapt to changing business needs [58]. Legacy systems with their poor maintainability and their technical obsolescence, constitute an organizational burden. They hinder the evolution of the entire enterprise [48]. Finally, outdated IS hold back the organization they serve as they cannot be changed or provide required functions at the required speed and cost [55].

In this context, legacies should be regarded as a severe burden, since they act as a barrier to achieving the company's vision. *"... legacy is rigid, and it is not flexible"* [9]. Thus, competitors may benefit from these competitive disadvantages: *"Your legacy system keeps your business from staying on the top"*[79, p. 3]. Accordingly, the main objective of modernization projects is to decrease costs and regain the required flexibility [9], [12].

The new IS being implemented today tie up existing resources (e.g., staff, budgets) for their development. On the other hand, typically, each year more resources are required for maintenance. These resources are then no longer available for the implementation of additional IS. If no effort is made to reduce annual operating and development costs, at some point there will be no budget available for innovation [49]. As much as 75% of the IT budget in insurance companies and banks is spent on maintenance [76]. It should be stated that legacies monopolize resources and experts [p. 3ff]79. In other words, legacy systems tie up an organization's scarce resources. The few legacy experts and budget available are often occupied with maintaining existing systems. This leaves insufficient capacity for the development of new IT applications or the redevelopment and evolution of existing ones. This monopolization slows down the evolution of the organization. Managers must, therefore, free up experts and resources so that they are available to develop new IT applications or modernize existing ones [49].

g: EVOLUTION AND OPERATION ARE EXPENSIVE

Over time, increasing efforts for development, monetary expenditure, and other organizational resources flow into the evolution of such systems [16]. As already mentioned, changes to legacy systems are becoming more time-consuming and risky. At a certain point, even minor changes and maintenance are considered expensive and cost-intensive. Consequently, the budget is quickly exhausted and not available for major redesigns or developments [49]. This means that countermeasures to improve maintainability and reduce costs are often too expensive [41]. In the case of externally developed or customized software, upgrades may have to be ordered and purchased from the original vendor at a high cost, which is (in the case of legacies) also often perceived as too expensive [53], [77]. At a certain point, the management considers the overall effort for the evolution of an information system to be too costly; at this point at the latest, the system is considered legacy [5], [40]. These high costs are one of the main drivers for initiating modernization (replacement) projects [9], [69].

In the case of legacy systems, the evolution leads to the situation of decreasing benefits per change. Each investment then realizes less and less marginal utility, especially if the investment is compared with a new development [49]. High costs for maintenance or evolution are therefore not necessarily a negative criterion in themselves, as long as the high costs are accompanied by adequate further development. Only if the high costs are due to inefficiencies should special attention be paid to this aspect [87].

Finally, it should be noted that the creation of reports is also described as time-consuming. In some cases, this is due to a lack of integration, computing power, quality problems in the data, and the need to prepare the reports, among other things. On the other hand, inconsistencies or incorrect reports can lead to incorrect processes and decisions. Thus, inconsistent and corrupt data also costs a lot of money indirectly [41].

h: CHALLENGING EVOLUTION AND ONLY REPLACEABLE AT HIGH RISK

A legacy system should be regarded as the confluence of two models, namely a business and an information technology model; both are intertwined and can, therefore, not be changed in isolation [53]. In addition, it should not be underestimated that the replacement of a legacy system is usually accompanied by the replacement of the associated evolution process. Thus, the software engineering method used, including the associated CASE tools, must usually be changed as well, since the old processes and tools are usually not compatible with the new technology [43]. Thus, for example, an assembler with the editors, compilers, and the structured programming associated with it is unsuitable for the evolution of an object-oriented paradigm such as Java. *"Even if the legacy system´s design is reused, it is unlikely that the original methodology and tools will also be reused"* [43]. All elements that make up an information system are intertwined. Changing one often affects the others, making it more complex to implement deeper changes [53]. This issue also explains why replacement projects are usually considered very demanding, risky and expensive. Hasty decisions regarding replacement later often prove to be an expensive and serious mistake [48].

Considering all of the above symptoms, the need to modernize or replace a legacy system with a new and more cost-effective alternative becomes obvious. However, this step is not trivial. In fact, these projects are considered some of the most challenging in industry and government [7], [9], and are not uncommon to fail in practice for several reasons [8], [51], [71]. For instance, some time ago the FAA[21] was asked to replace one of its legacy systems. The project had to be abandoned after 15 years without results and is considered a costly and protracted failure. Another example is the U.S. Air Force, which tried unsuccessfully three times over many years to introduce a modern logistics system to replace its outdated predecessor system. Ultimately, it cannot be overstated how complicated, expensive, risky, and dangerous the replacement of legacy systems is and that the explosive nature of such undertakings is very often underestimated [8].

[21]Federal Aviation Administration, United States Department of Transportation; https://www.faa.gov/.

In particular, far-reaching changes, such as reengineering,[22] modernization, migration, and replacement activities remain some of the most challenging tasks and require business and technical personnel [8], [12], [23], [77]. Migration, or the transition from a legacy system to a more modern counterpart, is highly uncertain and unpredictable, and influenced by many different factors, especially financial, technical, human, social, ethical, and cultural factors that make such a task complicated [64]. Owing to the lack of business and technical knowledge, changes in data management are also tricky. Data migration is described as a particularly difficult [12] or even the most challenging subtask when migrating legacy systems [51]. In turn, legacy business information can in many cases not simply be discarded or replaced and thus hinders the introduction of a new IS [41].

Furthermore, managements are more likely to deny a complete replacement of the old system by its successor (flush-cut approach) at one point in time (e.g. over a weekend) because of missing risk control opportunities and lacking progressive adjustment [79, p. 8ff], [16]. Consequently, a replacement must be typically done module by module, function by function, and element by element until all system elements are migrated. This step-by-step approach usually requires several years, meaning periods of 5-8 years, of simultaneous operation of the legacy and the successor system. Thus, a replacement is a long-term burden [79, p. 7].

One obstacle to the modernization or replacement of legacy systems is completing a project on time [9]. Moreover, a positive return on investment should be demonstrated beforehand, which can be challenging in far-reaching modernization projects. Projects that do not bring additional business value and bring only new information technology are typically not approved by managers [79, p. 8ff], [9].

If we look at the end of a system's lifetime, a flash-cut is in most cases not an option, as these systems are typically tightly intertwined with other system elements and at the same time suffer from high entropy[23] [55], [79]. Therefore, replacing a legacy system in one fell swoop, in a short time frame of a few days, or over a weekend, is risky and practically unfeasible in most cases [55], [75].

This is a kind of catch-22 situation for companies; often, referred to as *legacy dilemma*. On the one hand, legacy systems contribute significantly to the control of essential parts of the organization by implementing proven business models and (tacit) knowledge. On the other hand, they constitute a major obstacle to the evolution of the business, while any replacement is unpredictably expensive and risky. Many legacy systems were not able to be replaced despite extensive funding and time (sometimes decades of effort). Many of these replacement initiatives are in deep trouble or must be considered costly failures [7], [10], [44], [71].

[22]also referred to as renovation and reclamation [88].

[23]In a case study, Dhillon recounts an unsuccessful attempt to do such a redevelopment in parallel with the old application and an overnight switch that failed miserably [89].

In addition, it can be assumed that further requirements will arise during the several years the replacement is taking place, and this has to be taken into account, at least in part, in the old and new systems, which will significantly increase the effort [79, p. 7], [9]. Even the simple re-implementation of the legacy system is in practice often not feasible, as new requirements have to be implemented by the business during the usual 1-3 years re-implementation phase. In most cases, a business department will not accept going without new IT functions for years. Accordingly, *"Big rewrites fail"* [75].

Early legacy systems in particular were set up in such a way that they simply took over the manual work processes, most of which were not yet computerized (direct conversion). This means that business processes were not optimized for the use of IT, and, therefore, do not benefit from the use of IT to the extent that would be possible. To make matters worse, in practice some of the successor systems are business-replica, i.e. they merely implement the same processes in more modern IT. Despite new technology, sub-optimal processes that are still being operationalized do not exploit their full potential [46]. Therefore, simply migrating a legacy system to the latest keyword-compliant technology does not solve the legacy system problem; it merely migrates the legacy system to a technologically modern legacy system [42], [49], [54], [61]. If only the technology is renewed, the old business model remains. From a business perspective, such a purely technical innovation adds little (if any) business value. Accordingly, replacing a legacy system is not only a technical challenge; it's also a non-technical one [1]. This is mainly due to the fact that the previous business processes were often constrained by the old IT. A pure technical update to the latest shiny IT leaves the business model unchanged. If the business model is not reviewed in the course of the IT update, the corresponding business potential remains untapped [7].

Data migration, difficulties in testing, and complex eroded structures that resist modification are further heavy-weight technical reasons that hamper modernization or replacement initiatives [9]. Legacy experts for older technologies and experts for the latest technologies are needed together in the project, increasing the effort required [18], [43]. To make things worse, the new system may lack some of the required functions the old system provided [16], [43]. Thus, employees may have to work in both systems. Further, thousands of users must be retrained, which may result in high personnel absence and education costs [16].

In conclusion, it can be stated that legacy systems are often kept in operation, despite their shortcomings and uncertainties, to avoid the above-mentioned organizational strain and risks [18], [54]. In this case, companies face the typical legacy dilemma: *either continue to operate legacy systems at high and increasing expenditures and (too) slow evolution time or make a cost-intensive and very risky switch to a new solution* [1], [5], [15], [18].

D. REFLECTION

In the previous sections, we have used the knowledge base to develop a detailed empirical picture of the term 'legacy system' and the concepts behind it. In this conclusion, we will now summarize our findings and highlight and discuss the notable aspects.

First, strictly speaking, the term 'legacy' has neither a negative nor a positive connotation, but merely refers to *something being handed down by a predecessor*. A legacy, then, is something that is left behind after a certain event, typically the death of a person. In this definition, the term 'legacy' commonly implies a certain age, typically many years, often decades. The term 'legacy system'[24] first has to be considered in this context, with the exception being that a legacy system is typically taken over from another person who changes jobs or companies, retires, etc. In contrast to 'legacy', a legacy system is not related to the demise of the persons previously involved. In this strict sense, it is a neutral term refering to predecessor systems that have already been replaced or to existing systems that are still to be replaced.

Nobody writes a legacy system. Rather, legacy systems usually become obsolete unintentionally over time, and the characteristic negative features associated with them often emerge over decades[25] of operation. Having said that, the term 'legacy system' often has a negative connotation and is used in disparaging ways when weaknesses – often static or inflexible, and problematic IT that does an inappropriate job – are perceived and changes to them are being considered [42], [56], [62]. A legacy system is something old, at the end of its useful life, in the declining stage of its life cycle, or at a tipping point when it should be replaced, or when the successor is already on the horizon [7], [77], [78], [87]. They are often associated with high complexity, eroded structures, redundancies, and many different technologies; they are described as decayed, which means they are characterized by increased structural, technological, and functional entropy. This phenomenon was intensively researched by Lehman [39], [50] and has become known as the second law of software evolution. Sometimes, the aforementioned symptoms are also described as 'declining quality' (according to current ISO standards, for example [91], [92]).

The characteristics attributed to legacy systems vary significantly from author to author. In reality, however, these characteristics are by no means a consistent set of clear indicators of legacy systems. For example, while some legacy systems cause numerous disruptions in day-to-day business, others emphasize that their legacy systems are reliable due to years of troubleshooting, development, and proven

[24]To emphasize a positive connotation 'heritage system' was proposed in the USA and 'vintage system' in Europe. However, these terms have not caught on and have long since become legacies themselves [56].

[25]Some authors believe that legacy systems are emerging with each new technology that is introduced, which is approximately every 5 years. According to this view, application systems become outdated in less than 5 years [90].

technology and that proven solutions are deliberately preferred and are a key reason for not replacing them with modern systems. Some systems suffer from poor performance and outdated hardware. In contrast, others report excellent performance of their legacy information system [9], [69]. Some authors report that their legacy systems are highly integrated into the enterprise and serve as the backbone for enterprise-wide data delivery, whereas others see legacy systems as monolithic and insufficiently integrated [51]. There is not even consensus on whether an old programming language or old technology is a characteristic of legacy systems because they offer reliable and stable services that have been tried and tested in practice over many years. Only a fraction of employees confronted with legacy systems agree that a legacy programming language is a determining factor of a legacy [9]. Poor and outdated documentation is said to be a clear sign of legacy systems. However, in some cases, excellent documentation of legacy systems has been reported [17]. Moreover, legacy systems are often associated with outdated hardware or architectures, and with mainframes in particular. For instance, the IBM mainframe has become almost a synonym for 'old technology' [78]. There are few voices that state that a legacy system is one whose architecture has been compromised [10]. However, it should be noted that (especially) some mainframes are deliberately run because only they provide the required resources and stability [17]. Especially in old and large financial and insurance companies, mainframes together with COBOL are still widely used[26] [64], [72]. Therefore, technology in general should not be prematurely declared obsolete, especially the mainframe, which has been proclaimed dead for decades. In summary, on the surface, there is no common understanding of what attributes constitute a legacy system.

Finally, because of the large number of different, partly contradictory properties, we have concluded that typical statements attributed to legacy systems (poor performance, memory problems, degenerated architectures, lack of documentation, run on a mainframe [9], [43], [77]) should not be generalized in any case. Just as there is an unmanageable number of legacy systems, the properties attributed to these systems are just as varied.

Moreover, it is important to recognize that a legacy system is not inherently a uniformly negative entity implemented in a singular technology. In fact, the opposite is often true. Some system elements may be of good quality, others of poor quality, some may be young and modern, others old or outdated, and the rest somewhere in between [23], [82]. However, it would be inappropriate to view legacy systems as predominantly negative. Indeed, despite their shortcomings, legacy systems are frequently considered *valuable*. Therefore, these systems are often both a business asset and a business liability [58].

[26]There is also even evidence that IBM mainframes (as well as their smaller counterparts) still have their place and will continue to do so in the future [93].

Finally, the term 'legacy system' suffers from ambiguity despite its widespread use. The scope or definition of the term 'legacy system' varies widely from author to author [21], [78]. For example, some authors exclude hardware from the definition of legacy systems [17]. It often refers either to the application software only or to the entire information system in the sense of a socio-technical system [53]. A legacy information system is a broader concept that includes obsolete software in addition to non-technical issues (such as processes or business rules) [26], [56]. Others incorporate the evolutionary process that drives the dynamics of legacy systems into their considerations [62]. In addition, practitioners and academics regard and use the term differently. The former tends to focus on business and capabilities, while the latter focuses on technical issues [9].

Consequently, there is no common use. The interpretation of the term should therefore always be questioned. This wide diversity of usage hampers communication and action [10], [48]. Moreover, the concept of legacy systems partially overlaps with the various concepts of *debt* (e.g., technical debt, social debt, architectural debt) [75], *legacy health* [47], or *legacy status* [22, p. 8]. Depending on the underlying usage of the term legacy systems, the concepts may even overlap completely and describe the same phenomenon [74], [78]. Such systems cannot be seen purely as a technical problem, as they encompass and touch upon many other areas including business, organizational, technical, human, and strategic elements [9], [55], [56], [61]. Fortunately, for at least the last 20 years, a consensus has emerged that a legacy system should be viewed from both a business and technical perspective [42], [47]. Further, some argue that the legacy problem is more an organizational than a technical one [63].

Overall, the literature reviewed is lacking in stringently applied research methods. Most articles present their statements in an anecdotal narrative style. Accordingly, the statements attributed to legacy systems are often vague, undifferentiated, and vary from author to author. Legacy systems, for instance, are often described as large, but without providing any further details. This is difficult because, firstly, 'large' is relative and, secondly, it is not clear what it refers to (e.g., business data, source code, number of users, business processes). Another example: The documentation is often described as outdated and incomplete. However, it is usually not clear what type of documentation is meant (e.g., source code documentation, user documentation, IT operations documentation, or design decisions with a conceptual model). Like 'large', the term 'obsolete' also refers to a spectrum ranging from low to very high (obsolete). Therefore, the term 'obsolete' should also be regarded as a relative word and is consequently imprecise. Finally, unsubstantiated and unspecific claims (such as that legacy systems are difficult to understand, suffer from high entropy, and are poorly documented) are widespread. Such bold statements should be taken with a grain of salt.

Remarkably, some properties attributed to legacy systems are highly present in the reviewed literature, whereas others are underreported. For example, a lack of IT experts and domain knowledge has been regularly mentioned and deprecated software is addressed much more than legacy hardware. Software growth is also regularly cited in the literature, but the increase in business data and number of users is rarely discussed. Accordingly, this paper reflects these proportions. Furthermore, it could be assumed that legacy systems with discontinued technologies have many unresolved security gaps and that cyberattacks should be a relevant topic [17]. Surprisingly, this security topic was hardly present in the reviewed literature.

IV. EXPLAINING THE BUSINESS-TECHNOLOGICAL AGE OF INFORMATION SYSTEMS

In this section, we revisit the statements made in the previous section and abstract them to explain the underlying legacy phenomenon. To this end, we first give our approach to explain LIS. In the second part, we demonstrate the viability by mapping the elaborated statements to the model. The third part discusses and evaluates the model and draws conclusions.

A. THE ARTIFACT TO EXPLAIN THE LEGACY STATUS OF LEGACY SYSTEMS

We assume that legacy systems can be explained using a combination of socio-technical theory and the viewpoints approach. Having already explained the socio-technical theory at the beginning, we now turn to the viewpoint approach. In this approach, different people in their different roles generally adopt different, very subjective perspectives on the legacy system in which they are involved [64]. This concept of explaining legacy systems assumes that a (legacy) information system should be evaluated from different points of view. These result from the roles (like managers, business analysts, software developers, database specialists, direct system users, customers, and operators) and profiles (knowledge, education, opinion, etc.) of the persons involved and their respective views of the legacy system. Even for one criterion (such as documentation quality) for just one information system, the assessment differs from expert to expert, in some cases significantly [76]. The assessment of the individual attributes of a legacy is accordingly highly subjective (at least without a fixed parameter). The classification as a legacy system is an individual view or assessment of an information system and can change from person to person [26], [56]. A manager, for example, is typically more interested in the capabilities and cost of an information system than in its internals. In contrast, a developer is typically much more interested in modern information technology, quality code, modern CASE tools, and development methods than in the external functions and the benefits it brings to the organization. A business analyst, on the other hand, is interested in lean and efficient processes that come close to his ideal and may be modeled for him in modern notations (such as BPMN).[27] and tools. Therefore, which factors indicate a legacy system is strongly dependent on the viewpoint (technical, social, organizational, strategic, developmental) of the observer [10], [56].

The aforementioned viewpoints are interrelated. For instance, a strategic decision to enter a new market results in the organization's employees having to change the organization; they have to create new business processes and rules and evolve the existing ones. This, in turn, may influence the development perspective and subsequently, the operational perspective, which must correspond to the new business model. Accordingly, these viewpoints form a cascade that builds on each other; nevertheless, the subordinate viewpoints can contribute to the higher-level strategic perspective [10]. However, we argue, that this is not an exclusive characteristic of legacy systems, but for IS in general.

It should also be noted that there is also the 'capitalist view' or the 'populist view'. This meaning includes or refers to the understanding of vendors of technologies or solutions. These disparagingly refer to an earlier version of a product or technology as legacy systems (even though they may still be absolutely modern and do their job perfectly) to make them look old-fashioned with the aim of marketing and selling their own products [56]. This view of vendors only serves to malign other products and is not very useful in understanding or explaining legacy systems, yet it is very common in glossy magazines [26].

The model to explain legacy systems is illustrated in Fig. 5 (illustration based on Migley [95]). The legacy system is displayed within the *Information System Boundary* (Fig. 5, left). This outdated system is deployed for an intended operational task such as accounting, product data management, or production planning and scheduling. The system is operated by *Direct System Users*, who are primarily domain experts, clerks, line managers, and IT personnel, who perform their functions in various roles and use the application system expediently for their daily business. The *Information System Boundary* also includes the evolutionary processes. Generally, the evolution process has run through several times over many years. With each iteration, a new increment of the information system is created, which is then the input for the next evolution process run-through. The *System Designers* involved are typically business analysts, software developers, usability engineers, testers, and ultimately all staff involved in the evolution (we also include their development tools like compilers, linkers, development environments, software engineering methods, notations such as UML,[28] BPMN, programming languages, and storage tools such as repositories for developed artifacts).

In principle, an information system does not have an end in itself [32, p. 79f]. Instead, the purpose of a system is set by the business goals of the organizations it serves [15]. Obviously, this statement also applies to legacy systems. A legacy information system (with its *Application System*, meaning all technical equipment and the technology behind it, the

[27]Business Process Model and Notation.

[28]Unified Modeling Language.

TABLE 4. Mapping of legacy properties to systems elements.

		Characteristic or Statement	Refs
Time	(1)	longevity, created sometime in the past, decades ago, emerges at the earliest, after 3-5 years, typically at about 10+ but is often 20-30 or even more years old, is still operational	[1, 8–10, 13, 15–18, 23, 26, 40, 42, 43, 48, 51–53, 60, 62, 63, 67, 72, 73, 75, 76, 78, 94]
	(2)	evolution becomes more difficult (costly, time-consuming, etc.) over time	[1, 5, 8, 11, 57, 58, 66, 76, 78]
	(3)	have incrementally evolved over a long time to meet new, changed requirements and contain, therefore, many modifications	[1, 9, 11, 51–53, 60, 66, 76, 78, 82]
	(4)	is becoming increasingly outdated over time	[1, 8, 11, 55, 76]
Technology	(5)	consist of / built with previous or obsolete technologies (programming languages: like 2G, early 3G, or procedural languages, data storage: flat-file data-storage, IMS-data base, data transport: file and batch-based data transfer), outdated program versions)	[1, 8–10, 12, 16–18, 23, 41–43, 52, 53, 58, 60, 63, 64, 67, 70, 72, 75, 76, 78, 82]
	(6)	obsolete, lack of applied standards, non-standard or no software engineering methods applied (like, development without conceptual data model, no test available), deprecated CASE tools applied (in extreme cases, no IDE only compiler and linker is used) CASE tools are used), use of obsolete notations, use of home-made IT-artefacts (like self-developed programming languages, databases, or memory management)	[13, 15, 40, 41, 43, 44, 48, 53]
	(7)	limited remaining availability of suppliers/vendors; discontinued support of applied artifacts (such as hardware, software, technology)	[9, 40, 53, 67, 69, 78]
	(8)	optimized code and architectures to compensate for missing hardware resources, or to improve performance or availability	[1, 43, 44, 48, 53, 66, 70]
	(9)	use of deprecated hardware or electronics, outdated OS or an old version of it (such as elderly mainframe, floppy discs)	[8, 10, 40, 42–45, 54, 73, 76, 78]
	(10)	difficult to keep hardware operational, difficult to obtain spare parts and service, spare parts unavailable at all	[10, 43]
	(11)	hardware does not provide enough resources	[43–45, 48, 53, 54, 70, 73]
	(12)	documentation task neglected, missing or poor documentation, undocumented data structures, lack of specification, undocumented business processes and rules, the code is the only reliable source of information, submerged business processes, the business model is not mapped to IT artifacts	[1, 5, 9, 10, 12, 16, 23, 26, 40, 43–46, 48, 52, 54, 55, 58, 60, 63, 67, 73, 75, 76, 94]
	(13)	increased entropy in structural (eroded, degenerated architecture, code smells, increased complexity), technological (variety of different technics), or functional (redundancies, bloated code and code clones) dimensions	[11, 16, 18, 40, 41, 43, 44, 46, 48, 49, 52, 53, 55, 58, 61, 62, 66, 70, 73, 76, 78, 82]
	(14)	obsolete, inflexible architecture, that does not meet current standards, poorly layered or monolithic architectures with the absence of clear interfaces, inappropriate architecture (like mainframe instead of client-server)	[8, 9, 12, 16, 23, 40, 52, 55, 57, 58, 61, 62, 66, 67, 69, 76, 78, 82]
	(15)	difficult to integrate, lack of interoperability (due to missing interfaces, ad-hoc integration using glue code, for example	[7, 12, 23, 26, 44, 45, 54, 58, 67, 73]
	(16)	isolated, not properly integrated into the enterprise-integrated applications, or databases	[10, 18, 40, 41, 53, 67, 69, 78]
	(17)	operational complexity, lack of portability	[55, 58, 66]
	(18)	reduced reliability, day-to-day business hampered by constant disruptions, increasing error rate	[5, 26, 76, 77]
	(19)	large in terms of application size (often more than 1 million lines of code) and provided functions, increasing size and functionality over time, many databases and tables (100+ tables)	[1, 5, 26, 40, 52, 55, 57, 63, 72, 73]
	(20)	large in terms of business data	[40, 41, 51, 63, 73]
	(21)	corrupted, redundant, inconsistent, and widely dispersed business data	[16, 40, 41, 51–53, 70]
	(22)	deprecated (character-based, old-fashioned, green screens) UI, limited usability and user experience, one-screen-per-table-UI	[61, 63, 69, 73]
	(23)	security issues	[8]
Human	(24)	fragmented knowledge, no holistic understanding	[5, 9, 16]
	(25)	large in terms of users (1000+, frequently considerably more)	[16, 23, 41, 67, 82]
	(26)	unsustainable conducted changes (quick-fix approaches, workarounds, documentation neglected)	[1, 5, 9, 40, 48, 55, 58, 60, 61]
	(27)	deployment of new features or changing the implementations requires downtime	[61]
	(28)	resistance from users impedes modernization or replacements, experts do not want to share their knowledge	[1, 64, 69]
	(29)	corporate culture lacking in innovation, weak co-evolution, organizational resistance to renewal	[9, 53, 55, 61, 64]
	(30)	aging workforce with limited remaining legacy experience, whose retirement is upcoming	[7, 66, 69]
	(31)	reengineering was neglected, no undergoing systematic remedial actions, management inaction	[1, 11, 60, 61]
	(32)	lack of legacy experts due to multiple handovers, insufficient knowledge about systems implementation, employment of poorly instructed employees and lack of training by the employer, lack of qualified staff with experience in the legacy business model and applied technology, qualified personnel is difficult to obtain and expensive, original developers are not available anymore	[1, 7, 9–13, 16, 40, 43–45, 48, 51–53, 55, 57, 61–63, 70, 72, 73, 78]
	(33)	hard to understand, THE intellectual capacity of developers is overwhelmed, understanding the program is a major task, the impact of changes is no longer (fully) predictable, possible change anxiety, use of inscrutable, idiomatic variables and module names	[1, 5, 9, 12, 15, 42, 43, 49, 52, 54, 58, 61, 63, 70, 73, 78]

Direct System Users, and the *System Designers*) has a relationship to the context (that is the organization), meaning the respective purpose, *the IS demand formulated by the business activities* (referred to as the internal drivers of change) for which it is intended and used. These demands may be, for example, the result of company mergers, horizontal

TABLE 4. *(Continued.)* **Mapping of legacy properties to systems elements.**

Business	(34)	resist modification, profound and even small changes are difficult and error-prone, maintainability problems, hard to extend, upgrades are difficult or impossible, (non-extensibility, lack of openness due to missing interfaces, features cannot be added by plugins, brittleness, inflexibility, non-configurable, incomprehensible code, etc.), evolution (testing, fault detection) is demanding or impossible, no or insufficient test base, long build processes	[1, 5, 7, 9, 10, 12, 13, 16, 18, 23, 40, 42–45, 48, 49, 53–55, 57, 58, 60–63, 66, 67, 69, 70, 73, 75, 77, 78, 94]
	(35)	provide consolidated information about the business, and the organization (such as sales, spending, and earnings)	[44, 53, 54, 63]
	(36)	tasks take a considerable long time, slow response time, performance problems, increasing response time	[5, 9, 40, 48, 63, 73, 76, 77]
	(37)	slow evolution, (too) a long time to market, changing the solution (adding new features, fixing a bug) is time-consuming	[9, 10, 44, 45, 48, 53, 55, 60, 73, 82]
	(38)	lower productivity and competitive disadvantage (due to missing capabilities, poor performance, inconsistent data, for example), impede further business development, business opportunities are being lost, organizational inflexibility, risk of business failure	[1, 5, 8–10, 13, 16, 26, 42, 46, 53, 55, 58, 60, 63, 67, 69, 72, 94]
	(39)	lack of reliable business intelligence, often long provisioning time, and manual interventions required	[13, 23, 26, 40, 53, 63, 67, 73, 94]
	(40)	lack of required functions, unsatisfied functional requirements to support the desired business model, a long, growing backlog of unimplemented requirements a (potentially increasing) disparity between legacy systems and IT or business strategy	[1, 9, 10, 26, 48, 52, 53, 55, 57, 60, 62, 67, 76, 78]
	(41)	(software) further development is (unreasonably) expensive, consumes a great amount of the available budget, budgetary constraints may be exceeded, it is difficult to find cost-effective and at the same time quality solutions	[1, 5, 7–10, 12, 13, 15, 17, 41, 44, 45, 48, 49, 51–55, 58, 60, 66, 69, 70, 73, 76, 77]
	(42)	hardware is expensive to maintain (due to old or scarce spare parts)	[10, 43–45, 54]
	(43)	emerges often with a high volume of repetitive data processing, like payroll, payments or account posting, and personnel issues, likely to be found in the back office less in the front office	[9, 18, 44, 55, 63, 66, 67, 72, 78, 82]
	(44)	hard (i.e., costly, risky, time-consuming) to replace, migrate	[1, 7–9, 16, 43, 44, 48, 58, 64, 67, 73, 75]
	(45)	hinders the introduction of new information systems (due to monopolization of scarce resources), the further digitization is hampered	[8, 41, 49]

and vertical integration of suppliers, and customers and their IS.

The outer area beyond the *Organizational Boundary* in Fig. 5 represents the context of an organization that operates legacy information systems. External threats (such as new regulations or changed customer needs) are brought to an organization by its context. Advances in science and technology should be regarded as the main driver of technological aging. These are, in particular, the basic sciences constituting IS, namely business administration and computer science, as well as inter alia, sociology, law, and psychology. The *Current State of Science and Technology* is placed in the organizational context in Fig. 5. This is based on the assumption that innovations in science and technology are largely researched outside an organization (often at universities), although some (but not the majority) of innovations can also be researched within the organization.

The internal and external requirements of the company and the current state of technology are interdependent. *"Information technology should not be limited to just supporting a business strategy. Instead new technology should directly influence the strategic direction of organizations"* [53]. Thus, technological conditions and external requirements can influence corporate strategy. For example, using digital signatures to save expenditures and protect the environment could be an example of a business or legal requirement that is only feasible if appropriate knowledge about asymmetric cryptography is available in science. As long as cryptographic procedures have not been invented, digital signatures cannot be set as a business requirement or administrative regulation. In addition to the aforementioned requirements, existing IS – in the sense of E-Type-Systems mentioned before – drive their evolution and their respective sciences.

When a company is unable to bring the legacy system in line with the requirements, a gap between the system, the technology, the business strategy, and the business model emerges and widens. Furthermore, the larger the gap between provided capabilities and the demand of the business department, the more outdated the system. Moreover, a legacy system should be regarded as an instantiation of a specific state of technology. Similar to unimplemented requirements, if the information system is not adapted to the current state of scientific knowledge and technology, the system becomes outdated from a technological perspective. When a system can no longer keep pace with the constantly changing demands (which are typically expressed by a long and increasing backlog) and falls further and further behind what is required, we label it a *legacy system* [26], [53], [56]. In Fig. 5, the potential gap between the legacy system and the requirements from a business and technical perspective can be recognized by the distance between the *Application System* or *Information System* and the (unsatisfied) demands resulting from the involved roles (Direct System Users, System Designers, Managers). Thus, it can be stated that *the less the legacy system can meet business requirements and the more it falls behind the current state of science and technology, the more outdated the system under consideration is.* To illustrate this definition, we propose the term *"business-technological age"*.

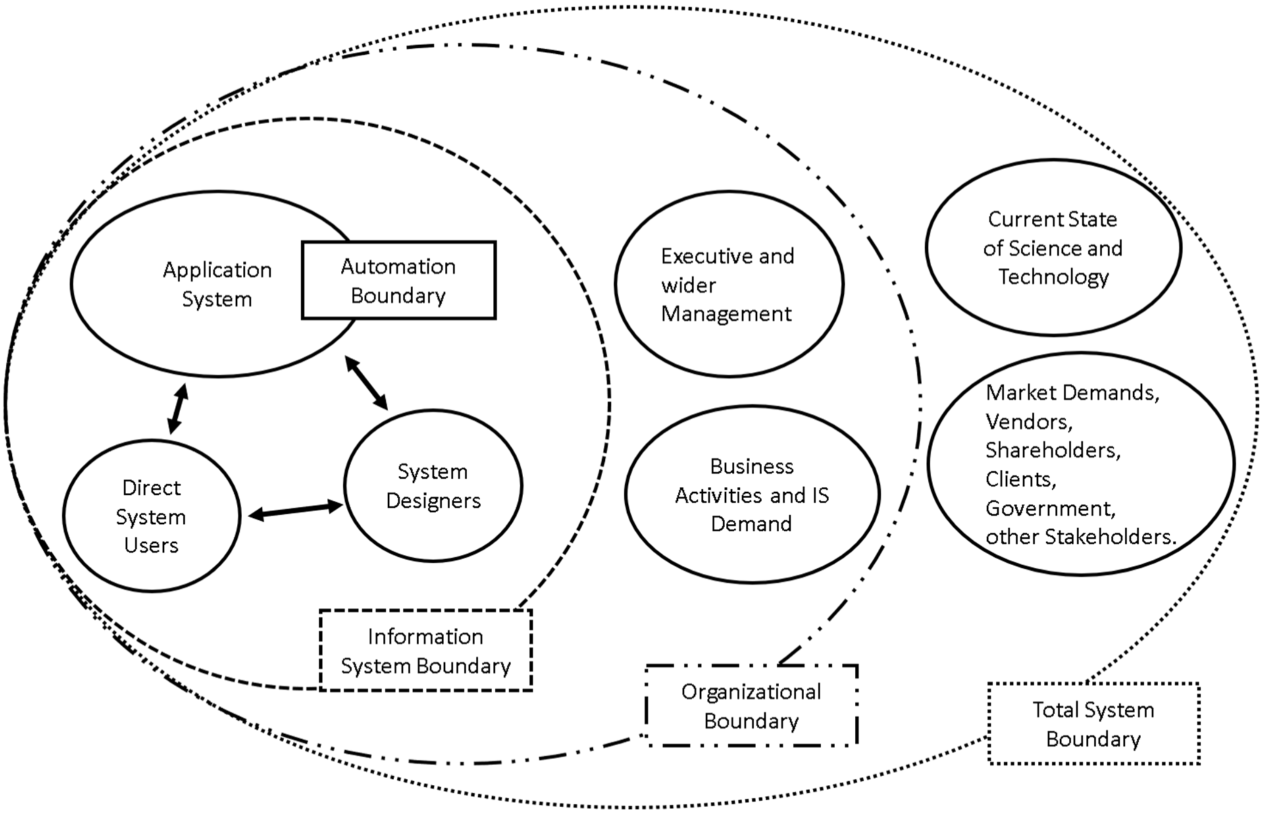

Application System
Automation Boundary
Direct System Users
System Designers
Information System Boundary
Executive and wider Management
Business Activities and IS Demand
Organizational Boundary
Current State of Science and Technology
Market Demands, Vendors, Shareholders, Clients, Government, other Stakeholders.
Total System Boundary

FIGURE 5. A legacy system can be explained by understanding it as a socio-technical system that has fallen behind the subjective expectations of the various stakeholders (users, designers/developers, managers, etc).

B. DEMONSTRATION & EVALUATION

The chosen design science research approach stipulates a demonstration and evaluation of the artifact. A deductive research approach was used for the demonstration. The subsequent evaluation was performed using argumentative-deductive analysis [33]. In the following, we will first discuss the demonstration and then the evaluation.

The model we propose claims to be able to explain all legacy systems. Accordingly, this model should also be able to represent at least the empirical legacy system properties found in the literature. In order to provide this proof, we have projected the properties of Table 4 onto the model, provided that the model can also explain them (cf. upper section of Fig. 6). It can be stated here that the model can explain all legacy system properties from Table 4 with the exception of the time-related statements (1-4).

In addition to business-technological age, we recall that age exists as a chronological (time-related) quantity resulting from the passage of time, evolution, and emergence of outdated systems over time. However, the underlying socio-technological theory is not designed to represent the temporal dimension. Accordingly, the model is incomplete, as is the underlying established theory, and the proposed artifact cannot explain time-related statements in principle. Time progression can only be estimated in the model. For example, outdated technologies in the legacy system usually emerge only after several years or decades. We will return to this point in the discussion and will now evaluate the proposed model via the argumentative-deductive analysis.

Firstly, the abovementioned strategic viewpoint is concerned with changing the company's strategy and with the related costs and capabilities of IS. This view primarily involves managerial roles. A legacy system from a strategic viewpoint is one whose financial benefit is less than the cost of operating and maintaining it. Similarly, a legacy is a system that cannot support a desired business function, thus preventing an organization from taking advantage of desired business opportunities.

Secondly, the organizational viewpoint (the view of the systems designers) is concerned with defining, changing, and improving an organization, thus, in particular its business processes. A typical legacy system in this viewpoint is a system that is *"old, inflexible, expensive, non-portable and undocumented but indispensable because they support core business functions"* [10].

Thirdly, the operational viewpoint (represented by *Direct System Users*) focuses on the efficient delivery of services with an appropriate response time. A legacy in this category is a system that does, for instance, not meet the demands mentioned before, thus it is too costly to operate, is hampered by interruptions, or only provides a poor user interface. If the security has been compromised or it utilizes hardware or

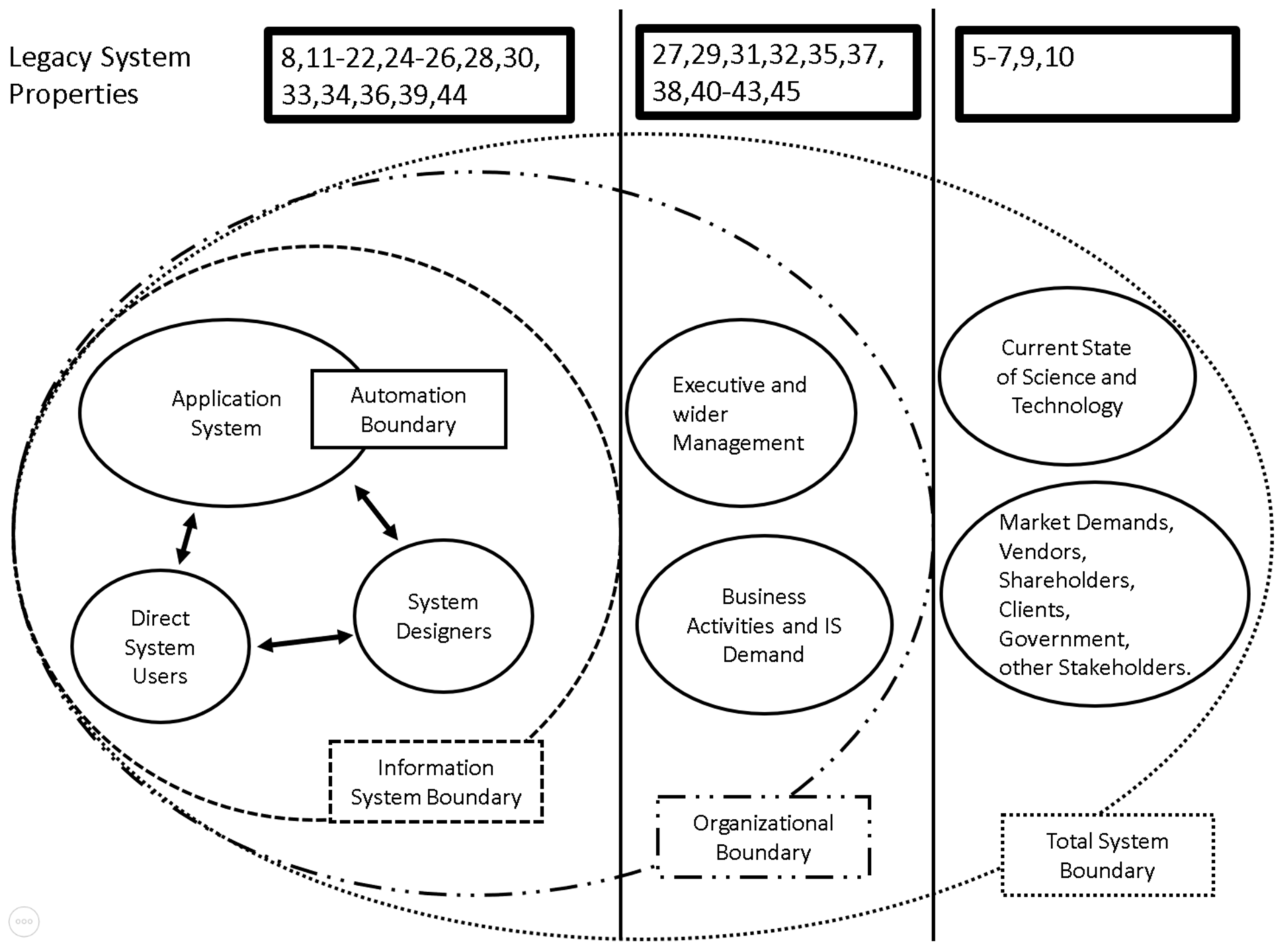


FIGURE 6. **Mapping of the legacy system properties found to the model to explain these systems.**

software that is not (anymore) in line with the strategy, it also must be considered legacy.

Lastly, the development perspective deals with the development and evolution of legacy systems and includes roles such as software developers or usability designers. Typical definitions in this category address the high entropy of the application, the lack of knowledge and documentation about the system internals, and the lack of legacy experts [10].

High entropy is a sign of an outdated information system. However, this property cannot be read directly in the model, because the degree of abstraction is too high for this. Nevertheless, the model explains the entropy problem indirectly, because low entropy (typically formulated as clarity, intelligibility, abstraction, or high cohesion and loose coupling) is postulated in business informatics (which is part of the *Current State of Science and Technology* in Fig. 5) as a property of well-designed artifacts [96, p. 255f], [97, p. 270ff], [20, p. 28]. Therefore, the current state of technology implies low entropy. Furthermore, the more an information system deviates from these objectives through increased structural, functional, and technological entropy, the more deprecated it becomes. This aspect of aging is only implicitly explained by the model, as the level of abstraction is too high for this; (low) entropy is not explicitly present as a model element. Thus, it can be that a single technology is considered outdated in one scenario and state-of-the-art in another [26]. Accordingly, it is impossible to determine the condition by only looking at the obsolete system in isolation. A legacy system assessment must consider the specific context or organization [12], [23], [67].

In the context of this study, the purpose of the model is to explain legacy systems and make better decisions regarding them. Based on the proposed model, we answer and discuss the research questions below and at the same time demonstrate the scientific validity of the artifact.

RQ-1: How can the (business-technological) age of information systems be explained in general?

Firstly, business-technological age is explained as a significant gap between the legacy system and what is needed by the roles or participants involved; secondly, by the deviation from the current state of science and technology; and thirdly, the aforementioned gap (typically documented in the backlog) must be widening for the legacy system to fall further behind the needs over time.

Following the model approach, an information system can satisfy different needs (requirements and technologies) to different degrees. For example, a deprecated system that perfectly fulfills all business requirements but at the same time implements deprecated information technology would therefore only be outdated from an IT perspective but at the same time be modern from a business perspective.

RQ-2: What are the factors that lead to the aging of IS?

Over time, an information system becomes deprecated to the extent that it is not adapted to the given requirements. The dynamics of the business strategy and the business model derived from it and implemented in the information system have a significant influence on aging. In a relatively stable environment where few requirements arise, evolution is relatively simple. In an unstable, very dynamic environment on the other hand, evolution is much more difficult in order to keep the information system in line with the business strategy [53]. The risk of emerging legacy systems is greater in technologically evolving environments. IS may age more quickly under these circumstances [49]. Finally, we see *change* as the primary (if not the only) cause of aging.

Another essential criterion, as mentioned above, for being able to speak of a legacy system is that it falls further behind the needs. One reason for this is that business and technical change requests pile up because the evolution process cannot keep the associated information system up to date. The throughput (for instance, due to a lack of personnel, lack of knowledge) of the evolution process of requirements is too low.

Accordingly, *an insufficient evolution process is to be understood as a factor for the aging of IS*. Conversely, if there is an excellent evolutionary process, it is difficult for a legacy system to emerge because the information system is promptly aligned with new requirements and technologies without increasing entropy. The previously mentioned criteria for legacy systems with a long backlog cannot emerge in this case. Following Sneed [98, p. 24], it must therefore be true that *an information system becomes outdated to the extent that it no longer keeps pace with the required changes by the organization.* This describes a situation where the backlog is growing and requirements are changing or emerging faster than the rate of development to catch up [61]. A large number of requirements, meaning a large backlog alone, is not sufficient to constitute a legacy system since many newly developed systems often still have a large backlog in their first early versions. In this context, an increase in the backlog (which is unlikely to improve) is an essential criterion for defining and valuating legacy.

As shown above, legacy systems and evolutionary processes influence each other. A legacy system is an essential input (and output) of the evolutionary process and its properties significantly influence evolution. A system characterized by high entropy, which is difficult to understand, obviously resists modification even in a well-set evolutionary process. We understand the high entropy of a legacy system as a self-reinforcing factor that contributes to further decay. We conclude that *the condition of the legacy system should be considered as another factor contributing to aging.*

In summary, the following statements can be made:

(1) *Change is necessary but not sufficient for IS to become outdated.*

(2)*The higher the change dynamics of a system's context or constituent technology, the higher the risk that it will become outdated.*

(3) *The (business-technological) age of an information system can never be assessed independently of its context.*

(4)*A poor evolutionary process accelerates the emergence of a legacy system.*

(5) *A poor condition of the legacy system contributes to further deterioration.*

RQ-3: Is the lifetime of the product related to obsolescence?

We found that time is necessary for change, as changes occur over time, rarely overnight. For example, new laws and regulations, technical innovations, scientific discoveries, and advanced production processes have been developed or improved over many years. Revolutions or disruptive events also occur over time, even if they occur over a shorter time frame. The dependency between time and change is apparent. *Time is a necessary precondition for change.* Moreover, as mentioned before, *change is an essential prerequisite for aging.* These statements lead to the following causal chain: time elapses → changes may occur → changes are not implemented or implemented unsustainably → the system ages → high entropy impedes evolution. This system of relationships should be regarded as a vicious circle, as shown in Fig. 7. In summary, it can be said that poor engineering initially impairs maintainability. As a consequence, requirements cannot be implemented on time and not at a reasonable cost, which in turn, causes an increase in poor engineering. Parnas [5] named these two factors a *"double punch"* that puts IS out of business over time. This means that IS increasingly develop resistance to further adaptations to the ever-changing business requirements (first punch), which in turn prevents such systems from being adequately adapted to the requirements placed on them (second punch) [58].

In conclusion, time must be considered as a risk factor for aging. The more time passes, the greater the probability that changes will occur to which the legacy system and its evolutionary process are not sufficiently adapted. Consequently , *it is impossible to infer directly from the chronological to the business-technological age of those systems.* However, it is also possible that an information system operates in a very stable context, in which no or only very few changes occur or the system has been handled with perfect care so that it is not affected by the aging drivers. This leads to another insight – *time is a necessary variable for aging, but not sufficient on its own.*

This statement is in line with Demeyer's empirical observation that legacy systems do not necessarily have to be up in years if they are not carefully maintained [19, p. 3].

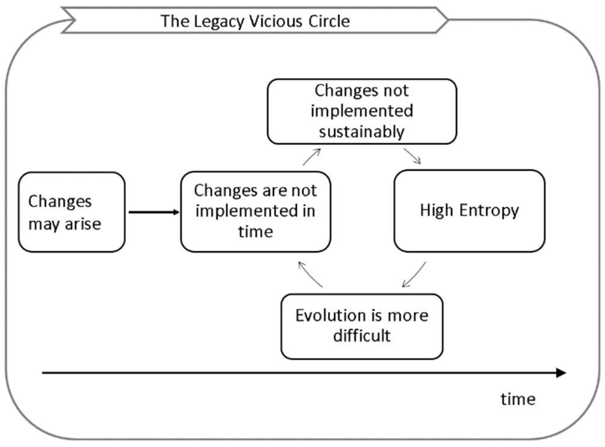

The Legacy Vicious Circle
Changes may arise
Changes are not implemented in time
Changes not implemented sustainably
High Entropy
Evolution is more difficult
time

FIGURE 7. **The vicious circle of legacy systems.**

The aging of IS perhaps analogous to that of human beings – with an unhealthy lifestyle, we age faster biologically and our medical condition deteriorates quickly. Yet, in contrast to humans, business-technological age can be substantially reduced through reengineering measures and reduction of the backlog requirements. At least theoretically, there is an opportunity to rejuvenate legacy systems by catching up and reengineering. However, other factors such as high costs or lack of management initiatives seem to hamper the rejuvenation of systems in reality and thus lead to the large stock of legacy systems mentioned at the beginning of this paper.

RQ-4: What are the symptoms of legacy systems?

The empirical characteristics attributed to legacy systems differ significantly, sometimes diametrically, from author to author. This is presented in Table 4. In our view, legacy systems are further behind the curve. They are characterized by high entropy, stubborn resistance to change, and are therefore unable to keep pace with change dynamics and fall increasingly behind the demands of the various roles involved. Despite this, such systems can be used to govern daily business and cannot simply be replaced as they are highly intertwined with the organization they serve.

The symptoms of a legacy system are typically not equally manifest in all modules but vary depending on the module. Their condition typically range from poor to excellent. Thus, the model explains age as a multi-faceted characteristic, which is consistent with the thoughts of Bennet and O'Byrne [1], [21]. Consequently, the presence of an old system cannot be broken down into an objective and generally globally valid binary statement [21], [56]. Depending on the symptoms and their severity, it makes sense to ask to what extent a legacy system has already emerged [22, p. 32f, p. 179], [21], [60]. If the status of a legacy system is reduced to a few metrics, the use of balanced scorecards seems to be a suitable tool [23].

C. DISCUSSION

We have presented a model that can initially explain legacy systems and their drivers in principle. Although the artifact presented answers the research questions and fulfills the objectives of the solution, it cannot provide a holistic model to explain the time and spatial-related characteristics of legacy systems. Based on the underlying theory, we pretend in any case that our IS are timeless. However, this was not the case here. Time is undoubtedly a factor in the evolution of IS. Thus, the emergence of a legacy system can only be fully explained if time is included as an influential variable in explaining the phenomenon. The 2000 problem, for example, can only be explained when time comes into play [53]. To fully describe any phenomenon (and thus also legacy systems) of reality, the time and space in which the phenomenon occurs must be included [32, p. 95]. Whether time and even space, the location of an information system, should be integrated as an additional dimension into formulating the underlying theory should be discussed in the future. Without improving the basis for the theory, the dynamics of change per unit of time, time-to-market, or simply the chronological age of a system can only be represented outside the model, as shown in Fig. 7, for example.

The attributes 'old' on the one hand and 'deprecated' on the other are often used synonymously. Sometimes old is used to describe deficiencies in terms of quality [16]. However, we argue that the meanings of these signifiers should be strictly separated. 'Old' is, in our view, an attribute that refers exclusively to the life span; that is, it is based on a chronological, time-based reference. In contrast, 'deprecated' refers exclusively to the state or condition of an artifact. The term 'deprecated' is correlated with high entropy, a lack of capabilities, and the use of technologies that no longer reflect the current state of science. This way of looking at things, in which the time and technical condition are considered separately from each other, makes it possible to distinguish between chronological age and the condition of an artifact. We recommend considering these different terms and meanings to improve scientific rigor.

A coarser level of abstraction based on the main elements (task, human, and technology) was selected for the model. Deeper elements, such as data, applications, hardware and networks are abstracted from the artifact. Consequently, the mapping performed during the demonstration could not directly assign some features to their respective sub-elements but only to the abstraction level above. For example, the statement, 'the business data is redundant and inconsistent' could not be assigned to a business data element directly but only to the *Application System* as a whole. Other elements of interest, such as entropy, redundancy, and architecture are only implicitly covered by the model. It would be helpful if the model could be further refined without reducing comprehensibility through too much detail.

As described at the outset, business informatics focuses on the intertwining of business administration and communications technology. Thus, the scope of the model was intentionally narrowed. This limitation is also reflected in the primary factors of aging, namely business management and technical requirements. Nevertheless, other factors (e.g., social, or cultural) also influence the aging of IS. Such influencing factors are, in principle, included in the model via the human component but are not empirically validated, as such issues are rarely discussed in the found literature.

Furthermore, the expressions 'condition,' 'business-technological condition,' or 'business-technological status' are conceivable. Moreover, the terms 'obsolescence' and 'deterioration' are occasionally used [60], which likewise refer to this issue. Additionally, other terms may be possible. Because humans think about concepts, associations, and images, well-chosen expressions support understanding and communication [99]. Therefore, to develop a model and thus an understanding, we recommend further research to determine which term most accurately describes the proposal we have developed.

V. CONCLUDING REMARKS AND FUTURE RESEARCH

Today's new and modern information services usually become tomorrow's legacy systems, as adaptations to the ever-changing world cannot be fully implemented, changes are often not sustainable, and remedial action is often omitted [1]. Accordingly, we agree with the statement that *"all technology products become a legacy"* [64]. Even though the phenomenon of legacy systems has been known since the 1970s and has been intensively researched since then, there is still no holistic concept to explain the aspects and dynamics of such systems. Accordingly, legacy systems are typically described one-sidedly by their characteristic symptoms, such as poor documentation, code smells, and high complexity. The term is often subjectively connoted according to the intention of the respective author. Finally, each author regards these systems differently, which in turn complicates comprehension, communication and decision-making in relation to these systems. For instance, knowledge of the characteristics and dynamics of legacy systems is valuable because it provides clues and starting points as to where and what measures need to be taken [61].

To understand legacy systems, we first conducted an extensive literature review and identified empirically observable phenomena in legacy systems. The key finding is that legacy systems have fallen behind what is required, which is, for example, a lack of business capabilities, the use of outdated programming languages, user interfaces, or storage technologies often accompanied by long backlogs, high entropy, as well as documentation deficiencies. Even if reality remains far too vague here – in our opinion, the following statement still best summarizes the nature of a legacy system (if it has to be summarized in one sentence): *"An information system becomes obsolete to the extent that it no longer keeps pace with reality"* [98, p. 24]. Furthermore, we have shown that changes in business and technology are the root of aging and that information systems in highly dynamic environments (which information technology as a constituent technology for IS typically is) are strongly affected by aging risk. Ultimately, it is the vicious circle of legacy systems that we have outlined which, as a combination of many factors, leads to the emergence of such systems.

Moreover, out of the confluence of the known socio-technical theory and the viewpoint approach, we have developed a model to explain such systems. On this basis, we understand legacy systems as aged information systems that are viewed from different viewpoints. Furthermore, we found that a legacy system cannot be considered in isolation to explain the underlying legacy phenomenon. Instead, the associated evolutionary process and the system context with which the legacy system is strongly wired has to be considered. This approach provides a viable theoretical basis for explaining legacy systems. Despite the relevance of time and space as drivers for the aging of legacy systems, the model cannot explain these factors because the underlying theory does not take time and space into account. This issue should be investigated in further research.

Poor alignment and unsustainable changes cause IS to age. The question arises as to whether the emergence of a legacy system can be avoided by (a) constantly implementing all change requirements promptly, (b) through a proper and quality-assured evolution process, and flanking this (c) with regular reengineering measures. It should be investigated whether this systematic 'keeping up-to-date' approach (which thus prevents the emergence of a legacy system in the first place) is cost-efficient and feasible to avoid the legacy dilemma and the associated negative consequences. In this case, the total return on capital must be significantly higher.

The literature research has shown that legacy systems are in many cases viewed as business-critical IS whose failure results in serious disruptions to the company. It can be assumed that these business-critical systems are very large business applications [84] and are disproportionately affected by the legacy phenomenon. This assumption should be substantiated by further research in this area in order to better explain and understand this type of system.

The four viewpoints used (development, operational, organizational, and strategic) should be used with caution: The viewpoint approach used clearly shows that an LIS must be viewed and evaluated from different perspectives. However, it is unclear which perspectives (in the sense of valid stereotypes) are particularly suitable for this topic. It is conceivable that other or further viewpoints (than those mentioned earlier) are required for LIS evaluation. Moreover, we have assumed the viewpoints to be static. However, the relevant viewpoints may change over time or vary from organization to organization. For example, the viewpoints of the software engineers may shift from inside information system boundary to outside the organization if a previously self-developed IT application is replaced by a COTS application. Ultimately, further research should focus on this issue and work out which viewpoints (including their changes over time) are appropriate for LIS evaluation.

In further research, we will combine the concept proposed above to explain legacy systems with methods for their evaluation (i.e. measuring the age of information systems).

We hope to gain insights into how the legacy stock of organizations can be determined (through a taxonomy we are planning for this purpose) and how this can be used to drive further digitalization.

Finally, we have shown that changes, and consequently the risk of aging, are inevitable. *"Only by acknowledging change as a constant in our industry can we successfully negotiate the challenges we face and ensure our continued survival in this computer age"* [6]. In this context, we have shown with our model that this change and the understanding about the business-technological age can be understood differently from various points of view. Ultimately, with the proposed model, we have shown a way to integrate the previously disparate and sometimes contradictory statements about LIS into a consistent model. In this way, this model helps to better understand the nature of LIS and, consequently, to support the communication, design and evolution of such systems by outlining the main reasons for the emergence and decay of such systems.

REFERENCES

[1] K. Bennett, "Legacy systems: Coping with success," *IEEE Softw.*, vol. 12, no. 1, pp. 19–23, Jan. 1995, doi: 10.1109/52.363157.

[2] J. Gauger, M. M. Art, and E. Sondergeld. (2024). *Legacy Systems and Modernization: Core Systems Strategy for Policy Administration Systems*. Accessed: Apr. 9, 2024. [Online]. Available: https://www2.deloitte.com/content/dam/Deloitte/us/Documents/financial-services/us-fsi-legacy-systems-and-modernization.pdf

[3] (2024). *Texas Department of Information Resources, Legacy Systems Study Public Report: Assessment and Recommendations*. Accessed: Apr. 9, 2024. [Online]. Available: https://dir.texas.gov/resource-library-item/legacy-systems-study-public-report

[4] Deloitte GmbH. (2018). *Studie Legacy-Modernisierung*. Accessed: Apr. 9, 2024. [Online]. Available: https://www2.deloitte.com/content/dam/Deloitte/de/Documents/technology/Deloitte_Legacy-Modernisierung_Studie_2018.pdf

[5] D. L. Parnas, "Software aging," in *Proc. 16th Int. Conf. Softw. Eng.*, Sorrento, Italy, pp. 279–287.

[6] M. M. Lehman, "Software's future: Managing evolution," *IEEE Softw.*, vol. 15, no. 1, pp. 40–44, Jan. 1998, doi: 10.1109/MS.1998.646878.

[7] A. Alexandrova, L. Rapanotti, and I. Horrocks, "The legacy problem in government agencies," in *Proc. 16th Annu. Int. Conf. Digit. Government Res.*, New York, NY, USA, May 2015, pp. 150–159. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/2757401.2757406

[8] R. N. Charette, "Dragging government legacy systems out of the shadows," *Computer*, vol. 49, no. 9, pp. 114–119, Sep. 2016, doi: 10.1109/MC.2016.289.

[9] R. Khadka, B. V. Batlajery, A. M. Saeidi, S. Jansen, and J. Hage, "How do professionals perceive legacy systems and software modernization?" in *Proc. 36th Int. Conf. Softw. Eng.*, 2014, pp. 36–47.

[10] A. Alderson and H. Shah, "Technical opinion: Viewpoints on legacy systems," *Commun. ACM*, vol. 42, no. 3, pp. 115–116, Mar. 1999, doi: 10.1145/295685.295722.

[11] S. Comella-Dorda, K. Wallnau, R. C. Seacord, and J. Robert, *A Survey of Legacy System Modernization Approaches*. Fort Belvoir, VA, USA: Carnegie Mellon Univ., Software Engineering Institute, 2000. [Online]. Available: https://insights.sei.cmu.edu/documents/1958/2000_004_001_13673.pdf

[12] A. M'baya, J. Laval, and N. Moalla, "An assessment conceptual framework for the modernization of legacy systems," in *Proc. 11th Int. Conf. Softw., Knowl., Inf. Manage. Appl. (SKIMA)*, Aug. 2017, pp. 1–11.

[13] S. M. Hussain, S. N. Bhatti, and M. F. Ur Rasool, "Legacy system and ways of its evolution," in *Proc. Int. Conf. Commun. Technol. (ComTech)*, Apr. 2017, pp. 56–59.

[14] I. Sommerville, *Software Engineering*, 6th ed. München, Germany: Pearson Studium, 2003.

[15] J. Ransom, I. Somerville, and I. Warren, "A method for assessing legacy systems for evolution," in *Proc. 2nd Euromicro Conf. Softw. Maintenance Reeng.*, Florence, Italy, 1998, pp. 128–134.

[16] G. Visaggio, "Ageing of a data-intensive legacy system: Symptoms and remedies," *J. Softw. Maintenance Evol., Res. Pract.*, vol. 13, no. 5, pp. 281–308, Sep. 2001, doi: 10.1002/smr.234.

[17] Catherine, J. D. Trisaktyo, T. Ranas, M. Rasyiid, and M. R. Shihab, "Embracing agile development principles in an organization using the legacy system: The case of bank XYZ in Indonesia," in *Proc. 6th Int. Conf. Comput. Eng. Design (ICCED)*, 2020, pp. 1–5.

[18] K. Liu and B. Sharp, "A strategic attempt to management of legacy information systems," in *Proc. IEE Colloq. Legacy Inf. Syst.-Barriers Bus. Process Reeng.*, 1994, pp. 4/1–4/6.

[19] S. Demeyer, S. Ducasse, and O. Nierstrasz. (2024). *Object-oriented Reengineering Patterns*. Accessed: Apr. 21, 2024. [Online]. Available: http://scg.unibe.ch/download/oorp/

[20] H. M. Sneed and R. Seidl, *Softwareevolution: Erhaltung Und Fortschreibung Bestehender Softwaresysteme*, 1st ed. Heidelberg, Germany: Dpunkt.Verlag, 2013.

[21] P. O'Byrne and B. Wu, "LACE frameworks and technique-identifying the legacy status of a business information system from the perspectives of its causes and effects," in *Proc. Int. Symp. Princ. Softw. Evol.*, 2000, pp. 170–174.

[22] P. O'Byrne, "An investigation into the causes and effects of legacy status in a system with a view to assessing both systems currently in use and those being considered for introduction," Ph.D. dissertation, School Comput. Sci., Faculty Comput., Digit. Data, Technol. Univ. Dublin, Staffordshire Univ., Dublin, Ireland, 1999. Accessed: Apr. 9, 2024. [Online]. Available: https://arrow.tudublin.ie/scschcomoth/7/

[23] B. Y. Alkazemi, M. K. Nour, and A. Q. Meelud, "Towards a framework to assess legacy systems," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, Oct. 2013, pp. 924–928.

[24] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS Quart.*, vol. 28, no. 1, p. 75, 2004, doi: 10.2307/25148625.

[25] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *J. Manage. Inf. Syst.*, vol. 24, no. 3, pp. 45–77, Dec. 2007, doi: 10.2753/mis0742-1222240302.

[26] M. Ramage, "Global perspectives on legacy systems," in *Systems Engineering for Business Process Change: New Directions: Collected Papers From the EPSRC Research Programme*. London, U.K.: Springer, 2002, pp. 309–316. [Online]. Available: https://link.springer.com/chapter/10.1007/978-1-4471-0135-2_19

[27] L. J. Heinrich, A. Heinzl, and R. Riedl, *Wirtschaftsinformatik: Einführung und Grundlegung*, 4th ed. Berlin, Germany: Springer, 2011.

[28] M. Taylor, E. Moynihan, and T. Wood-Harper, "Knowledge for software maintenance," *J. Inf. Technol.*, vol. 12, no. 2, pp. 155–166, Jun. 1997, doi: 10.1177/026839629701200207.

[29] M. Lind and A. Lauder. (1999). *Legacy Systems: Assets or Liabilities: A Language Action Perspective on Respecting and Reflecting Negotiated Business Relationships in Information Systems*. School Bus. IT. Accessed: Apr. 9, 2024. [Online]. Available: https://urn.kb.se/resolve?urn=urn:nbn:se:hb:diva-4215

[30] R. A. Teubner, *Organisations-und Informationssystemgestaltung: Theoretische Grundlagen und Integrierte Methoden*. Wiesbaden, Germany: Deutscher Universitätsverlag, 1999, doi: 10.1007/978-3-322-99957-3.

[31] N. Wiener and E. H. Serr, "Kybernetik: Regelung und Nachrichtenübertragung in Lebewesen und Maschine," in *Proc. 34th Reinbek bei Hamburg*. Hamburg, Germany: Rowohlt-Taschenbuch-Verlag, 1971, p. 166.

[32] G. Ropohl. (5445). *Allgemeine Technologie: Eine Systemtheorie Der Technik*. Accessed: Apr. 9, 2024. [Online]. Available: https://www.ksp.kit.edu/site/books/m/10.5445/KSP/1000011529/

[33] T. Wilde and T. Hess, "Forschungsmethoden der wirtschaftsinformatik," *Wirtschaftsinformatik*, vol. 49, no. 4, pp. 280–287, Aug. 2007, doi: 10.1007/s11576-007-0064-z.

[34] J. vom Brocke, A. Simons, B. Niehaves, K. Riemer, R. Plattfaut, and A. Cleven. (2009). *Reconstructing the Giant: On the Importance of Rigour in Documenting the Literature Search Process*. [Online]. Available: http://www.alexandria.unisg.ch/Publikationen/67910

[35] Y. Levy and T. J. Ellis, "A systems approach to conduct an effective literature review in support of information systems research," *Informing Sci., Int. J. Emerg. Transdiscipline*, vol. 9, pp. 181–212, Jan. 2006, doi: 10.28945/479.

[36] J. Webster and R. T. Watson, "Analyzing the past to prepare for the future: Writing a literature review," *MIS Quart.*, vol. 26, no. 2, pp. 13–23, 2002.

[37] H. M. Cooper, "Organizing knowledge syntheses: A taxonomy of literature reviews," *Knowl. Soc.*, vol. 1, no. 1, pp. 104–126, Mar. 1988, doi: 10.1007/bf03177550.

[38] H. W. J. Rittel and M. M. Webber, "Dilemmas in a general theory of planning," *Policy Sci.*, vol. 4, no. 2, pp. 155–169, Jun. 1973, doi: 10.1007/bf01405730.

[39] M. M. Lehman, "Programs, life cycles, and laws of software evolution," *Proc. IEEE*, vol. 68, no. 9, pp. 1060–1076, Jan. 1980, doi: 10.1109/PROC.1980.11805.

[40] P. Aiken, A. Muntz, and R. Richards, "A framework for reverse engineering DoD legacy information systems," in *Proc. Work. Conf. Reverse Eng.*, Baltimore, MD, USA, Sep. 1993, pp. 180–191.

[41] R. Nassif and D. Mitchusson, "Issues and approaches for migration/cohabitation between legacy and new systems," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, Jun. 1993, pp. 471–474.

[42] E. A. L. Phillips, "Legacy or liability-the will to change," in *IEE Colloquium on Legacy Information System-Barriers to Business Process Re-Eng. (Digest, No. 1994/246)*. IET, 1994, p. 112.

[43] W. S. Adolph, "Cash cow in the tar pit: Reengineering a legacy system," *IEEE Softw.*, vol. 13, no. 3, pp. 41–47, May 1996, doi: 10.1109/52.493019.

[44] J. Bisbal. *A Survey of Research Into Legacy System Migration*. Accessed: Apr. 9, 2024. [Online]. Available: https://www.scss.tcd.ie/publications/tech-reports/reports.97/TCD-CS-1997-01.pdf

[45] J. Bisbal, "An overview of legacy information system migration," in *Proc. Joint 4th Int. Comput. Sci. Conf. 4th Asia Pacific Softw. Eng. Conf.*, Hong Kong, 1997, pp. 529–530.

[46] J. J. Kaasbøll, "How evolution of information systems may fail: Many improvements adding up to negative effects," *Eur. J. Inf. Syst.*, vol. 6, no. 3, pp. 172–180, Sep. 1997, doi: 10.1057/palgrave.ejis.3000264.

[47] N. H. Weiderman, L. M. Northrop, D. B. Smith, S. R. Tilley, and K. C. Wallnau. (2024). *Implications of Distributed Object Technology for Reengineering*. Accessed: Apr. 9, 2024. [Online]. Available: https://apps.dtic.mil/sti/tr/pdf/ADA326945.pdf

[48] Y.-G. Kim, "Improving legacy systems maintainability," *Inf. Syst. Manage.*, vol. 14, no. 1, pp. 7–11, Jan. 1997, doi: 10.1080/10580539708907023.

[49] C. Slee and M. Slovin, "Legacy asset management," *Inf. Syst. Manage.*, vol. 14, no. 1, pp. 12–21, Jan. 1997, doi: 10.1080/10580539708907024.

[50] M. M. Lehman, J. F. Ramil, P. D. Wernick, D. E. Perry, and W. M. Turski, "Metrics and laws of software evolution-the nineties view," in *Proc. 4th Int. Softw. Metrics Symp.*, Albuquerque, NM, USA, Nov. 1997, pp. 20–32.

[51] S. Bollig and D. Xiao, "Throwing off the shackles of a legacy system," *Computer*, vol. 31, no. 6, pp. 104–106, Jun. 1998, doi: 10.1109/2.683012.

[52] K. H. Bennett, M. Ramage, and M. Munro, "Decision model for legacy systems," *IEE Proc. Softw.*, vol. 146, no. 3, p. 153, Jan. 1999, doi: 10.1049/IP-SEN:19990617.

[53] S. Kelly, N. Gibson, C. P. Holland, and B. Light, "Focus issue on legacy information systems and business process change: A business perspective of legacy information systems," in *Proc. CAIS*, vol. 2, 1999, pp. 1–27. [Online]. Available: https://aisel.aisnet.org/cais/vol2/iss1/7/, doi: 10.17705/1CAIS.00207.

[54] J. Bisbal, D. Lawless, B. Wu, and J. Grimson, "Legacy information systems: Issues and directions," *IEEE Softw.*, vol. 16, no. 5, pp. 103–111, Sep. 1999, doi: 10.1109/52.795108.

[55] E. Mitleton-Kelly and M.-C. Papaefthimiou, "Co-evolution and an enabling infrastructure: A solution to legacy?" in *Systems Engineering for Business Process Change: Collected Papers from the EPSRC Research Programme*, P. Henderson, Ed. London, U.K.: Springer, 2000, pp. 164–181.

[56] M. Ramage and M. Munro, "It's not just about old software: A wider view of legacy systems," in *Systems Engineering for Business Process Change: Collected Papers from the EPSRC Research Programme*, P. Henderson, Ed. London, U.K.: Springer, 2000, pp. 279–290.

[57] L. Erlikh, "Leveraging legacy system dollars for e-business," *IT Prof.*, vol. 2, no. 3, pp. 17–23, 2000, doi: 10.1109/6294.846201.

[58] A. Lauder and S. Kent, "Legacy system anti-patterns and a pattern-oriented migration response," in *Systems Engineering for Business Process Change: Collected Papers from the EPSRC Research Programme*, P. Henderson, Ed. London, U.K.: Springer, 2000, pp. 239–250.

[59] (2000). *Software Maintenance and Evolution: A Roadmap*. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/336512.336534

[60] A. De Lucia, A. R. Fasolino, and E. Pompelle, "A decisional framework for legacy system management," in *Proc. IEEE Int. Conf. Softw. Maintenance (ICSM)*, Florence, Italy, Sep. 2001, pp. 642–651.

[61] A. Lauder and S. Kent, "More legacy system patterns," in *Systems Engineering for Business Process Change: New Directions, Systems Engineering for Business Process Change: New Directions: Collected Papers From the EPSRC Research Programme*, P. Henderson, Ed. London, U.K.: Springer, 2002, pp. 225–240.

[62] B. Light, "An alternative theory of legacy information systems," in *Proc. ECIS*, 2003, pp. 1093–1111. Accessed: Apr. 9, 2024. [Online]. Available: https://aisel.aisnet.org/ecis2003/93/

[63] M. P. Gupta and D. Bhatia, "Reworking with a legacy financial accounting system: Lessons from a pharma company," *Vikalpa: J. Decis. Makers*, vol. 30, no. 3, pp. 79–92, Jul. 2005, doi: 10.1177/0256090920050307.

[64] A. Dedeke, "Improving legacy-system sustainability: A systematic approach," *IT Prof.*, vol. 14, no. 1, pp. 38–43, Jan. 2012, doi: 10.1109/MITP.2012.10.

[65] H. K. A. Bakar and R. Razali, "A preliminary review of legacy information systems evaluation models," in *Proc. Int. Conf. Res. Innov. Inf. Syst. (ICRIIS)*, Kuala Lumpur, Malaysia, Nov. 2013, pp. 314–318.

[66] R. Khadka, A. Saeidi, S. Jansen, J. Hage, and G. P. Haas, "Migrating a large scale legacy application to SOA: Challenges and lessons learned," in *Proc. 20th Work. Conf. Reverse Eng. (WCRE)*, Koblenz, Germany, Oct. 2013, pp. 425–432.

[67] B. Y. Alkazemi, "A framework to assess legacy software systems," *J. Softw.*, vol. 9, no. 1, pp. 111–115, 2014, doi: 10.4304/jsw.9.1.111-115.

[68] V. Rajlich, "Software evolution and maintenance," in *Future of Software Engineering Proceedings*. New York, NY, USA: Association for Computing Machinery, 2014. [Online]. Available: https://dl.acm.org/doi/10.1145/2593882.2593893

[69] R. Khadka, P. Shrestha, B. Klein, A. Saeidi, J. Hage, S. Jansen, E. van Dis, and M. Bruntink, "Does software modernization deliver what it aimed for? A post modernization analysis of five software modernization case studies," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Bremen, Germany, Sep. 2015, pp. 477–486.

[70] R. Pérez-Castillo, B. Mas, and M. Pizka, "Understanding legacy architecture patterns," in *Proc. Int. Conf. Eval. Novel Approaches Softw. Eng. (ENASE)*, Apr. 2015, pp. 282–288.

[71] S. Matthiesen and P. Bjørn, "Why replacing legacy systems is so hard in global software development," in *Proc. 18th ACM Conf. Comput. Supported Cooperat. Work Social Comput.*, Vancouver, BC, Canada, Feb. 2015, pp. 876–890.

[72] T. C. Fanelli, S. C. Simons, and S. Banerjee, "A systematic framework for modernizing legacy application systems," in *Proc. IEEE 23rd Int. Conf. Softw. Anal., Evol., Reengineering (SANER)*, vol. 1, Osaka, Japan, Mar. 2016, pp. 678–682.

[73] M. Srinivas, G. Ramakrishna, K. R. Rao, and E. S. Babu, "Analysis of legacy system in software application development: A comparative survey," *Int. J. Electr. Comput. Eng. (IJECE)*, vol. 6, no. 1, p. 292, Feb. 2016, doi: 10.11591/ijece.v6i1.8367.

[74] (2016). *Co-Existence of the'Technical Debt'and'Software Legacy'Concepts*. [Online]. Available: https://www.utupub.fi/bitstream/handle/10024/169404/apsec_tda_2016_paper_6_camera_ready.pdf?sequence=1

[75] S. Johann, "Dave Thomas on innovating legacy systems," *IEEE Softw.*, vol. 33, no. 2, pp. 105–108, Mar. 2016, doi: 10.1109/MS.2016.38.

[76] J. Crotty and I. Horrocks, "Managing legacy system costs: A case study of a meta-assessment model to identify solutions in a large financial services company," *Appl. Comput. Informat.*, vol. 13, no. 2, pp. 175–183, Jul. 2017, doi: 10.1016/j.aci.2016.12.001.

[77] H. Huijgens, A. van Deursen, and R. van Solingen, "Success factors in managing legacy system evolution," in *Proc. Int. Conf. Softw. Syst. Process*, May 2016, pp. 96–105.

[78] B. D. Monaghan and J. M. Bass. (2020). *Redefining Legacy: A Technical Debt Perspective*. Accessed: Apr. 9, 2024. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-64148-1_16

[79] M. L. Brodie and M. Stonebraker, *Migrating Legacy Systems: Gateways, Interfaces & the Incremental Approach*. San Francisco, CA, USA: Kaufmann Publ, 1995.

[80] A. Hunt and D. Thomas, "Software archaeology," *IEEE Softw.*, vol. 19, no. 2, pp. 20–22, Jan. 2002, doi: 10.1109/52.991327.

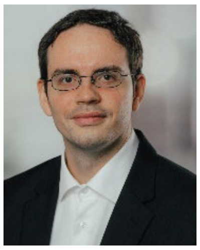
[81] D. Staegemann, M. Volk, C. Daase, and K. Turowski, "Discussing relations between dynamic business environments and big data analytics," *Complex Syst. Inform. Model. Quart.*, vol. 23, pp. 58–82, Jul. 2020, doi: 10.7250/csimq.2020-23.05.

[82] P. Vemuri, "IEEE TENCON–2008 modernizing a legacy system to SOA–Feature analysis approach," in *Proc. TENCON IEEE Region 10 Conf.*, Hyderabad, India, Nov. 2008, pp. 1–6.

[83] Z. Masood, R. Hoda, K. Blincoe, and D. Damian, "Like, dislike, or just do it? How developers approach software development tasks," *Inf. Softw. Technol.*, vol. 150, Oct. 2022, Art. no. 106963, doi: 10.1016/j.infsof.2022.106963.
[84] D. Dreschel, "Towards a classification framework for very large business applications," in *GI-Edition Lecture Notes in Informatics Proceedings*, vol. 208. Bonn, Germany: TU Braunschweig, 2012, pp. 273–283.
[85] B. Grabski, S. Günther, S. Herden, L. Krüger, C. Rautenstrauch, and A. Zwanziger, "Very large business Applications," *Informatik-Spektrum*, vol. 30, no. 4, pp. 259–263, Jun. 2007, doi: 10.1007/s00287-007-0171-7.
[86] B. Paradauskas and A. Laurikaitis, "Business knowledge extraction from legacy information systems," *Inf. Technol. Control*, vol. 35, no. 3, pp. 214–221, Sep. 2006, doi: 10.5755/j01.itc.35.3.11772.
[87] H.-J. Kung, "Quantitative method to determine software maintenance life cycle," in *Proc. 20th IEEE Int. Conf. Softw. Maintenance*, Sep. 2004, pp. 232–241.
[88] E. J. Chikofsky and J. H. Cross, "Reverse engineering and design recovery: A taxonomy," *IEEE Softw.*, vol. 7, no. 1, pp. 13–17, Jan. 1990, doi: 10.1109/52.43044.
[89] G. Dhillon and M. Caldeira, "A bumpy road to success (or not): The case of project genesis at Nevada DMV," *Int. J. Inf. Manage.*, vol. 28, no. 3, pp. 222–228, Jun. 2008, doi: 10.1016/j.ijinfomgt.2008.02.001.
[90] H. M. Sneed, E. Wolf, and H. Heilmann, *Softwaremigration in der Praxis: Übertragung Alter Softwaresysteme in Eine Moderne Umgebung*, 1st ed. Heidelberg, Germany: Dpunkt.verlag, 2010. [Online]. Available: https://ebookcentral.proquest.com/lib/subhh/detail.action?docID=4347080
[91] *Software Engineering Software Product Quality Requirements and Evaluation (SQuaRE) Data Quality Model*, document ISO/IEC 25012:2008, Dec. 2008. [Online]. Available: https://www.iso.org/standard/35736.html
[92] *Systems and Software Engineering Systems and Software Quality Requirements and Evaluation (SQuaRE) System and Software Quality Models*, document 25010:2011, ISO/IEC 25010:2011, Mar. 2011. [Online]. Available: https://www.iso.org/standard/35733.html
[93] G. Sagers, K. Ball, B. Hosack, D. Twitchell, and D. Wallace, "The mainframe is dead. Long live the mainframe!" *AIS Trans. Enterprise Syst.*, vol. 2, no. 1, pp. 4–10, 2013. [Online]. Available: https://www.aes-journal.com/index.php/ais-tes/article/view/6
[94] S. Vesić and D. Laković, "A framework for evaluating legacy systems—A case study," *Kultura Polisa*, vol. 20, no. 1, pp. 32–50, Apr. 2023, doi: 10.51738/kpolisa2023.20.1r.32vl.
[95] G. Midgley, "The sacred and profane in critical systems thinking," *Syst. Pract.*, vol. 5, no. 1, pp. 5–16, Feb. 1992, doi: 10.1007/bf01060044.
[96] C. Rautenstrauch and T. Schulze, *Informatik Für Wirtschaftswissenschaftler und Wirtschaftsinformatiker*. Berlin, Germany: Springer, 2003.
[97] H. R. Hansen and G. Neumann, *Wirtschaftsinformatik 1: Grundlagen und Anwendungen*, 10th ed. Stuttgart, Germany: Lucius & Lucius, 2009.
[98] H. M. Sneed, *Softwarewartung Und-Wiederverwendung*. Köln, Germany: Müller, 1991.
[99] J. Schelp, Ed., *Integration, Informationslogistik und Architektur: DW 2006; 21. 22.09.2006 in Friedrichshafen*. Bonn, Germany: Gesellschaft Für Informatik, 2006. [Online]. Available: http://subs.emis.de/LNI/Proceedings/Proceedings90.html

SEBASTIAN ROSENKRANZ was born in Halberstadt, Saxony-Anhalt, Germany, in 1979. He received the Diploma degree in business informatics from Otto von Guericke Universität Magdeburg (OVGU), Saxony-Anhalt, in 2007, where he is currently pursuing the Ph.D. degree.

In 2007, he experienced his first practical and scientific contact with legacy systems as a Student in the context of his Diploma thesis at Volkswagen AG. Since then, he has worked on legacy systems in various roles in this company as a Business Analyst, a Tester, and a Requirements Engineer. In 2012, he was a Product Owner, responsible for the information system that provides software updates to the global fleet, and working on the replacement of a legacy product data management system. In addition to his practical work as a Product Owner, he has also been working scientifically on legacy systems, focusing on the legacy phenomenon in companies.



DANIEL STAEGEMANN was born in Berlin, in 1989. He received the master's degree in computer science from the Technical University Berlin (TUB), in 2017. He is currently pursuing the Ph.D. degree with Otto von Guericke University Magdeburg (OVGU).

Since 2018, he has been a Scientific Researcher. During this time, he has published over 70 papers in prestigious outlets, such as the Americas Conference on Information Systems (AMCIS), the Pacific Asia Conference on Information Systems (PACIS), the International Conference on Design Science Research in Information Systems and Technology (DESRIST), the International Conference on Business Information Systems (BIS), the Hawaii International Conference on System Sciences (HICSS), the *Journal of Big Data*, and IEEE Access. Besides being an author and a speaker at conferences, as well as a mini-track and workshop chair, he also regularly acts as a reviewer. His research interests include big data and the corresponding quality assurance, but it also encompasses all other related topics.

MATTHIAS VOLK was born in 1991. He received the master's degree in business informatics from Otto von Guericke University Magdeburg (OVGU), in 2016, and the Ph.D. degree in engineering, in 2022, with a focus on decision support for the technology selection in big data projects.

After receiving the master's degree, he continued his work as a Scientific Researcher at OVGU, where he became a Research Coordinator and the Operational Head of the Very Large Business Application Laboratory (VLBA Lab), in 2020. After the Ph.D. degree, he entered the industry while continuing his scientific work as an Associate Researcher. During his time at the OVGU, he published over 60 papers in prestigious outlets, such as the Americas Conference on Information Systems (AMCIS), Pacific Asia Conference on Information Systems (PACIS), International Conference on Design Science Research in Information Systems, and Technology (DESRIST), International Conference on Business Information Systems (BIS), Hawaii International Conference on System Sciences (HICSS), and IEEE Access. Besides being an author and a speaker at conferences and a mini-track and workshop chair, he regularly acts as a reviewer. His research interests include big data, data-intensive systems, and decision support systems, but it also encompasses all other related topics.

KLAUS TUROWSKI was born in 1966. He received the degree in business and engineering from the University of Karlsruhe, the Ph.D. degree from the Institute for Business Informatics, University of Münster, and the Habilitated degree in business informatics from the Faculty of Computer Science, Otto von Guericke University Magdeburg.

In 2000, he deputized the Chair of Business Informatics, University of the Federal Armed Forces München. In 2001, he headed the Chair of Business Informatics and Systems Engineering, University of Augsburg. Since 2011, he has been heading the Chair of Business Informatics (AG WI), Very Large Business Applications Laboratory (VLBA Lab), Otto von Guericke University Magdeburg, and the world's largest SAP University Competence Center (SAP UCC Magdeburg). In addition, he was a Guest Lecturer at several universities around the world and a Lecturer with the Universities of Darmstadt and Konstanz. He was a (co-) organizer of a multiplicity of national and international scientific congresses and more than 30 workshops and acted as a member of more than 130 program committees and expert groups. In the context of his university activities and as an independent consultant, he gained practical experience in the industry.