Towards a Topology for Legacy System Migration

Stefan Strobl
stefan.strobl@inso.tuwien.ac.at
Vienna University of Technology
Vienna, Austria

Mario Bernhart
mario.bernhart@inso.tuwien.ac.at
Vienna University of Technology
Vienna, Austria

Thomas Grechenig
thomas.grechenig@inso.tuwien.ac.at
Vienna University of Technology
Vienna, Austria

ABSTRACT

Dealing with legacy systems is a decade old industry challenge. The pressure to efficiently modernise legacy both to meet new business requirements and to mitigate inherent risks is ever growing. Our experience shows a lack of collaboration between researchers and practitioners inhibiting innovation in the field. To facilitate communication between academia and industry and as a byproduct to obtain an up to date picture of the state of affairs we are creating a legacy system migration topology based on generalisations from a multi-case study as well as extensive literature research. We expect the topology to be useful in connecting industry needs and challenges with current and potential future research and to improve bidirectional accessibility.

CCS CONCEPTS

• **Social and professional topics** → **Reengineering**; *Software maintenance*; *Software reverse engineering*; • **Software and its engineering** → **Software evolution**.

KEYWORDS

legacy system migration, academia industry communication, academia industry collaboration, reengineering

ACM Reference Format:
Stefan Strobl, Mario Bernhart, and Thomas Grechenig. 2020. Towards a Topology for Legacy System Migration. In *IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW'20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3387940.3391476

1 INTRODUCTION

Large companies and organisations typically host a wide variety of systems that have grown and evolved over the decades as digital technology was gradually adopted. With technological progress the landscape is continuously changing and diversifying as new features and capabilities are developed with new technologies and some (smaller) parts of existing systems are replaced. However, at the core of the enterprise, petrified [26] systems remain that are too complex to easily replace and at the same time much too valuable to the business to be removed. These so called legacy systems (LS) are as diverse as the histories of the associated IT departments. Some are tightly coupled, highly monolithic monstrosities. Others are loosely coupled quagmires with hardly discernible boundaries. They all have in common that the responsible managers and technologists have long identified them as a highly problematic, but at the same time lack either the know how or resources to adequately address their shortcomings. Researchers on the other hand struggle to elaborate industrially widely applicable methodologies and solutions in an academic setting [23]. Similar to other areas of computer science (like datasets for training AI models), large scale legacy systems and their complex sociotechnical surroundings cannot easily be simulated for experimentation.

A fundamental challenge that enterprise IT is facing at the moment, is the speed of technological evolution [21]. This becomes especially apparent in migration scenarios as new technology is directly compared and also benchmarked against legacy technologies. Existing legacy systems have been in place for decades, undergoing relatively little technological evolution. Of course there were some fundamental changes, like replacing data storage mechanisms by relational databases or text based terminal interfaces by graphical user interfaces, but at the core business functionality was developed and delivered in similar ways over many years.

This is in stark contrast to the technological landscape today. The life expectancy of a software system has shortened significantly, in some areas even radically. Already almost 30 years ago the average of a lifespan was found to be around 6-8 years, slightly increasing with the size of software [44]. No indication could be found that the lifespan has increased since. The size of enterprise systems can likely explain parts of a perceived longer lifespan for these systems when compared to consumer-facing web or mobile applications. However, rapid technological turnaround does affect enterprise systems in similar ways as consumer applications, especially when it comes to frontend technology. This opens another way of reasoning in connecting the lifetime of a system with the evolution of today's presentation technologies based on the web and mobile platforms. The browser and the phone as replacements for classic operating systems and therefore user interface platforms are inherently connected to the pace of evolution of consumer technology. This creates a disparity as these platforms lack the requirements for stability related to significantly longer lifetimes dictated by enterprise RoI calculations.

We therefore propose a topology (see section 3 for our definition) of legacy system migration (LSM) based on experiences and generalisations extracted from our own case study research as well as published third party cases. The main purpose of this topology is to facilitate communication between researchers and practitioners. For the latter a topology provides an easy way of navigating the complexity of LSM, to classify challenges encountered in practice and to articulate experiences from an industrial setting. For the former the topology helps to understand the industries' needs and

ICSEW'20, May 23–29, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7963-2/20/05…$15.00
https://doi.org/10.1145/3387940.3391476

find open research questions, to make existing research more visible to a broader audience and most importantly to connect academic approaches and industrial cases.

2 RELATED WORK

Van Deursen et al. observed in 1999 that we are sitting on a "software volcano" [47]. This was the time when the Y2K problems as well as the Euro conversion brought to light the sheer amount of skeletons we had been collectively hiding in our closets. Twenty years later little overall progress is visible, memories of the year 2000 and pre Euro currencies seem to be fading quickly and along with them, the interest in the research topic of software renovation in general. The gap between academics and industry seems to be widening: The academic circles are focusing mainly on technical issues and, as a side note, almost completely neglecting the topic in the curricula [33]. The industry is primarily experiencing organisational and business perspective problems [8, 23] as well as an acute lack of skilled professionals [39, 45]. This is hardly a novel realisation. Parnas already wrote in 1994: "We end up writing papers that are read by our fellow researchers but not many others." [31], the relevance of the issue has grown with the scale of the legacy.

Extensive work has been done in mapping current publications on migration with a strong focus on target architectures. Lewis et al. addressed the challenges in migrating to Service Oriented Architecture (SOA) and Cloud computing environments [28]. Razavian et al. [34] created a systematic literature review, illustrating the parallels between the Gartner Hype Cycle for SOA and the publication of papers on the topic of "SOA migration". More recently the focus has shifted to migration towards Microservices [16] or "the Cloud" [6, 22] or a combination of both [5]. This culminates in the topic of migrating from one hype to the next, as a systematic mapping study by Botto-Tobar et al. [13] shows.

Khadka et al. have done a post modernization analysis of five cases after a LSM was performed. The expected benefits were largely met, but there were also some negative side effects related to user acceptance and performance degradation. In addition, some "unexpected" benefits (positive side effects) in the space of organisational transparency and flexibility were identified [24].

Akhtyamov et al. address system migration from a systems engineering perspective. They describe the lack of a joint terminology on system modernization, provide a classification and propose a framework for selecting a modernization strategy [3].

Fanelli et al. propose a high level framework for legacy system migration, focusing on the application system and leaving the information system and hardware changes for subsequent steps. Their goal is an incremental migration with a strong focus on utilizing subject matter expertise [19].

3 APPROACH

The term "topology" in a non mathematical context is defined by the Oxford dictionary as "The way in which constituent parts are interrelated or arranged." We chose this term as other, more common terms would have narrowed down the scope too much. A taxonomy on the one hand would have reduced our scope to a classification of LSM terms, methods or challenges. A research agenda on the other hand would have the primary focus on open (research) challenges in the field. Our topology combines both and adds significantly more in terms of relationships and relative positioning of the identified abstractions.

3.1 Case Study

To assess the industry state of performing legacy system migration strategies we conducted an exploratory case study involving multiple cases following the guidelines of Verner et al. [48]. This approach is suitable as a "case study is preferred in examining contemporary events, but when the relevant behaviours cannot be manipulated" [52].

All cases have in common that our industry partner has a long standing business relationship with the organisations behind them. The partner has more than two decades of experience in supporting clients to deal with legacy IS challenges. The initial contact to these organisations as well as later collaboration to extract the necessary data were established through this partner.

A total of four cases as listed below (abbreviated form used in narrow, column width tables) from different domains were selected. The chosen domains are a result of the availability and accessibility of cases. The authors paid special attention to ensure a maximum of variation, both in terms of project and information system characteristics and business domain. Multiple cases from the same organisation or from very similar business backgrounds were excluded.

- **Insurance** (Ins.): A renovation and consolidation project in a heterogeneous IT landscape with the goal of providing a central platform for handling master data and automate business processes. The type of migration was a partial replacement combined with wrapping of multiple interacting LS.
- **Ministry** (Min.): A replatforming effort with the goal of freeing up resources to subsequently incrementally replace a very large government IT system.
- **Airport** (Apt.): An incremental, full replacement of a highly critical core information system (IS) that digitally mirrors the state of an airport.
- **University** (Uni.): A complete replacement project for the core information system that consolidates a highly heterogeneous landscape into a coherent enterprise management system.

Aspects from all four cases were published in scientific literature. For the Insurance case "A Tiered Approach Towards an Incremental BPEL to BPMN 2.0 Migration" [43] goes into the specifics of migrating process instances from a legacy process engine to a modern, highly integrated counterpart. "Automated Code Transformations - Dealing with the Aftermath" discusses the long term effects of automated source code transformation [42]. In the Ministry case the paper "Challenges in re-platforming mixed language PL/I and COBOL IS to an open systems platform" [49] discusses some of the main technical obstacles of performing the re-platforming. In the context of the Airport case the focus of scientific publication was on the incremental migration strategy [11]. For the University case a specific focus was set on the reverse engineering of the legacy

system data sources in "Digging deep: Software reengineering supported by database reverse engineering of a system with 30+ years of legacy" [41].

The suitability of the cases was evaluated based on the legacy characteristics the affected information system exhibits. The following characteristics were identified during the selection process. A summary of the initial legacy system assessment for all four cases is shown in Table 1.

- **Need** for renovation is affected by multiple characteristics: the possibility to decompose the legacy system (LS), the composition of the LS, the main technology used in the LS and the database technology. Table 2 gives a detailed breakdown of additional classification factors (as outlined in [40]).
- **Age** of the legacy system is evaluated to distinguish from new, but badly constructed software.
- **Size** of the legacy system is relevant to eliminate trivial and therefore non-representative cases.
- The **Relevance** of the legacy system to the organisation can be judged by characteristics as the number of users, its key purpose, the organisational significance and the maximum downtime until standstill.
- No possible (easy) **substitution** with a COTS product is also evaluated to avoid cases that try to reinvent the wheel.

The need for renovation was subsequently further evaluated utilizing the factors shown in Table 2.

The analysis of the cases was done by studying primary documentation and by direct first hand observation. The observation period in all cases was at least a year, in some instances significantly longer. The description of each case consists of an analysis of the initial state and context [32] from both a technical and organisational point of view, a description of the solution from a project and architecture perspective as well as a discussion of selected experiences and key decisions or turning points that need to be highlighted.

3.2 Literature Search

The search for literature to support the findings in the case study was done using forward-snowballing [51]. The starting set for the literature search was selected based on the authors' experience with the goal of starting from established base literature. It included the following publications: [8, 14, 17, 35, 37, 46]. The last round of the search was conducted in early 2018. Some additional and more recent publications were introduced during the subsequent review process.

3.3 Analysis

Using thematic analysis [18] we extracted generalisations from all four cases (Section 4). Subsequently we validated the generalisations against current state of the art literature. The initial coding utilising an inductive approach was done by one researcher, but subsequently reviewed by three senior colleagues.

The initial results were a total of 12 generalisations mapped to one of the three levels (themes [18]): technical abstraction (Section 4.1), architectural abstraction (Section 4.2) and organisational abstraction (Section 4.1). The business strategy level of abstractions is out of scope for the work presented in this paper. It is only shown to provide a complete picture, to indicate a degree of overlap and to acknowledge that any effort related to a legacy system must have some outside (business) justification.

In a second step of deeper interpretation we grouped these individual items into closely related clusters. This reveals a set of five topics (higher order themes [18]) which are presented in Section 5 and provides the basis for our proposed topology (thematic map [18]). The result is graphically presented in Figure 1.

3.4 Validation

For early validation the proposed topology is then used to classify against three recently papers published at the 2018 International Conference of Software Maintenance and Evolution (ICSME). We selected all papers from all papers of this conference (research and industrial track) relating to the topic of legacy system migration. The goal of this evaluation is to use recent, independent observations to either confirm or possibly enhance our generalisations on the one hand or identify gaps in the topology on the other hand. The validation is presented in Section 6.

4 GENERALISATIONS

A full discussion of all available generalisations would significantly exceed the available space for this paper. However, in this section we provide a brief summary of each generalisation in order to give the reader an idea on how these generalisations are presented with the focus on the main argument or observation. The full description of each generalisation is structured similar to a research paper. It starts with a general introduction and related work, followed by a description of the aspect itself. It is then substantiated by arguments sourced from the previously presented cases as well as referenced literature and rounded off with a summary and outlook. The traceability from case to generalisation is visualised in Table 3. This table can be read as follows. All four cases contributed findings that lead to the elicitation of the generalisation "Cleanup" (4.1.1). Only the cases "Insurance" and "University" on the other hand support the generalisation "Planning" (4.3.4). The traceability from publications to generalisation, including the publications based on the respective cases themselves, is stated as part for the descriptive text below.

4.1 Technical

4.1.1 Cleanup in the legacy system. One of the primary generalisations was to actively commit to performing as many cleanup tasks as possible in the legacy system. This is also described as "Improve Maintainability" by Fanelli et al. in the "Pre-Processing Phase" of their structured modernisation effort [19]. The main purpose is to reduce complexity and potential sources of errors prior to migration, to avoid forcing workarounds in the target system and to enhance data quality. It might consist of rewriting assembler programs or modules where the source is no longer available. We also observed functional changes. This does not only reduce effort and risk during the actual migration, but also has the benefit of reducing the required maintenance in the target system. This generalisation is supported by all cases and these publications: [11, 19, 36, 41, 43].

Table 1: Overview of the main (legacy) characteristics of the four selected units of investigation

		Insurance	Ministry	Airport	University
Need	**LS decomposable**	semi	semi	non	yes
	LS composition	heterogeneous	homogeneous	homogeneous	heterogeneous
	LS main technology	PL/I, COBOL, proprietary, scripting	PL/I, COBOL, assembly	COBOL	COBOL, PL/I, Smalltalk, scripting
	LS database	relational	relational	file-based	relational
Age	**Age**	> 20 years	> 40 years	> 30 years	> 40 years
Size	**Size**	approx. 2.5 MLoC	5 MLoC	250 KLoC	approx. 2 MLoC
Relevance	**# of Users**	6.000	12.000	1.000	6.000 employees, 40.000 students
	Key purpose	customer acquisition and management	taxation and benefits	central operational database	campus management system
	Significance	complete master data, all claim and file handling	complete inner workings of ministry	manages all information distribution	complete IS
	Downtime until stand still	12 hours	6 hours	4 hours	12 hours
COTS	**COTS available**	no	no	yes, evaluated & deemed infeasible	yes, evaluated & deemed infeasible

Table 2: Factors (see also [40]) used to identify the need for renovation of a legacy system and their applicability to our cases.

	Ins.	Min.	Apt.	Uni.
End of Life	partly	no	yes	yes
Knowledge	yes	yes	no	yes
Skills	yes	yes	yes	partly
Extensibility	yes	yes	yes	yes
Time to Repair	partly	partly	no	yes
Scalability	yes	yes	yes	yes
Limitations	yes	yes	yes	yes
Costs	yes	yes	yes	yes

4.1.2 Elicit unused assets. A specialised form of cleanup in the legacy system is the elicitation of unused assets. There is some fundamental overlap with the previous generalisation in terms of eliminating small, unused code snippets and other assets. However, detecting larger chunks of unused artifacts or features becomes increasingly hard and therefore requires specialised techniques like

Table 3: Tracing generalisations back to the originating case(s).

		Ins.	Min.	Apt.	Uni.
Technical	**Cleanup (4.1.1)**	✓	✓	✓	✓
	Unused Assets (4.1.2)	✓	✓	✓	✓
	Code Transform (4.1.3)	✓	✓	✓	
	Leading System (4.1.4)	✓	✓	✓	
Architect.	**IT Strategy & EA (4.2.1)**	✓	✓		✓
	Myths (4.2.2)	✓	✓		
	Prototype (4.2.3)	✓		✓	
	Repeat. Approach (4.2.4)	✓	✓	✓	✓
Organisat.	**Strong Architect (4.3.1)**	✓		✓	✓
	Tailored Process (4.3.2)	✓			✓
	Legacy Technol. (4.3.3)	✓			✓
	Planning (4.3.4)	✓			✓

accessibility analysis or data flow analysis [7] are needed. The main goal is to reduce the overall problem space of a subsequent legacy system migration effort. This generalisation is supported by all four cases and these publications: [11, 15, 41, 43]

4.1.3 Automated transformation. Automated code transformation is frequently sold as the easy way out of a legacy situation [45]. However, as Seacord states in [35]: "Automatic translation [...] cannot significantly change the structure of the code. For example, automatically translating COBOL to Java often results in code that looks like COBOL written in the Java programming language". The main point of this observation is to highlight the difficult trade offs that have to be made when choosing this approach and under which (special) circumstances automated source code transformation is justifiable. We furthermore illustrate the observed long term effects of "successful" automated transformations. These include the continued need for developers with skills in the legacy source technologies and a sustained difficulty in maintaining and extending the migrated systems [42]. This generalisation is supported by three out of four cases and these publications: [35, 38, 39, 45]

4.1.4 Leading system. Besides business functionality the data contained in and managed by a legacy system is a primary value asset [15]. When applying incremental legacy system migration, data ownership (gradually) shifts and this process needs to be explicitly managed. To avoid consistency problems the transfer of ownership is a part of each (incremental) migration step. This generalisation is supported by three out of four cases and these publications: [4, 11, 15]

4.2 Architectural

4.2.1 IT Strategy & EA. A key observation in three out of the four cases was a lack of a well defined IT strategy and Enterprise Architecture (EA) vision. This situation creates an environment where operational and technological decisions have to be made without an overarching strategic and technological guidance, resulting in high additional risks due to misguided or wrong decisions in the day to day project environment. The issue is further complicated by the frequent application of agile approaches combined with the misconception that these approaches do not require any upfront design [50]. This generalisation is supported by three out of four cases and these publications: [2, 4, 35, 46]

4.2.2 SOA & Microservice Myths. On the architecture level, a main generalisation was the phenomenon of migration research and target architectures closely following industry trends and "silver bullet thinking". In the last decade, this was mainly present through Service Oriented Architecture (SOA) and Microservices. While it is of course valid to design target architectures according to current industry standards and best practices, too little critical thinking goes into the suitability of these targets for the problem space covered by the legacy system, as well as associated risks (like architectural mismatch [12]). This generalisation is supported by two out of four cases and the publications: [16, 34].

4.2.3 Build one to throw away. Validation of chosen approaches is relevant in all software engineering settings. However, LS introduce additional, inherent risks, connected for example with the lack of knowledge or the degraded architectural design. It is therefore imperative to practically validate key assumptions and design decisions on an architectural as well as technical level. Apart from verifying concepts this also allows the earlier discovery of unexpected problems and their mitigation. This generalisation is supported by two out of four cases and the publications: [4, 11, 43].

4.2.4 Repeatable Approach to Initial Understanding. Few LSM strategies start early enough to discuss how to approach an (unknown) legacy system. However, even before diving into knowledge extraction and reverse engineering, it is necessary to learn about the nature and history of the LS in question. We identify five aspects (stakeholders, system history & milestones, architecture, processes and supporting infrastructure) that should be summarised in a short biographic summary of the LS. This enables multiple benefits from identifying key experts to avoiding common misconceptions and misunderstandings. It furthermore relays the information that the LS is still vital to the organisation and that adaptations in the LS are still a possibility. This generalisation is supported by all four cases and by the publication [19, 35].

4.3 Organisational

4.3.1 Strong Architect. This generalisation is closely related to the observations described in Section 4.2.1. However, even if IT strategy and enterprise architecture is in place, a strong and impartial (from a project perspective) decision making body (either single role or group) is required. If this body is missing, decision making descends to a project local level losing the ability to abstract from project specific priorities and therefore introducing bias and a high potential for conflicting or oscillating decisions. This generalisation is supported by three out of four cases.

4.3.2 Tailored Development Process. This observation is a combination of two interacting problems. Legacy system migration efforts have unique requirements when it comes to software development methodology. Attempting technological and organisational transformation in the same step puts massive stress on all stakeholders and a LSM project at risk. Naturally, LS development or maintenance is organised around "legacy" best practices, while the development of a target system with newer technologies tends to introduce new concepts. However, these transitions should be sequentially executed as much as possible. This generalisation is supported by three out of four cases and the publication [1].

4.3.3 Learn Legacy Technologies. As already mentioned in the introduction it can be observed that industry is facing a significant skill shortage and gap when it comes to legacy technologies. Billions of lines of legacy code persist in production while the workforce developing and maintaining them is gradually retiring. One proposal to counter this effect is to reintroduce COBOL in computer science curricula [29]. In addition a second important aspect is a strategy on how to deal with legacy subject matter experts [19]. Retraining them in "modern" technologies seems to be a waste of know how already in very short supply. This generalisation is supported by all four cases and the publication [19, 29].

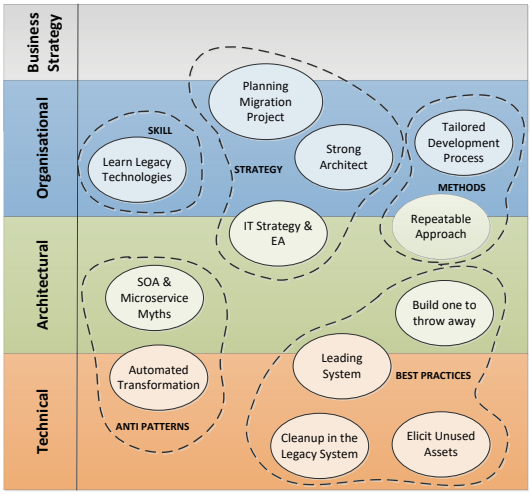

Figure 1: Legacy System Migration Topology: Grouping the generalisations into related topics to highlight relationships and create a thematic map [18].

4.3.4 Planning Migration Project. A key finding on an organisational level was how legacy system migration projects are approached. Only a fraction of them are planned and staffed as migration projects. Typically this happens in scenarios where a strict one-on-one replacement is required [11, 30]. However, in the context of large scale enterprise modernisation projects, migration frequently happens as an afterthought. This negatively affects both the approaches taken in these scenarios as well as resulting solutions. This generalisation is supported by two out of four cases and the following publications: [35, 46].

5 A LEGACY SYSTEM MIGRATION TOPOLOGY

Based on the generalisations elaborated out of the four cases as well as current literature, a topology for legacy system migration was created by grouping them into a total of five topic areas. The result is depicted in Figure 1. In this section we will briefly describe the reasoning behind each topic area and highlight matching potential future research.

5.1 Best Practices

Best practices are relatively easy to apply patterns designed to facilitate the actual implementation of the migration project. These recipes should be part of the repertoire of every experienced legacy system migration expert. Key aspects include actively engaging with the legacy system early on in the project to eliminate initial fears of interacting with "the beast" and thus establishing it as the important source of information that it is ("Cleanup in the legacy system" Section 4.1.1, "Elicit unused assets"). Another focus point is the requirement for early and thorough piloting to validate the chosen solutions directly with the realities imposed by the legacy system ("Build one to throw away"). Furthermore, ensuring a clearly defined leading system in terms of data ownership across (incremental parts of) legacy and target data sources has been identified as a key success factor ("Leading System"). The need for future research in this area ranges from augmenting the pattern repository to validating individual patterns through additional case studies.

5.2 Anti Patterns

As a contrast to the previously presented best practices, anti patterns or smells have been identified. In industry a strong tendency towards "silver-bullet" solutions, like automated source-code transformation ("Automated Transformation") [45], is observable. On an architectural level, every new style generously promises to finally solve all problems of prior generations ("SOA & Microservice Myths, Section 4.2.2). Little attention is paid to previous experiences and remarkably little to the long term consequences [10, 45]. The situation is further complicated by a perceived publication bias and a strong reluctance in industry to openly talk about negative lessons learned and complete project failures, which therefore represents a key potential for future research.

5.3 Skill

The lack of an adequately skilled and willing workforce to maintain legacy systems is one of the most frequently named reasons for legacy system migration [30, 39, 45]. For example Bennett already noted in 1995 a limited supply in developers for assembly code [8]. However, we make the case that on the one hand young IT professionals urgently need to learn about legacy technologies not only to be able to support or retire such systems, but also to better understand the genesis of modern technology. On the other hand we argue that software maintenance and legacy system migration need to receive a much higher focus in curricula to prepare the next generation for the legacy challenges they will surely face once in contact with reality ("Learn Legacy Technologies"). Future research is needed to understand the required skill sets for dealing with legacy systems and to learn about suitable curricula changes that better prepare students for the reality of legacy technologies.

5.4 Methods

The topic of "methods" discusses the necessity to adapt established approaches to the specifics of a legacy system (migration) project. The two key points in this respect are the need for established methodologies prior to the commencement of the migration effort on the one hand and on the other hand the requirement to adjust them to the unique challenges presented by this type of project ("Tailored Development Process"). In addition, systematic approaches to understand the proper handling of legacy systems during the decommissioning phase have been identified ("Repeatable Approach"). The uncertainties of migration projects need to be accounted for in development methodology as well as project and risk management. One area of interest for future research are possible combinations of requirements engineering and reverse engineering, including, but not limited to, the context of agile development methodologies. Another should focus on methods and processes for retiring legacy systems.

5.5 Strategy

Strategic alignment in our context is about making sure the individual legacy system migration effort is embedded in a sufficiently complete and long term big picture following an overarching business strategy. This means on the one hand to lay out the necessary roadmap to give everyone a common vision to aspire to ("IT Strategy & Enterprise Architecture"). On the other hand established structures for decision making according to the previously laid out goals are required. Furthermore, the need for continuous evaluation and evolution of the target as a whole has to be acknowledged ("Strong Architect"). Due to the lack of easily communicable and visible customer value, LSM projects need continuous and sophisticated technical, organisational and political leadership for success. However, it is crucial to avoid masking legacy migration aspects of larger scale digital innovation projects, as this promotes neglecting inconvenient truths until the very last moment ("Planning Migration Projects, Section 4.3.4). Future research is needed especially in the area of creating awareness for the cost of hidden or ignored legacy assets and illustrating the connection with the reduced abilities to innovate and rapidly adjust in a volatile business environment.

6 VALIDATION

Alfred Korzybski already pointed out: "A map is not the territory it represents, but, if correct, it has a similar structure to the territory, which accounts for its usefulness." [25]. For early validation, we selected three papers related to legacy system migration from ICSME 2018 industry track and put them into context with the proposed topology. On the one hand this allows to validate the existing codes by applying them to new content. On the other hand this approach allows to identify missing codes and therefore potentially white, unexplored areas (gaps) in our thematic map.

De Marco et al. [30] present an experience report on the automated translation of a LS from COBOL to Java migrating it from a mainframe to a Linux-based platform. They highlight the need for a deep understanding of the COBOL language and mainframe concepts as well as difficulties in recruiting Java developers to maintain automatically translated source code as primary obstacles towards successful long term maintenance. This case study aligns with our findings in the topology areas of best practices, anti patterns and skill and has been used to refine the generalisations on legacy technology skill and automated transformation. Two gaps could also be identified. The first gap is around test automation and the lack of existing test cases for the legacy system. The second gap is a the prevalence of batch-processing in legacy systems. In addition, difficulties in performing accurate estimates for planned work were noted.

Florez-Ruiz et al. [20] present a successful legacy system migration using a black-box screen scraping approach. The utilised patterns fit well into the best practices area, as a way of interacting with the legacy system, similar to the approach used in the airport case in this study. The presented approach for automated verification through comparison is not yet covered by our proposed topology. Unfortunately no information was given on how the migration affected the maintainability of the resulting system.

Leemans et al. [27] propose a software process analysis methodology based on runtime analysis of a legacy system. This matches analysis techniques employed in both the airport and government cases of our work and supports the generalisation focused on decommissioning legacy systems. However, as with quality assurance in the previous two samples, the topic of knowledge extraction and program comprehension for legacy system analysis is not currently covered in our topology.

In addition to mapping existing publications we successfully used the topology at our research group to identify focus areas for future research as well as the necessity of additional supporting research. A key finding was the lack of a clearly defined classification system (taxonomy) for LIS. As another example we have identified a lack of case studies on the long term modifiability and maintainability of legacy information systems that have undergone automated source code transformation and are now in production, which prompted us to publish our own experience in "Automated Code Transformations - Dealing with the Aftermath"[42].

7 CONCLUSION

The research field of legacy system migration is not new by any means. Standard works like the book by Brodie and Stonebraker [15] as well as the taxonomy by Chikofsky and Cross [17] were established in the 90s and still hold up well. The literature was later complemented with the intensive learning phase from the Y2K frenzy. Examples include the books by Ulrich [46] and Seacord [35]. A roadmap for the upcoming decades was suggested by Bennett and Rajlich [9]. They conclude that "Software evolution needs to be addressed as a business issue as well as a technology issue, and therefore is fundamentally interdisciplinary." and furthermore state "Too much focus at present is on the technology, not on the end-user."

This paper has presented the results of a thematic analysis as a set of 12 extracted generalisations distributed on a technical, architectural and organisational level. All of the generalisations are grouped into five topics that form the proposed topology as a thematic map for legacy system migration as visualised in Figure 1. This topology provides means for navigating the subject of legacy system migration and already highlights key topic areas as well as important aspects within. Both researchers and practitioners will benefit from it individually. However, facilitating communication between the two groups is the primary achievement. It therefore represents the main contribution and at the same time provides the direction going forward. For future work this topology can be used to identify blank areas that need additional charting both in forms of academic research and industrial experience reports. This further exploration can be done utilising existing as well as new and targeted publication material. We expect that future contributions to this work will add new content on the level of generalisations and potentially create whole new topics.

Apart from facilitating communication and exchange between the industrial and research communities we believe that this topology can also be of value in several other areas. It can be used to elaborate or tailor software development processes and project management methodology for the specific needs of LSM projects. It can be used in education to give students an overview of topics in the space of legacy systems and different methods and strategies in dealing with them. Most importantly, however, it can serve as a

guiding structure for the elaboration of software architectures and maintenance methodologies that will ultimately lead to longer lifespans of our software systems by ensuring long term maintainability and modifiability.

Our initial validation of the topology against recent case studies highlighted two important aspects. On the one hand these case studies can be used to both strengthen the argument for a generalisation and to refine the findings within. On the other hand it helps to clearly draw the boundaries of the topology. Two cases show significant overlap with the cases utilised in this work. However, all three papers used in the validation also cover ground on the topics of knowledge extraction or quality assurance that are currently intentionally not covered by the presented topology at this time. Future work is therefore needed to elaborate clear boundaries to these scientific research areas. In addition, we plan to further validate the topology by conducting a survey amongst subject matter experts from both industry and academia.

We are still at a level of professionalism where legacy system migration can be best compared with finding a way through a dense jungle. Rarely do we truly know a route, let alone the best route. Too many go into the jungle, find something resembling what they were looking for and then pave the way back out to sell it as the solution. Equipped with a map highlighting the experience of fellow explorers and the knowledge of the uncertainties ahead preparing us for possible detours and backtracking we should be increasingly able to reliably find a way out of our entangling legacy.

REFERENCES

[1] Fabio Abbattista, Alessandro Bianchi, and Filippo Lanubile. 2009. A Storytest-Driven Approach to the Migration of Legacy Systems. In *Agile Processes in Software Engineering and Extreme Programming*, Pekka Abrahamsson, Michele Marchesi, and Frank Maurer (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 149–154.

[2] M. Abi-Antoun and W. Coelho. 2005. A Case Study in Incremental Architecture-Based Re-engineering of a Legacy Application. In *5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*. 159–168. https://doi.org/10.1109/WICSA.2005.5

[3] R. Akhtyamov, R. Vingerhoeds, and A. Golkar. 2018. Measures and approach for modemization of existing systems. In *2018 IEEE International Systems Engineering Symposium (ISSE)*. 1–8. https://doi.org/10.1109/SysEng.2018.8544427

[4] Johannes Bach and Martin Schulze. 2008. Das Debeka-Projekt MiKe âĂŞ Migration der Debeka-Kernanwendungen von Bull/GCOS8 auf AIX. In *Software archeology and the handbook of software architecture*, Rainer Gimnich, Uwe Kaiser, Jochen Quante, and Andreas Winter (Eds.). Gesellschaft fÃijr Informatik e. V., Bonn, 21–33.

[5] Armin Balalaie, Abbas Heydarnoori, and Pooyan Jamshidi. 2016. Migrating to Cloud-Native Architectures Using Microservices: An Experience Report. In *Advances in Service-Oriented and Cloud Computing*, Antonio Celesti and Philipp Leitner (Eds.). Springer International Publishing, Cham, 201–215.

[6] A. Balobaid and D. Debnath. 2017. An Empirical Study of Different Cloud Migration Techniques. In *2017 IEEE International Conference on Smart Cloud (SmartCloud)*. 60–65. https://doi.org/10.1109/SmartCloud.2017.16

[7] Camila Bastos, Paulo Afonso Junior, and Heitor Costa. 2016. Detection Techniques of Dead Code: Systematic Literature Review *(SBSI 2016)*. Brazilian Computer Society, Article 34, 8 pages. http://dl.acm.org/citation.cfm?id=3021955.3021998

[8] Keith Bennett. 1995. Legacy Systems: Coping with Success. *IEEE Software* 12, 1 (1995), 19–23.

[9] Keith H. Bennett and Václav T. Rajlich. 2000. Software Maintenance and Evolution: A Roadmap *(ICSE '00)*. ACM, 73–87. https://doi.org/10.1145/336512.336534

[10] John Bergey, Dennis Smith, Scott Tilley, Nelson Weiderman, and Steven Woods. 1999. *Why Reengineering Projects Fail.* Technical Report CMU/SEI-99-TR-010 ESC-TR-99-010. Software Engineering Institute, Carnegie Mellon, Pittsburg, PA, USA.

[11] Mario Bernhart, Andreas Mauczka, Michael Fiedler, Stefan Strobl, and Thomas Grechenig. 2012. Incremental reengineering and migration of a 40 year old airport operations system.. In *ICSM*. 503–510.

[12] Kevin Bierhoff, Mark Grechanik, and Edy S. Liongosari. 2007. Architectural Mismatch in Service-Oriented Architectures. In *Proceedings of the International Workshop on Systems Development in SOA Environments (SDSOA '07)*. IEEE Computer Society, Washington, DC, USA, 4–. https://doi.org/10.1109/SDSOA.2007.2

[13] Miguel Botto-Tobar, Richard Ramirez-Anormaliza, Lorenzo J. Cevallos-Torres, and Edwin Cevallos-Ayon. 2017. Migrating SOA Applications to Cloud: A Systematic Mapping Study. In *Technologies and Innovation*, Rafael Valencia-García, Katty Lagos-Ortiz, Gema Alcaraz-Mármol, Javier Del Cioppo, Néstor Vera-Lucio, and Martha Bucaram-Leverone (Eds.). Springer International Publishing, Cham, 3–16.

[14] Michael L. Brodie. 1992. The Promise of Distributed Computing and the Challenges of Legacy Systems.. In *BNCOD (Lecture Notes in Computer Science)*, Peter M. D. Gray and Robert J. Lucas (Eds.), Vol. 618. Springer, 1–28.

[15] Michael L. Brodie and Michael Stonebraker. 1995. *Migrating legacy systems: gateways, interfaces & the incremental approach.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[16] Andrés Carrasco, Brent van Bladel, and Serge Demeyer. 2018. Migrating Towards Microservices: Migration and Architecture Smells *(IWoR 2018)*. ACM, 1–6. https://doi.org/10.1145/3242163.3242164

[17] Elliot J. Chikofsky and James H. Cross II. 1990. Reverse Engineering and Design Recovery: A Taxonomy. *IEEE Softw.* 7, 1 (Jan. 1990), 13–17. https://doi.org/10.1109/52.43044

[18] D. S. Cruzes and T. Dyba. 2011. Recommended Steps for Thematic Synthesis in Software Engineering. In *2011 International Symposium on Empirical Software Engineering and Measurement*. 275–284. https://doi.org/10.1109/ESEM.2011.36

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

[20] S. Flores-Ruiz, R. Perez-Castillo, C. Domann, and S. Puica. 2018. Mainframe Migration Based on Screen Scraping. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 675–684. https://doi.org/10.1109/ICSME.2018.00077

[21] G. R. Gangadharan, Eleonora J. Kuiper, Marijn Janssen, and Paul Oude Luttighuis. 2013. IT Innovation Squeeze: Propositions and a Methodology for Deciding to Continue or Decommission Legacy Systems. In *Grand Successes and Failures in IT. Public and Private Sectors*, Yogesh K. Dwivedi, Helle Zinner Henriksen, David Wastell, and Rahul De' (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 481–494.

[22] P. Jamshidi, A. Ahmad, and C. Pahl. 2013. Cloud Migration Research: A Systematic Review. *IEEE Transactions on Cloud Computing* 1, 2 (07 2013), 142–157. https://doi.org/10.1109/TCC.2013.10

[23] Ravi Khadka, Belfrit V. Batlajery, Amir M. Saeidi, Slinger Jansen, and Jurriaan Hage. 2014. How Do Professionals Perceive Legacy Systems and Software Modernization? *(ICSE 2014)*. ACM, 36–47. https://doi.org/10.1145/2568225.2568318

[24] R. Khadka, P. Shrestha, B. Klein, A. Saeidi, J. Hage, S. Jansen, E. van Dis, and M. Bruntink. 2015. Does software modernization deliver what it aimed for? A post modernization analysis of five software modernization case studies. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 477–486. https://doi.org/10.1109/ICSM.2015.7332499

[25] A. Korzybski, Institute of General Semantics, R.P. Pula, and R. Meyers. 1958. *Science and Sanity: An Introduction to Non-Aristotelian Systems and General Semantics.* International Non-Aristotelian Library Publishing Company. https://books.google.de/books?id=KN5gvaDwrGcC

[26] A. Lauder and S. Kent. 2000. Legacy System Anti-Patterns and a Pattern-Oriented Migration Response. In *Systems Engineering for Business Process Change*, Peter Henderson (Ed.). Springer London, 239–250. https://doi.org/10.1007/978-1-4471-0457-5_19

[27] M. Leemans, W. M. P. van der Aalst, M. G. J. van den Brand, R. R. H. Schiffelers, and L. Lensink. 2018. Software Process Analysis Methodology âĂŞ A Methodology Based on Lessons Learned in Embracing Legacy Software. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 665–674. https://doi.org/10.1109/ICSME.2018.00076

[28] Grace A. Lewis and Dennis B. Smith. 2013. *Migrating Legacy Applications: Challenges in Service Oriented Architecture and Cloud Computing Environments.* IGI Global.

[29] Ed Lindoo. 2014. Bringing COBOL Back into the College IT Curriculum. *J. Comput. Sci. Coll.* 30, 2 (Dec. 2014), 60–66. http://dl.acm.org/citation.cfm?id=2667432.2667440

[30] Alessandro De Marco, Valentin Iancu, and Ira Asinofsky. 2018. COBOL to Java and Newspapers Still Get Delivered. In *2018 IEEE International Conference on Software Maintenance and Evolution, ICSME 2018, Madrid, Spain, September 23-29, 2018*. 583–586. https://doi.org/10.1109/ICSME.2018.00055

[31] David Lorge Parnas. 1994. Software Aging *(ICSE '94)*. IEEE Computer Society Press, 279–287. http://dl.acm.org/citation.cfm?id=257734.257788

[32] K. Petersen and C. Wohlin. 2009. Context in industrial software engineering research. In *2009 3rd International Symposium on Empirical Software Engineering and Measurement*. 401–404. https://doi.org/10.1109/ESEM.2009.5316010

[33] Václav Rajlich. 2014. Software Evolution and Maintenance *(FOSE 2014)*. ACM, 133–144. https://doi.org/10.1145/2593882.2593893

[34] Maryam Razavian and Patricia Lago. 2015. A Systematic Literature Review on SOA Migration. *J. Softw. Evol. Process* 27, 5 (May 2015), 337–372. https://doi.org/10.1002/smr.1712

[35] Robert C. Seacord, Daniel Plakosh, and Grace A. Lewis. 2003. *Modernizing Legacy Systems: Software Technologies, Engineering Process and Business Practices*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[36] Alex Sellink, Harry Sneed, and Chris Verhoef. 2000. Restructuring of COBOL/CICS legacy systems. *Science of Computer Programming* 45 (09 2000). https://doi.org/10.1016/S0167-6423(02)00061-8

[37] Harry M. Sneed. 1995. Planning the Reengineering of Legacy Systems. *IEEE Softw.* 12, 1 (Jan. 1995), 24–34. https://doi.org/10.1109/52.363168

[38] Harry M. Sneed. 1998. *Objektorientierte Softwaremigration [Object-Oriented Software Migration]*. Addison Wesley Longman, Bonn, Germany.

[39] Harry M. Sneed. 2010. Migrating from COBOL to Java. *2013 IEEE International Conference on Software Maintenance* 0 (2010), 1–7. https://doi.org/10.1109/ICSM.2010.5609583

[40] John Spaces. 2015. 8 Types of Legacy System. https://simplicable.com/new/legacy-systems

[41] Stefan Strobl, Mario Bernhart, Thomas Grechenig, and Wolfgang Kleinert. 2009. Digging deep: Software reengineering supported by database reverse engineering of a system with 30+ years of legacy.. In *ICSM*. 407–410.

[42] Stefan Strobl, Christina Zoffi, Christoph Haselmann, Mario Bernhart, and Thomas Grechenig. 2020. Automated Code Transformations - Dealing with the Aftermath. In *2020 IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2020, London, ON, Canada, February 18-21, 2020, accepted*.

[43] Stefan Strobl, Markus Zoffi, Mario Bernhart, and Thomas Grechenig. 2016. A Tiered Approach Towards an Incremental BPEL to BPMN 2.0 Migration. In *2016 IEEE International Conference on Software Maintenance and Evolution, ICSME 2016, Raleigh, NC, USA, October 2-7, 2016*. 563–567. https://doi.org/10.1109/ICSME.2016.41

[44] T. Tamai and Y. Torimitsu. 1992. Software lifetime and its evolution process over generations. In *Proceedings Conference on Software Maintenance 1992*. 63–69. https://doi.org/10.1109/ICSM.1992.242557

[45] A.A. Terekhov and C. Verhoef. 2000. The Realities of Language Conversions. *IEEE Software* 17 (2000), 111–124.

[46] William M. Ulrich. 2002. *Legacy Systems: Transformation Strategies*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

[47] Arie van Deursen, Paul Klint, and Chris Verhoef. 1999. Research Issues in the Renovation of Legacy Systems. In *Fundamental Approaches to Software Engineering*, Jean-Pierre Finance (Ed.). Lecture Notes in Computer Science, Vol. 1577. Springer Berlin Heidelberg, 1–21. https://doi.org/10.1007/978-3-540-49020-3_1

[48] J. M. Verner, J. Sampson, V. Tosic, N. A. A. Bakar, and B. A. Kitchenham. 2009. Guidelines for industrially-based multiple case studies in software engineering. In *2009 Third International Conference on Research Challenges in Information Science*. 313–324. https://doi.org/10.1109/RCIS.2009.5089295

[49] T. Wagner, C. Brem, S. Strobl, and T. Grechenig. 2019. Challenges in re-Platforming Mixed Language PL/I and COBOL IS to an Open Systems Platform. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 364–364. https://doi.org/10.1109/ICSME.2019.00056

[50] M. Waterman, J. Noble, and G. Allan. 2015. How Much Up-Front? A Grounded theory of Agile Architecture. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. 347–357. https://doi.org/10.1109/ICSE.2015.54

[51] Claes Wohlin. 2014. Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering *(EASE '14)*. ACM, New York, NY, USA, Article 38, 10 pages. https://doi.org/10.1145/2601248.2601268

[52] Robert K. Yin. 2009. *Case study research: design and methods* (4 ed.). Sage Publications, Los Angeles.