Software Migration: A Theoretical Framework.

Santiago Bragagnolo, Nicolas Anquetil, Stéphane Ducasse, Abderrahmane Seriai, Mustapha Derras

23-March-2023; Technical Report: hal-03171124v2

Software Modernisation: Theoretical Framework

Santiago Bragagnolo · Nicolas Anquetil · Stephane Ducasse · Abderrahmane Seriai · Mustapha Derras

Abstract Understanding the modernisation literature systematically and grasping the significant concepts is challenging and time-consuming. Even more, research evolves, and it does it based on the assumption that many words (such as migration) have a single well-known meaning that we all share. Since these word meanings are rarely explicit and their usage heterogeneous, these words end up polluted with multiple and many times opposite or incompatible senses. In this context, we ask the following question: *What would be a sound theoretical framework that relates and gives meaning to the techniques, technologies and concepts required to achieve an iterative, incremental migration process successfully?*. This article contributes multiple bottom-up taxonomies that cover the various concepts that characterise modernisation: Legacy systems, their decline by decadence and obsolescence, the reasons that drive to recover from this decline, the different families of approaches to recover from decline, how each of these families of solutions instruments their processes and the material relation between these processes and the features recognised as key in software engineering: iterativity, incrementality and validity. For building these taxonomies, we propose eleven different research questions to guide the crafting of the taxonomies. To respond to this question, we conducted a carefully detailed Systematic Literature Review protocol resulting in the selection of 30 articles. We extract the qualitative data to respond to the research questions by applying the Grounded Theory approach with open codification. The application of the method produced 756 different codes and an appendix, all of this content available online. This article includes more than 20 definitions, arranged in 5 taxonomies, responding to each of the eleven research questions. It also maps different taxonomies and classifies all our readings with those produced. After discussing the threats to validity, the article finishes with the proposal of a few research directions and a conclusion.

Keywords Software Reengineering · Migration · Modernization · Taxonomy.

Santiago Bragagnolo
Université de Lille, CNRS, Inria, Centrale Lille,
UMR 9189 – CRIStAL France,
Berger-Levrault
ORCID: 0000-0002-5863-2698
E-mail: santiago.bragagnolo@berger-levrault.com

Nicolas Anquetil
Université de Lille, CNRS, Inria, Centrale Lille,
UMR 9189 – CRIStAL France,
ORCID: 0000-0003-1486-8399
E-mail: nicolas.anquetil@inria.fr

Stephane Ducasse
Université de Lille, CNRS, Inria, Centrale Lille,
UMR 9189 – CRIStAL France,
ORCID: 0000-0001-6070-6599
E-mail: stephane.ducasse@inria.fr

Abderrahmane Seriai
Berger-Levrault, France
E-mail: abderrahmane.seriai@berger-levrault.com

Mustapha Derras
Berger-Levrault, France
E-mail: mustapha.derras@berger-levrault.com

1 Introduction

Software modernisation happens. With the fast innovation pace of the software industry, it happens more and more often. The academic and industrial implementations of software modernisation evolve the software and the natural language we use to understand and communicate the knowledge required for conducting such processes.

The broad and heterogeneous cases of migration, as well as the specificity of most of the approaches, threaten the reusability of the existing knowledge by polluting our language with multiple or incompatible definitions. "Legacy system" is a name used to refer to widely different systems from different epochs with different symptoms [12, **?**] as if these systems require the exact solutions. Even what we do understand by migration is unclear: on the one hand, [12] points wrapping to undoubtedly not be a migration approach, on the other hand, [16] cites many wrapping-based migrations. This reality facilitates the production of broad, scattered, and hard-to-systematize literature, impacting the understandability of the subject as a whole: what has been done, which risks have been identified, or how do we position our work on further research works. In our quest to understand, share and contribute scientifically in this domain, we recognise this situation as a problem. To tackle this problem, we propose building bottom-up taxonomies along with an articulating definition of the subject as a theoretical framework grounded on a Systematic Literature Review (SLR).

Taking into account that software modernisation is a kind of software engineering project susceptible to risks and failure, and aware of our software development history ([29]), we expect our framework to know the process, iterativity and incrementality.

Such a theoretical framework is our response to the question *What would be a valid theoretical framework that relates and gives meaning to the techniques, technologies and concepts required to successfully achieve an iterative, incremental migration process?* .

This article contributes multiple bottom-up taxonomies that cover the various concepts that characterise modernisation: Legacy systems, their decline by decadence and obsolescence, the reasons that drive to recover from this decline, the different families of approaches to recover from decline, how each of these families of solutions instruments their processes and the material relation in between these processes and the features that are recognised as key in software engineering: iterativity, incrementality and validity.

This article proposes a contribution based on deep qualitative data analysis of 30 articles. An SLR process selected these articles. Grounded theory has been applied to these articles, producing 756 codes by the open codification method. Phrases of each of the articles have been interleaved into the context of each recognised entity of migration, producing an appendix of 18 pages.

Following, we present the planning and parameters of the systematic literature review protocol (section 2) and the grounded theory codification (section 3). We get after to the definition of a taxonomy (section 4), followed by the literature review and article classification based on the proposed taxonomy (section 5). We identify the threats to the validity of our study (section 6) and contribute a list of research directions on areas that we find to be yet unexplored (section 7). The article finishes with a conclusion on the study (section 8).

2 Systematic Literature Review: protocol definition

To reduce bias and improve the reproducibility of this study, we use the method defined by [26] and refined by the comments done by [28]. This systematic literature review, therefore, relies on a well-defined and measured protocol to extract, analyse and document the results. Figure 1 documents the phases and steps of our systematic literature review.

Since our research questions are qualitative, we use grounded theory as proposed by [43] for data extraction and synthesis.

2.1 Planning

The first phase of the protocol aims to cover three main aspects of the SLR: (I) Explain why it is important to conduct this SLR. (II) Present the research questions. (III) Consider the construction of the search string used for gathering the relevant articles. (IV) Consider the main aspects of the validation of the results.

2.1.1 Evaluate necessity: Motivations and Related work

A migration project is highly constrained by its circumstances. By circumstances, we understand the specific features and environment of the project, such as the kind of system to migrate, reasons, specific objectives, technological destinations, and processes. Our primary motivation is to build a theoretical framework that (i) organizes and systematically defines the different key concepts in a migration project, including the characterisation of the process and the project's circumstances, and (ii) maps the existing solutions to these circumstances. We produced a series of bottom-up taxonomies based on a systematic literature protocol with ground theory as a qualitative analysis method to achieve our objective. These taxonomies aim to unify the language by reusing as many words from the existing literature as possible. Such taxonomy is challenging to build since the same terms are used differently in different migration contexts, *e.g.* only for the term wrapping, used widely in other segments of software

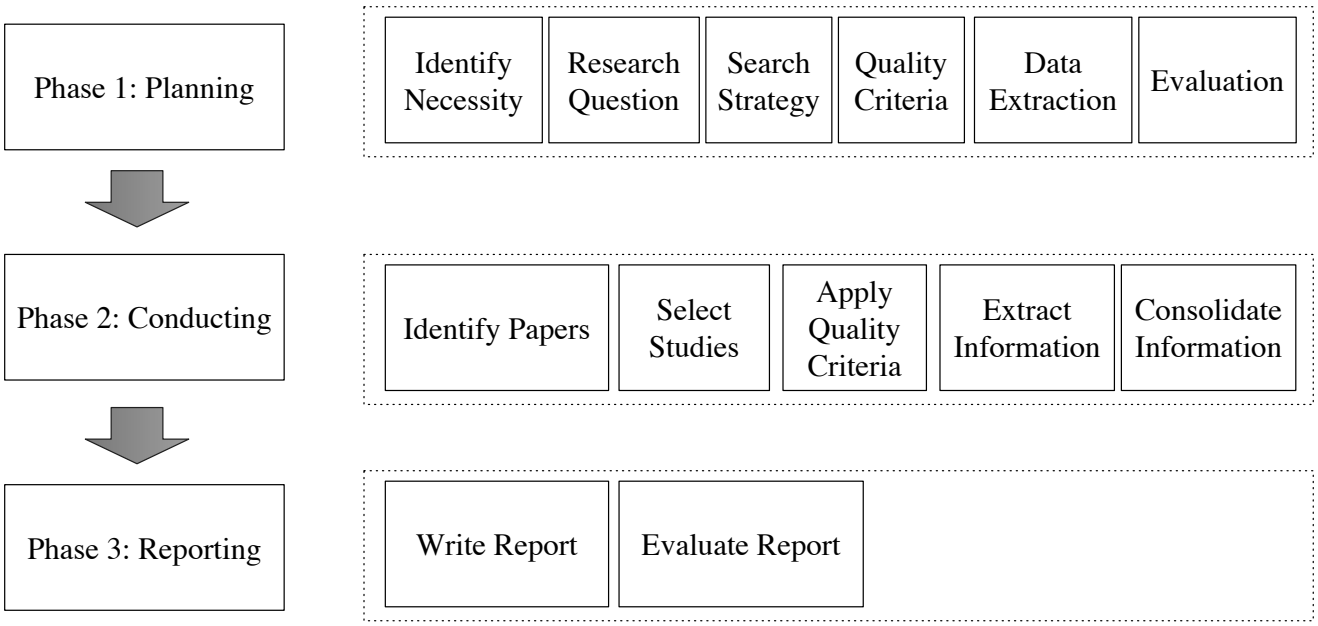

Fig. 1 Systematic literature review process

engineering, we found at least three incompatible meanings: for [24,12] is a modernisation approach along with migration, reengineering, and replacement, for [36] is a step in a migration process, and for [18] is a kind of migration.

Other systematic literature reviews [24] proposes a systematic literature review to respond to the questions: What are the available approaches for modernisation with a cloud perspective? How to propose a generalised framework for assessing legacy systems to adapt to the cloud? [10] Proposes a survey on different approaches for system modernisation. It explains the place of software modernisation in the context of system evolution and provides some existing approaches to UI, Data and Functional modernisation. [16] Proposes a survey on surveys detailing different migration solutions divided into three main parts, the "earlier works" including main strategies to migrate software about the database migration, the migration to "SOA architecture" and finally migration to "Cloud environments". The study provides different approaches and existing tools and projects responding to these approaches.

Qualitative assessments [36] contributes a survey on understanding what a lean and mean strategy for migration to services in the industry is. Explaining how industrial professionals tackle the different parts of the process of such a migration and detailing what kind of technology and strategies they use. [25] proposes a ground theory method applied over a corpus of 26 semi-structured interviews, validated with a poll over the industrial understanding of what is understood as legacy systems and system modernisation.

Migrating Legacy Systems [7] offers the main book on software migration. This book gives insight into planning and guiding a migration project according to chicken little. Different migration cases according to the inherited decomposability of the project are proposed. The book treats all the different parts of a migration: data, functional and UI.

We conclude that our work is relevant since none of the previous works satisfies our need to systematise and classify migration techniques and their circumstances.

2.1.2 Research Questions

Context Our research project takes place in an industrial collaboration for achieving large migration of Microsoft Access applications to web technologies: Angular front-end and microservices backend. This is a broad and heterogeneous software migration project involving different kinds of migration: GUI Migration (Desktop to Web), Architectural migration (Monolithic to Microservice), and Language Migration. Our study intends to discover the different approaches, elucidate the risks and how to mitigate these risks, and understand if the software migration processes respond to iterativity and incrementality as software engineering processes.

RQ#	Question	Aim
RQ1	What is a software modernisation solution?	Provide a definition that reveals and relate the different concepts found in the act of modernising
RQ2	What kind of system requires a software modernisation solution?	Profile the object of the modernisation process
RQ3	What different kinds of software modernisation solutions exist?	Understand the main families of solutions
RQ4	What are the specific objectives of the different software modernisation solutions?	Profile the different material possible objectives
RQ5	Which are the drivers of software modernisation?	Profile the technological and organisational reasons that make modernisation possible
RQ6	How do the different objectives satisfy the drivers?	Understand the degree of driver satisfaction from the point of view
RQ7	What are the different existing approaches from the point of view of knowledge requirement?	Profile the knowledge requirement of each solution
RQ8	Which are the existing families of the process of modernisation?	Comprehend the procedural nature of software modernisation
RQ9	Which elements and concepts are involved in a modernisation process?	Link migration with the artefacts involved
RQ10	How are these processes incremental/iterative?	Link processes with planning
RQ11	What validations/verifications are proposed?	Link processes with guarantees

Table 1 Research Questions

Research context question Following the method proposed by [26], we define the context of our research questions. This question is the main source of keywords for searching for articles. It also directs and relates the different research questions. Our research context question is: *What would be a valid theoretical framework that relates and gives meaning to the techniques, technologies and concepts required to successfully achieve an iterative, incremental migration process?*

Research questions definition Our goal is to apply qualitative analysis to the article selection. For this purpose, we propose eleven different open qualitative research questions listed in the Table 1. It is to remark that most of these questions appeared during the qualitative analysis process, which is expected to happen in the context of qualitative analysis such as ours.

2.2 Search Strategy

As a strategy for searching, we choose what [28] defines as "automatic search". An automatic search is done on one or more search engines giving a search query.

Following the method proposed by [26], we build a keyword-based query to gather articles based on the following steps:

(i) Obtain keywords from the research context question. (ii) Obtain keywords synonyms to be able to widen the search. (iii) Build the search string using PICOC (Population, Intervention, Comparison, Outcomes, Context) [35]

Obtaining keywords and synonyms Responding to the main keywords related to the proposed research questions and obtaining synonyms based on our query-tuning process experience, we propose the following list of keywords and synonyms. We recognise that some proposed synonyms are not linguistically correct, but they give an equivalent insight in the context of our study.

- Software
- Migration / Modernisation
- Transliteration / Translation / Reengineering
- Iterative
- Incremental
- Validation / Analysis / Verification / Solution

Contextualizing The PICOC technique, proposed by [35], aims to contextualise the query building based on understanding our study's elements.

Population: Who/What? The population that we aim to represent in our study are the software migration projects.

Intervention: How? The intervention or process under study are the methods and processes used for software migration.

Comparison: In comparison with? The comparison to be able to measure this work should be made against a canonical software migration definition, which does not exist. Therefore, the comparison does not apply to our work.

Outcome: What we try to accomplish? The production of a series of taxonomies to classify the approaches proposed by the literature.

Context The analysed articles have been written in industrial and academic contexts. We consider then the context to be the industry and academy.

Search string ("migration" OR "modernisation") AND ("reengineering" OR "transliteration" OR "translation") AND ("software") AND ("iterative" OR "incremental") AND ("validation" OR "analysis" OR "verification" OR "solution")

2.2.1 Selection Criteria

We included all papers that comply with the following criteria: (i) In the context of software migration. (ii) Study that proposes a solution, experience, opinion, evaluation, or review of software migration. (iii) Includes awareness of the migration as an incremental or iterative process. (iv) Related to computer science, software engineering.

We systematically excluded any paper that does not comply with the following items: (i) Must be written in English. (ii) Must be available as full-text. (iii) Must be peer-reviewed.

2.2.2 Qualitative Data Extraction

To produce this bottom-up taxonomy grounded on the literature, inspired by [37,25], we decided to apply the Grounded Theory (GT) approach over a systematic literature review to extract what is explicitly and implicitly understood. GT has been used before in systematic literature reviews by many authors. We use this method following the lineaments proposed by [43]. GT is an experimental research method that aims at discovering new perspectives and insights rather than confirming existing ones [8]. We adopted a qualitative research strategy to have an open mind, reduce bias, and let the knowledge emerge from the text rather than find responses to tough pre-existing questions (which implies a bias on how to read and interpret content). The two main techniques used in our study are open coding and axial coding. The open coding process consists in breaking down the content into different parts and labelling them with words or short phrases, with the goal of content discretisation. Axial coding consists of categorising the found open codes.

3 Conducting Protocol

The protocol has been conducted the **29/10/2020**.

3.1 Articles identification and selection process

Figure 2 shows the application of each rule and the number of accepted articles.

We aim to produce a bottom-up taxonomy and link it with more general and standard concepts. To achieve this, we relied on support literature during the confection of the taxonomy. We chose ISO IEC Software Standards due to the international acceptance and the citation of it by some of the selected articles such as [1]. We rely on the standards ISO IEC 25010, ISO IEC 42010, ISO IEC 14764, and ISO IEC 90003 for any definitions related to quality, process, and architecture.

The Table 2 includes the 30 articles obtained by the search string, and, at the end of the table, we find those articles added as support literature.

3.2 Ground theory-based data extraction

Each of the articles has been read systematically two times in two phases. The first phase is the lapse of two weeks, taking overview notes of each reading. The second phase read has been assisted by using qualitative research software MAXQDA2020[3]. Notes taken in the first phase are meant to be dismissed but expected to help to contextualise the researcher.

We applied open coding methodology at sentence/-paragraph levels during the second reading of each article. The sort of codifications at the level of a document is by example *"migration: multiple actor problem"*, *"migration is related with decomposability"*, *"a legacy system may have no external information (doc, manual), or obsolete"*, etc.

After reading of each article, we incrementally reorganised the open coding codes into simple axial coding

#	Year	Title	Publisher
1	2019	GUI Migration using MDE from GWT to Angular 6: An Industrial Case [41]	IEEE
2	2018	An Approach for Creating KDM2PSM Transformation Engines in ADM Context: The RUTE-K2J Case [2]	ACM
3	2017	White-Box Modernisation of Legacy Applications [17]	Springer
4	2016	A Survey on Survey of Migration of Legacy Systems [16]	ACM
5	2015	Modernisation of Legacy Systems: A Generalized Roadmap [24]	ACM
6	2014	How do professionals perceive legacy systems and software modernisation? [25]	ACM
7	2014	A framework for architecture-driven migration of legacy systems to cloud-enabled software [1]	ACM
8	2013	Migrating Legacy Software to the Cloud with ARTIST [4]	IEEE
9	2012	Seeking the ground truth: a retroactive study on the evolution and migration of software libraries [11]	ACM
10	2012	Searching for model migration strategies [42]	ACM
11	2012	A lean and mean strategy for migration to services [36]	ACM
12	2010	Extreme maintenance: Transforming Delphi into C# [6]	IEEE
13	2009	Parallel iterative reengineering model of legacy systems [39]	IEEE
14	2008	Can design pattern detection be useful for legacy system migration towards SOA? [3]	ACM
15	2008	Developing legacy system migration methods and tools for technology transfer [12]	Wiley & Sons
16	2007	OPTIMA: An Ontology-Based PlaTform-specIfic software Migration Approach [47]	IEEE
17	2007	Reversing GUIs to XIML descriptions for the adaptation to heterogeneous devices [15]	ACM
18	2005	Quality driven software migration of procedural code to object-oriented design [48]	IEEE
19	2004	Incubating services in legacy systems for architectural migration [46]	IEEE
20	2003	Network-centric migration of embedded control software: a case study [38]	IBM Press
21	2002	C to Java migration experiences [31]	IEEE
22	2002	A framework for migrating procedural code to object-oriented platforms [49]	IEEE
23	2000	A Survey of Legacy System Modernisation Approaches [10]	DTIC [1]
24	1998	Code migration through transformations: an experience report [27]	IBM Press
25	1997	Lessons on converting batch systems to support interaction: experience report [13]	ACM
26	1997	Reverse engineering strategies for software migration (tutorial) [34]	ACM
27	1996	Strategic directions in software engineering and programming languages [19]	ACM
28	1996	Rule-based detection for reverse engineering user interfaces [33]	IEEE
29	1995	Workshop on object-oriented legacy systems and software evolution [40]	ACM
30	1994	Knowledge-based user interface migration [32]	IEEE
–	2015	ISO IEC 90003 (ISO 9001 applied to Software) [23]	ISO
–	2011	ISO IEC 25010 (ex ISO IEC 9126)[21]	ISO
–	2011	ISO IEC 42010 [22]	ISO
–	2006	ISO IEC 14764 [20]	ISO
–	2002	Object-Oriented Reengineering Patterns [14]	M Kaufmann
–	1990	Reverse Engineering and Design Recovery: A Taxonomy [9]	IEEE
–	1985	Program evolution: Processes of software change. [30]	LAP [2]

Table 2 Initial Dataset

[3] https://www.maxqda.com/

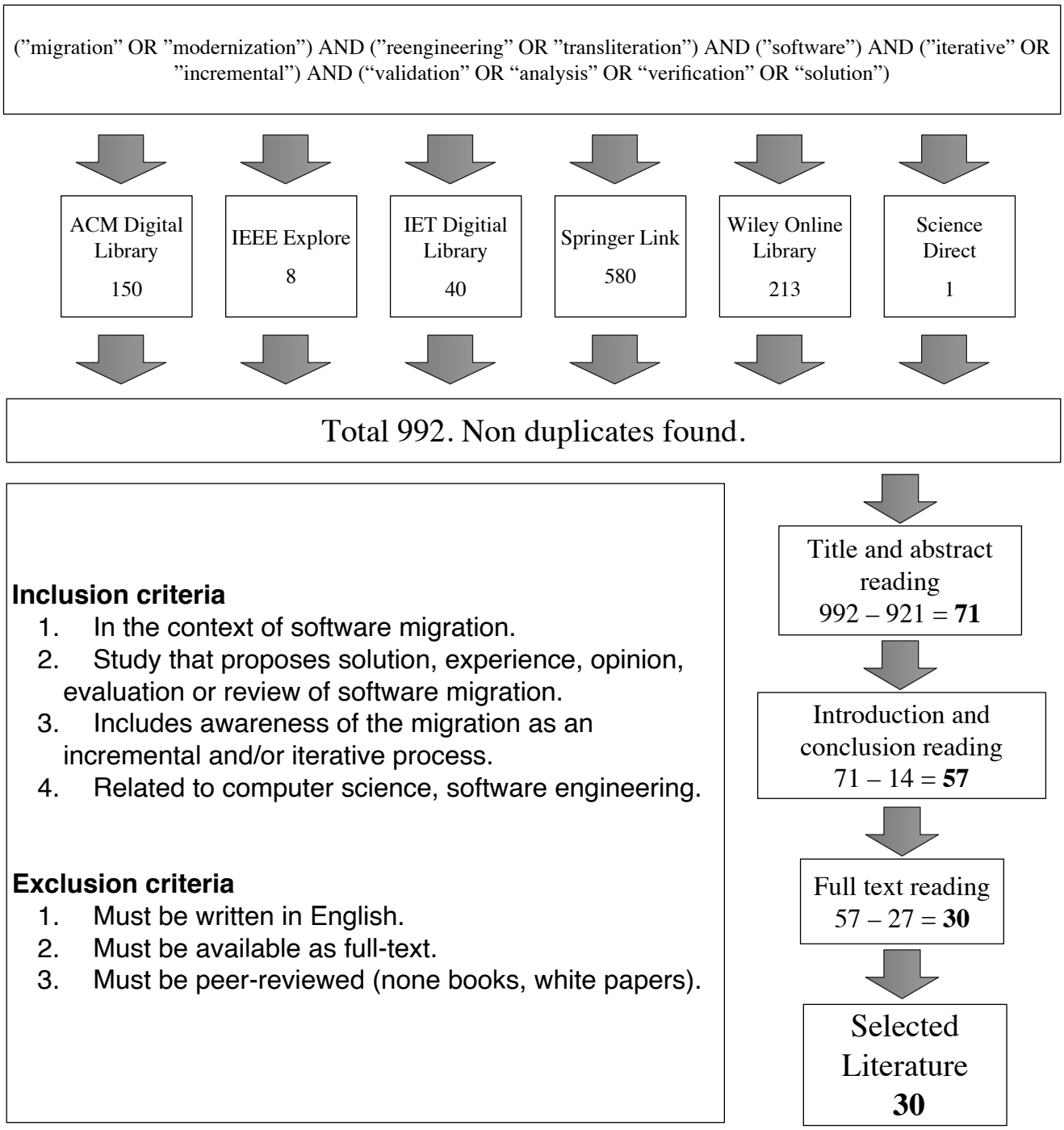

Fig. 2 Application of selection criterias

hierarchies based on the detection of general categories, such as *"migration definition"*, *"migration process implications"*, *"legacy system"*, *"engineering variables"*, etc. Each axial coding iteration implied many times the restructuring of existing coding categories.

The process yielded 756 codes organised on a hierarchical axial coding. During the writing process, for better understanding and writing, based on the open coding, we interleaved explicit text from each paper for each of our taxonomical axes into an appendix document.

The reading notes, the appendix and the coding structure, and the MAXQDA2020 project can be found in the following GIT repository `https://gitlab.inria.fr/sbragagn/slrmigration/`. The appendix can also

be found submitted in the HAL platform `https://hal.inria.fr/hal-03169377`.

4 Reporting: a literature emergent bottom-up taxonomy

As explained by [45], taxonomies' principal utility is communicating knowledge, providing a shared vocabulary, and helping structure and advance knowledge in the field. Taxonomies can be developed in one of two approaches; top-down, also referred to as enumerative, and bottom-up, also referred to as analytico-synthetic. The taxonomies created using the top-down method use the existing knowledge structures and categories with established definitions. In contrast, the bottom-up approach's taxonomies are created using the available data, such as experts' knowledge and literature. We propose a bottom-up taxonomy based on the analysis and synthesis of the selected literature.

The following subsections will clarify some basic definitions required to contextualise the study. We answer each research question by extending concepts or defining and relating taxonomies.

4.1 Software System definitions

System. Following the definition given by [22] man-made entities that may be configured with one or more of the following: hardware, software, data, humans, processes (e.g., processes for providing service to users), procedures (e.g., operator instructions), facilities, materials and naturally occurring entities. We also add that all these entities and their relationships configure what we understand as the environment where our software takes place.

Architecture & Design. Following the definition given by [22], we recognise architecture to be the fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and principles of its design and evolution. Its elements: the constituents that make up the system; the relationships: both internal and external to the system; the principles of its design and evolution. Furthermore, we differentiate architecture from design: **architecture is outwardly focused** on the system in its environment, whereas **design is inwardly focused** once the system boundaries are set.

From this, we can infer that the architecture migration also entails the evolution of the inward design, directly impacting the piece of software but also indirectly impacting the implementation of the business rules.

Software Quality. According to [21] we talk about quality from three points of view. The quality is perceived "internally" by measuring the quality of source code and by metrics, documentation, and knowledge on the maintaining organisation. The quality is perceived "externally" by measuring its artefact behaviour. Finally, quality is perceived as "in-use" as the software's capacity to accomplish requirements and adapt to new changes. [21] also spots the inter-relationship of these qualities.

Software Modernity. The modernity of software is related to the distance between the up-to-date techniques and technologies of software development and those used during the development of the source code. An example is if this software cannot profit from using up-to-date technologies and concepts, such as AI, IoT, Blockchain, and microservices.

Software Continuity. The continuity of a piece of software (persistence or permanence) is directly related to the resource allocation policy for its maintenance and evolution. Despite the modernity or the quality, software continuity is related to how much this software is needed and how many resources the owners can afford to keep it working. A direct implication of continuity is incrementing the investment value in multiple aspects: money, time, and knowledge. Lehman et al. [30] proposes the *the law of continuing change*: A program that is used in a real-world environment *must* change or become progressively less useful in that environment.

4.2 **RQ1** What is a software modernisation solution?

We provide a general definition that emerged from our study and is aware of the different elements and concepts involved in a software migration. *Given a legacy system and a driver (which implies an evolution of the given legacy system), a software modernisation solution is a reengineering process (subsection 4.7) of migration or adaptation (subsection 4.4) that applies a specific method subsection 4.6 – which responds to a general approach (subsection 4.6)– in order to achieve an objective (subsubsection 4.5.1) that contributes to the satisfaction of the given driver (subsubsection 4.5.2), by impacting specific parts of the given legacy system(subsection 4.3).*

4.3 **RQ2** What kind of system requires a software modernisation solution?

The constant passage of time and evolution often contribute also with the decline of a system. In our context we recognise two main kinds of decline: (i) the decadence, (ii) the obsolescence.

Decadence . Decadence is the continuous deterioration of the **inherent internal qualities** of a software: unreliable documentation, lack of knowledge, increase of accidental complexity, highly tangled and coupled source code, loss of consistency and cohesion. The decadence of the system **hampers its evolution**. [27] states a crucial fact on this aspect: "Some system components are not owned by any member of the development team and are therefore very difficult to maintain. Not surprisingly, the team is reluctant to perform radical changes to its structure since this may negatively affect its overall performance.".

Obsolescence , we understand the changes in the environment where our software exists and how these changes affect the **inherent external qualities** of the software: the apparition of new technologies and paradigms, or the deprecation of dependent technologies impacts on the way a system interacts with other systems: Apparition of online services competition, the apparition of radically cheaper infrastructure, the deprecation of dependent software (libraries, compilers, etc.), the out-of-production of required hardware platforms, changes in business legislations, etc. The obsolescence of the system **justifies and causes its evolution**. [38] exposes the urgency of system evolution in a project requiring enabling network communication on a system that includes embedded software since this requirement implies hardware-level modifications.

Legacy systems. These are successful systems with a long continuity, which cannot accomplish strategic decisions due to some grade of decadence or obsolescence at some part of the system. [12] spots the importance of systems that runs 24/7. [27] points out that software that migrates "are often mission-critical for the organisation that owns and operates them". One of the interviews in [25] proposed a definition: "My definition of a legacy system is systems and technologies that do not belong to your strategic technology goals". This is a weak definition, but it points out something important: a system can become a legacy with a simple strategic change. Demeyer et al. Deme02a says that a legacy system is a constantly evolving system critical to your business and cannot be upgraded or replaced except at a high cost. The constant evolution of this system is what exposes it to decline.

By external parts, we refer to all the material and intellectual elements that may affect or constrain the impacted source code. Internal parts refer to the crafting quality aspects that may affect or constrain the impacted source code. The following list exposes the different external and internal parts found during the SLR.

- External
  - Architecture
  - Third party (Libraries – Frameworks)
  - Runtime
  - Hardware
- Internal
  - Design
  - Concerns
    - UI
    - Data
    - Functionality
  - Used APIs / ABIs
  - Language – Paradigm
  - Source code

What kind of system requires a software modernisation solution? (i) legacy system due to third-party library obsolescence, (ii) legacy system due to an obsolete programming language, (iii) legacy system resulting in decadent source code, (iv) legacy system due to decadent design.

4.4 **RQ3** What different kinds of software modernisation solutions exists?

We propose two large families of solutions first, including all possible solutions concerning the whole system. **Reengineering & Replacement**:

Figure 3 gives a general overview of the Solution's taxonomy. In grey, we find those concepts that are not further explored in this article. Those nodes are not explored because it is out of scope, and the selected literature does not provide experience on this family beyond acknowledging its existence. Nevertheless, their inclusion and definition are maintained to insist on what is not modernisation.

Reengineering Reengineering is all processes based on modifying a previously existing system.

Modernisation All processes recover a system from **Obsolescence**, achieving better integration with the environment and enhancing the external quality of our

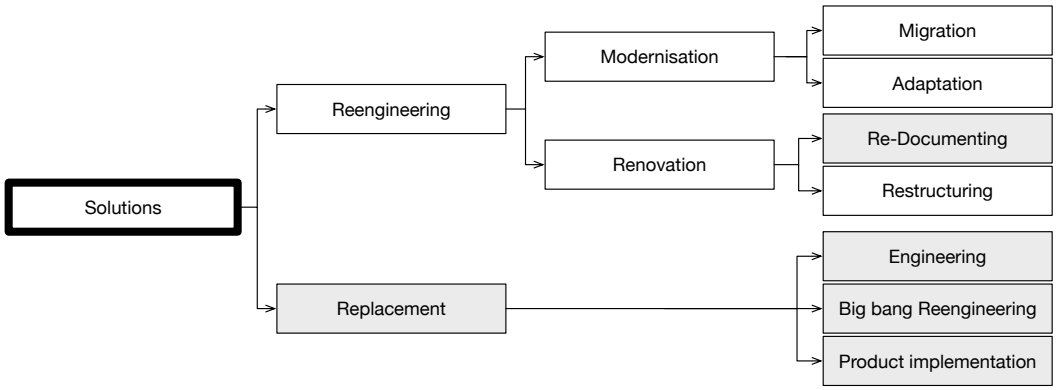

Fig. 3 Solution's Taxonomy Overview (In grey, we find those nodes that are not further explored in this article).

system. These processes affect external and internal elements of a Legacy System. *Adaptation* is a Modernisation process that enables using a new technological environment without threatening currently used technology. There are many kinds of adaptations, from, e.g., (i) [19], proposing to compile C in C++, to be able to add new code in an object-oriented fashion, to, e.g., (ii) [38] proposing to modify hardware, or, e.g., [15] who adapts a website to be rendered on different running devices. *Migration* is all Modernisation process that moves from one *Provenance* technological environment to a *Destination* technological environment that is in relation to mutual exclusion (either for technological or strategical reasons) with the *Provenance* environment. There are many kinds of migrations, like source code translation proposed by [6,27,31], GUI migrations proposed by [41, 17,32], or library migration [47,11,31]

Renovation Renovation is all processes that recover a system from **Decadence**, achieving better internal quality or a better understanding of the internal structure. These processes affect only internal elements of a Legacy System. *Restructuring* is all Renovation processes issued over the source code (e.g., refactoring). *Re-Documenting* is all Renovation process that produces new or enhances existing documentation of the code, such as writing manuals, specifying processes, and formalising requirements. "The spectrum of reengineering activities includes re-documentation, restructuring of source code, the transformation of source code, abstraction recovery, and reimplementation." [34]

Replacement Replacement is all processes that discard the existing system and establish a different one. *Engineering* is a Replacement process that creates a new system based on understanding the current requirements. *Big-bang Reengineering* is all Replacement processes that create a new system based on the understanding of the historical requirements by reverse engineering an existing system. Proposed and rejected by many of the articles, such as [6] *Product implementation* is all Replacement processes that implement and customise a Commercial off-the-shelf (COTS) system to solve the current requirements. E.g., [38] proposes as possibility an off-the-shelf product.

What different kinds of software modernisation solutions exist? (i) legacy system, due to third-party library obsolescence, requires Migration. (ii) legacy system, due to an obsolete architecture, requires Adaptation. (iii) legacy system, due to decadent source code, requires Re-Documenting. (iv) legacy system, due to decadent design, requires Restructuring.

4.5 Objectives & Drivers

As a metaphor to understand the general mindset of these two words, we explain the case of a hammer. A hammer is a tool consisting of a weighted "head" fixed to a long handle swung to deliver an impact to a small area of an object. Different kinds of hammers fit different *objectives* depending on the context: to drive nails into wood, to shape metal, or to crush rock. The direct *drivers* of using a hammer often relates to larger processes with more general targets: build a shelf, forge a sword, etc.

4.5.1 ***RQ4*** *Which are the specific objectives of the different software modernisation solutions?*

In this context, the objective is the expected specific outcome of applying a solution. In our SLR, we found the following objectives:

Migrate Data Access Protocol : Modify the data accessing architecture.

Centralized to distributed database : Distribute or replicate the databases.

Migrate text UI to GUI : Create a GUI able to interact with a text-based tool.
Migrate to Service : Offer existing functionalities as a service.
Client-Server To Web : Migrate a client-server architecture to web architecture.
Enable Cloud : Execute existing software on a cloud environment.
Migrate data management to RDBMS : Delegates the internal concern of data storage to a third party.
Paradigm Change : Transform code organisation and semantics from procedural to object-oriented programming.
Translation : Translate source code from one language to another.
UI Translation : Translate the UI representation from one model to another.
Library Migration : Change the API to delegate a concern to a given library/framework.
KDM to PSM : Automatic generation of a platform-specific model from a Knowledge discovery model.
Adapt UI to multiple devices : Provide different UI representations depending on the rendering device.
Adapt the embedded system to support networking : Implement network communication between devices.
Adapt batch to support interactive control : Adapt batch to support interactive control

4.5.2 ***RQ5*** *Which are the drivers of a software modernisation?*

Overview Figure 4 gives a general overview of the Driver's taxonomy.

Reengineering processes are often expensive in time and money. The expected outcome is often a system that responds to the same problem but differently. Significant spending of resources for a system that does not solve new problems is often left for critical situations when the continuity of the software is seriously threatened. Drivers for conducting such enterprises are related with some implication of the nature of the "legacy systems" (by nature, we refer to the external and internal characteristics that make this system a legacy system, as exposed on subsection 4.3).

Our bottom-up taxonomy groups the findings on drivers into the groups of **Direct & Indirect** in the context of **Modernisation & Renovation**. We focus then on the **Evolutionary** processes of **Modernisation & Renovation** to recover a legacy system from **Obsolescence & Decadence** to respond to **Direct & Indirect** requirements. We do not analyse drivers on the **Replacement** processes because the selected literature provides no experience or hard evidence on this family beyond acknowledging its existence.

Direct drivers Direct drivers are all decisions that find their reasons in the **immediate impact** of applying a specific solution. Most of the drivers in this branch respond to strategic technological and/or system quality objectives.

Indirect drivers Indirect drivers are all decisions that find their reasons in the **expected implications** of the impact of applying a specific solution. Most of the drivers in this branch respond to strategic organisational objectives.

4.5.3 Modernisation related drivers

- Direct
  - Move from a dying technology [41,11]
  - Enable new architectural variables (scalability, elasticity, availability) [1,4,24]
  - Enable new features (interactivity, run on new devices) [15] [31]
- Indirect
  - Ease the process of hiring qualified employees [40]
  - Provide a competitive service [24,1,4]
  - Enable new businesses / markets [15,46]
  - Enhance developers' performance [27]
  - Reduce costs [27]

4.5.4 Renovation related drivers

- Direct
  - Enhance architectural variables by design (scalability, elasticity, availability) [1,4,24]
  - Enhance design quality variables (decomposability, maintainability, understanding, reliability) [19, 15,38]
  - Recapitalize knowledge [12]
- Indirect
  - Enhance developers' performance [34]
  - Flat the learning curve for newcomers [40]
  - Enhance business adaptability [33]
  - Recapitalize knowledge[12]
  - Reduce costs[12]

Which are the drivers of software modernisation? (i) Legacy system due to third-party library obsolescence, requiring modernisation to move out from a dying technology. (ii) Legacy system due to an obsolete architectural paradigm, requiring modernisation because of the low availability of experts for hire. (iii) Legacy system

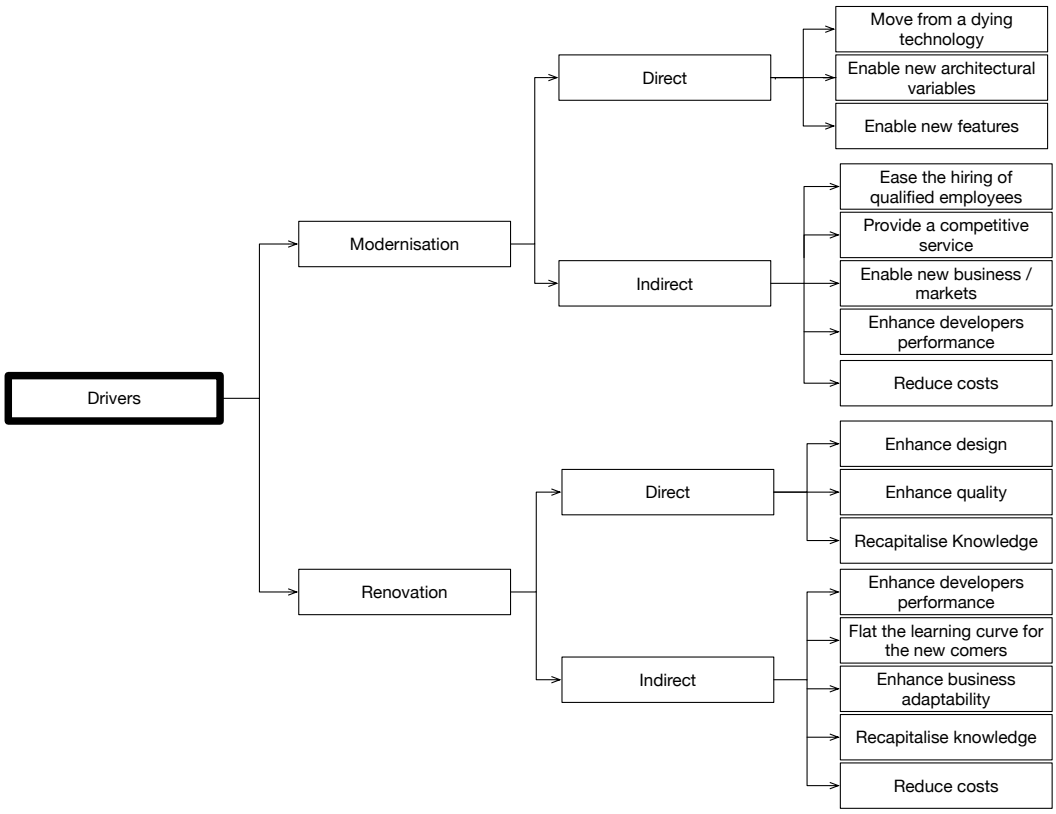

Fig. 4 Driver's Taxonomy Overview

due to decadent source code, requiring renovation to run on new devices. (iv) Legacy system due to decadent design, requiring renovation to enhance maintainability.

4.5.5 ***RQ6*** *How do the different objectives satisfy the drivers?*

Objectives and Drivers are two orthogonal notions, but objectives can be mapped to one or more drivers according to the circumstances of a specific project. Table 4 [4] shows the Cartesian product between those objectives mapped to the drivers by the literature.

Please note that Table 4 includes **only** those objectives directly treated by our articles when our objective list includes all those objectives plus the proposed by different surveys. All the objectives are mapped to one or more drivers. Still, some drivers have not found an explicit solution to the proposed methods; those drivers are not included in the table. The table includes the acronym NER which stands for Not Explicit Relationship. This means that the work did not provide an explicit link between a solution and a specific driver. In the other cases, the crossing points give us the Contribution of the solution's objective to the driver.

[4] Given the size of the table, it has been annexed at the end of the manuscript

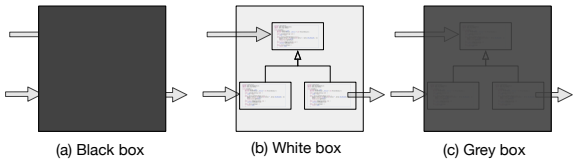

Fig. 5 Approaches

4.6 **RQ7** Which are the different existing approaches from the point of view of knowledge requirement?

In our study, we found three big families of technical approaches that tackle most of the reengineering challenges in our field. They are those based on a deep understanding of the *Provenance* system/subsystem, those based on the analysis of input and outputs [10] and those based on hybrid approaches.

4.6.1 Black-box Approaches

Black-box or external approaches(Figure 5 (a)) are named after the fact that they disregard the internal composition of the system and focus on understanding the inputs and outputs of a legacy system within an operating context to gain an understanding of the system/-subsystem interfaces. These approaches often imply low

or no modifications to the existing system. Black-box approaches are often based on wrapping techniques.

Wrapping surrounds a piece of software with a software layer that hides unwanted complexity and exports a new interface. Wrapping removes mismatches between the interface exported by a software artefact and the interfaces required by current integration practices. Since wrapping impacts over devices aiming to enable communication, it is only applicable on the different levels of interoperability: Third-party solutions, exhibited API/ABI, and Architecture. Figure 6(a) shows a schematic of a hypothetical wrapped system. As the image shows, wrapping many times implies the development of new code that articulates the black box into the new environment.

4.6.2 White-box Approaches

White-box or internal approaches(Figure 5 (b)) are named after the fact that they consider the internal composition of the system. Often based on an initial reverse engineering process to gain a deep internal understanding of the *Provenance* system/subsystem. This process usually aims to identify components and relationships at different levels of abstraction (classes, patterns, dependencies, etc). Automatic and semi-automatic white-box techniques are typically based on producing representational models, such as meta-models or ontologies. These approaches are often implied a high amount of modifications to the existing system. White-box approaches are often based on transforming techniques.

Transforming produces a software component semantically equivalent to an existing one. This produced software component responds to an equivalent level of abstraction and exhibits different technological features or assumptions. Since a transformation impacts the source code directly or indirectly, it can be applied to all the software's different internal and external parts. Architecture, Design, Language, exhibited and used API/ABI, Paradigm, Deployment environment, and Third-party products. Figure 6(b) shows a schematic of a hypothetical transformed system. As the image shows, transforming implies modifying all the internal design and even adding or removing existing source code to articulate the system into the new environment.

4.6.3 Grey-box Approaches

Grey-box or hybrid approaches (Figure 5 (c)) are those approaches that use internal approaches for enabling certain granularity on external approaches or using general external approaches to reduce risks and not operational time of invasive internal approaches. On the first kind, we find most of the proposals of migration of software to service architectures using internal approaches to recognise parts of a system and decompose it, enabling to wrap parts of a system instead of the whole system [16]. We found the usage of the second approach, especially on modernisation processes that are required to delegate what once was a system concern to a third-party product. Such is the case of the migrations from language-support data management to third-party products (most of the iconic cases come from the migration from COBOL registry files to RDBM systems) [12].

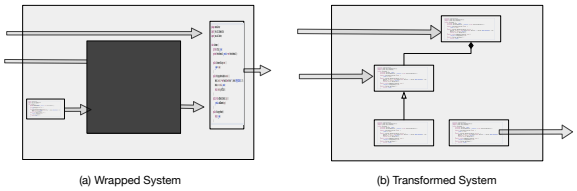

Fig. 6 Produced artefacts schematics

4.7 **RQ8** Which are the existing families of the modernisation process?

Software migrations are often lengthy and highly risky enterprises [36, 25]. Such projects often deal with legacy systems that suffer from both Decadence and Obsolescence on multiple artefacts.

In short such projects are bound to a lot of detailed variables that impose the instrumentations of many times ad-hoc processes, which makes it especially hard (if not impossible) to generalise practical procedures (as the suitable process we understand an exhaustive definition able to fit all possible cases of modernisation and renovation), but only some process form for the sake of knowledge organisation.

According to our study of the literature, we recognise that, in general, software migration responds to two families of process forms shown on **??**: Attached and detached, relative to the target project.

Detached processes. Detached process often responds to variations of a *phased* process or the butterfly method proposed by Wu et al. [44]. This model is related to processes that take as input a system and give as output a new system that should comply with the old and new specifications.[42, **?**, **?**, **?**]. We name this kind of process *detached* because the process does have any relation

with the target other than generating it. The produced target is not used as input.

Disadvantages: Due to the forking nature of the process (which produces a new system), it threatens the maintenance and development of new features. This process requires producing a new system based on the original system [5, **?**]. This is often split into parts like a first-class citizen: classes, modules, widgets, etc. This kind of granularity imposes the entity over the feature, requiring the process of whole entities to produce a feature; this is likely to increase the delivery time. Products may take much time to be implemented, seen and valorised.

Advantages: On the other hand, it does not threaten the quality or stability of the *Provenance* system. The *Provenance* system can still be used as it is [5, **?**].

Attached processes. These processes respond to variations of the classical Spiralling forward-engineering model [20] or the chicken little method proposed by Brody et al. [7]. Related to the nature of a process that takes as input a system and gives as output the same system but modified. [46, 13]

Disadvantages: Due to the continuously integrating nature of the process, it is essential to remark that it threatens the stability and internal consistency of the system [5, **?**].

Advantages: Each process iteration may apply arbitrary size transformations [5, **?**]. The smaller the size of a modification over a running system, the easier to test, deliver and deploy new versions. Regular delivery increases the access to user feedback and the visibility and valorisation of the migration process.

As shown in Table 3, we find that migration responds to attached and detached processes, Adaptation, in our findings, responds only to Attached processes, as it is about adding support to a new feature. On the renovation side, we find both kinds of processes. Below we present each step.

4.7.1 ***RQ9****Which elements and concepts are involved in a modernisation process?*

Plan Activities in this phase are typically conducted to define the reach and expectations of the process at the operational level[23], including risk and feasibility assessment. [34] Recognises that risk is related to planning "Minimizing the migration risk is a key requirement. The most common strategy is an incremental approach to minimise the risk". [36] Remarks the importance of understanding "Associating costs and risks to core activities makes the core an even more powerful tool for planning how to migrate."

Understand Provenance System Activities in this phase are typically conducted to acquire knowledge of the system. [1, 4]. These activities are accomplished manually, semi-automatically, or automatically. The proposed activities range from intellectual understanding (based on interviewing team members of the project, reading documentation and or code [36]), to computational models built from reverse engineering (as those proposed especially by model-driven engineering [2, 41, 17, 42, 6]) or ontological methods [47], that propose a computational representation of the semantics and structures of the system. This knowledge is required at many levels, from management and planning (to measure risk, to prioritise tasks, etc. [36, 11]) to the input of automatic/semi-automatic algorithms with many usages such as code enhancement recommendations, language translation, etc. [42, 6].

Understand Expectations of the Destination System Activities in this phase are generally conducted to acquire knowledge of the *Destination* system. [1, 4]. These activities are typically accomplished manually. The proposed activities are related to understanding how the new system will behave and interact with the environment. This knowledge is required to choose a correct and optimal approach [1] for the process, estimating costs, times, risks, and assessing task prioritisation [36, 11].

Transform Knowledge Activities in this phase are normally conducted to work over the acquired knowledge regarding the process expectations. [1] These activities are accomplished manually, semi-automatically, or automatically. The nature, size, and order of the tasks change from white to black-box approaches. Still, these activities range from the intellectual understanding (of the required transformations and re-structuration to apply to accomplish the target expectations of the current process as proposed by [36], to leverage and transform computational models built during the previous step, to fit better on the *Destination* system restrictions [32, 3], or [49] who uses clustering algorithms over models for proposing classes and methods in the context of procedural to object-oriented migrations).

Modify system Specific for attached processes. Activities in this phase are typically conducted to apply the transformed knowledge to the current system. These activities are accomplished manually, semi-automatically, or automatically. The nature of the modification range from modifying manually some asset of the system (source code, documentation, etc.) [36, 3, 13] to the automatic/semi-automatic modification of these assets [47].

Table 3 Process x Solution (NF: Not found)

Process	Modernisation		Renovation
	Migration	Adaptation	Restructuring
Detached	White-box / Grey-box	NF	Refactoring / Transform
Attached	Black-box / Grey-box	White-box / Black-box	Refactoring / Transform

Produce Destination Specific for horseshoe processes. Activities in this phase are typically conducted to use the transformed knowledge to produce a *Destination* system. These activities are accomplished manually, semi-automatically, or automatically. The nature of the product range from the manual creation of the *Destination* system (based on the transformed knowledge), to the automatic/semi-automatic generation of this *Destination* system [42,17,2]

4.8 **RQ10** How are these processes incremental/iterative?

Planning is directly constrained by the ability to break down the process into tasks. The smaller and more independent the task can be, the better. In modernisation and renovation, this may not always be the case. In all our cases, the ability to split the workload into small and manageable tasks requires a high level of decomposability, as pointed by [46,12,27], [39,6] and [1]. And the fact is that the decomposability of a system is related to source code qualities, such as coupling and cohesion (obtained metrics analysis). This means that a decomposable system usually is a healthy, not-decadent system. Since the process takes as input what we named a "Legacy System", this is not likely to be the case. This is why a modernisation process usually requires a tightly interleaving renovation process. [46]. And many other times, renovation is just too expensive in an obsolete environment, and therefore it requires a tightly interleaving modernisation process [49].

To interleave these processes tightly enough to reduce risks, a highly documented and informed iterative strategic plan is required [6]. To obtain this information, we required constant metrics analysis over the system and the evolution of the process as well as from the tasks. One of the essential tasks-metrics is related to validate-ability and testability, which also requires decomposability to be possible.

This is why we conclude that a virtuous circle in between these points is required to reduce the risks. And this virtuous circle is highly likely to require the help of reliable tooling [6,38,40]

In the planning process, we recognise two different levels of planning (as proposed by ISO 9001 [23]: Strategic and Operational.

4.8.1 Strategical planning

Strategical planning is situated on the overall vision of a project of Modernisation & Renovation. At this level, the critical activities are the recognition of "strategic" milestones [6,39], and their linking in terms of interactivity. Strategic milestones in the context of modernisation may imply the recognition of which parts of the system to modernise, and in which order of priority acknowledging dependencies.

Iterativity is a key property to make migration a possible process [6]. This feature is related to the way to define the project's roadmap. It is managed at the strategic level. The most important pillars to ensure iterativity in the context of Modernisation & Renovation are (i) Breaking the project into milestones. [6,39] (ii) Each of the milestones must be independent and testable. [6], (iii)The milestones must be efficiently prioritised. [39] [27] (iv) Each milestone should work on refining the previous milestones. [1] (v) Instrumentation of feedback devices. [6,49]

4.8.2 Operational planning

Operational planning is situated on the vision of one specific iteration of a project of Modernisation & Renovation. At this level, the critical activities are recognised as "operational" milestones and their linking in incrementality. Operational milestones in modernisation may imply the recognition of sprint-length tasks, task dependencies, priorities opportunities of parallelism [39], and the mapping to incremental change and systematic validation of the results.

Incrementality is proposed for reducing operational risks [27]. This feature is related to the way to define the tasks to do to accomplish one strategic milestone. It feeds back to the strategic planning on how the milestone was accomplished. It is managed at the operational level. The most important pillars of incrementality, in the context of Modernisation & Renovation, are:

(i) Deep and systematic understanding of the *Provenance* system is required for task measuring.[34,12] (ii) Tasks must result from the coarse-grained decomposition of larger tasks. [1] (iii) Tasks must be measured, and their impact on the next tasks must be understood.[6] (iv) Tasks outputs must be mergeable with the results produced before and those to be produced after [48] (v) Tasks outputs must be tested. [6] (vi) Instrumentation of feedback devices. [6]

RQ11 *What validations/verifications are proposed?* Validation is required as it is the main feedback for operational planning, informing evolution and increment accomplishment. Validability is managed at the task level. According to the SLR, the most critical pillars of validation and evaluation in the context of Modernisation & Renovation are: (i) Unit testability. The task output must allow instrument tests that prove their behaviour [6]. (ii) Integration testability. The task output must allow being tested on the expected context of usage of the output [6]. (iii) Performance measurability. The task performance must be measured [39,12,27]. (iv) Comparability. In an automatic/semi-automatic transformation context, the task must be comparable with the equivalent manual outcome [49,12]. (v) Correctness. In an automatic/semi-automatic transformation, the tasks must respond to correctness analysis, and testing [32], [6]. (vi) Soundness. In an automatic/semi-automatic transformation context, the tasks must report the same results for equivalent objects. [27] (vii) Understandability. The result of a task must be interpretable for further comparisons with the previous state/ *Provenance* system. [48,12].

5 The impact over the Legacy system

We previously presented a definition for software modernisation to respond to **#RQ1**. *Given a legacy system and a driver (which implies an evolution of the given legacy system), a software modernisation solution is a reengineering process (subsection 4.7) of migration or adaptation (subsection 4.4) that applies a specific method subsection 4.6 – which response to a general approach (subsection 4.6)– to achieve an objective (subsubsection 4.5.1) that contributes to the satisfaction of the given driver (subsubsection 4.5.2), by impacting specific parts of the given legacy system(subsection 4.3).*

Below we present six tables detailing the parts of a Legacy system affected by each proposed solution. The first three responses to the approaches (white box, black box, and grey box) on migrating solutions. The second triad responds to the three approaches in the context of adaptation solutions.

Migration solutions have been gathered and divided by approach in the following three tables. Black-box approaches in Table 5. We can see in this table that all the findings in this classification work over a specific concern and the architecture. Grey-box approaches are in Table 6. We can see in this table that most of the work is on how to enable architectures, such as SOA, cloud, etc. White-box approaches are in Table 7. We can see in this table that the heterogeneous, from paradigm to architectural migrations. The number of variables that are accessible from white-box approaches is much broader. Nevertheless, white-box approaches are more detailed, generally related to risk ideas, and time-consuming.

Adaptation solutions have been gathered and divided by approach in the following three tables. In Table 8 and Table 9, we find the different classifications on Adaptation proposals. Table 8 is our literature's only black-box adaptation approach. This approach bridges requests to some internal and well-known services. Finally, our last Table 9 holds the white-box approaches adaptation. The adaptation proposals are interesting since they tackle problematic software development assumptions, control, and hardware implications.

5.1 The taxonomy in action

Finally, to guide the reading of our selected articles, we offer Table 10 and Table 11, consisting of the classification of each article studied by the SLR.

6 Threats to validity

The base dataset of the study is both a strength and a weakness. We proposed open and significant research questions to capture the large sense of migration. It can threaten validity because many articles of importance may be missing just because they are too specific. Also, the lack of insight into software migration from other disciplines (such as finances, management, etc.) may redound in a theoretical framework that lacks bridges over those disciplines, which we consider essential in such large projects.

The article selection was done based on our understanding of what is and is not related, taking as input title, and sometimes title and abstract. This selection threatens the reproducibility of our experiment. To reduce the impact of this bias, we ran the screening of the articles many times during the writing process, including the last time at the end of the process.

Single researcher bias Despite the work we did on avoiding bias during the selection of the articles, from picking them to organising the reading and to having one reading before the process of open coding, the open codification done in the context of grounded theory has been conducted by a single researcher. This is known to be a threat to validity by the researcher's bias. Even knowing that all the authors participate in the confection of the paper, the systematic codification of the whole dataset is a time-consuming task that cannot be afforded by other than the primary author. The measures we took for reducing bias are: spacing the first lectures from the coding part and spacing the writing process from the coding. As well as digesting a large interleaving of phrases related to the axis of the paper before writing each part of the taxonomy, ensuring that for each part, all the articles have been properly re-overviewed and analysed about the ongoing taxonomy part.

7 Proposed Research

During our research, we found new or barely-explored ground.

Process risk assessment is recognised by most of the articles as one of the essential activities to succeed in such large projects. On material results of risk assessment, our best finding is that most of the papers describe the challenge of their process, which we can interpret as a risk. We found the neither systematic classification of risks, systematic measurements of risk, nor risk mitigation strategies.

Process implications We found evidence of implications on the studied processes, it seems to be a correlation between runtime migration and library migration: whenever there is a runtime migration, a library migration becomes compulsory. Also, there is a correlation between language migration and runtime migration. Having a clear view of the modernisation processes' implications can give an important hint on the measure of the size of a project. This information can be used for process risk assessment and planning and as a guide for reuse.

Product risk assessment whatever the flavour of the process is implemented, we end up with a product that must take over the requirements. This "new product" must respond to the current requirements in a specific form. We found only one work that considers the produced system during a modernisation process [48], ensuring that the produced quality responds to the expectation. We found no work on the acceptance of the product or the security risk of a hypothetical product of modernisation. This may be academic talk, but we get to use old code in new ways during migrations. These new ways were not part of the assumption of the development time. This can lead to large security breaches of multiple kinds, which we can easily foresee, from vulnerabilities and denial of service to data leakage.

Metrics and planning during the study, we find a direct relationship between decomposability and feasibility, primarily due to claims and not statistical analysis or measuring devices. The link between the system decomposability (by architecture and design), the modernisation approach, and the process may be the link required to recommend a specific kind of solution to a specific problem. It may also be a key to understanding the material requirements of a smooth, incremental modernisation process.

Validation and verification Most works propose, at best, an evaluation of tools over a single system, which is not enough to generalise or systematise. This may seem reasonable enough industrially, but this talk also about the general lack of modularity in the approaches and the lack of reusability. Validation and verification may also seem like academic words, but even systematic testability seems neglected in the literature.

Knowledge recapitalisation as an umbrella to discuss how to return project ownership to the operational teams. We acknowledge that other domains work on generating documentation or comments over running code (such as natural language processing), which could be handy in this context. But there is also a second part that seems to be neglected: all of these evolution processes are knowledge-intensive. We did not find any literature exploring how to leverage these processes to generate knowledge about the new product, like which requirements the new product will respond to or which were valid assumptions on the old system and not on the new system. There is a place in this context to recover documentation, generate ontological knowledge, etc.

8 Conclusion

During this work, we analyse the literature finding qualitative responses to our research questions. For responding "Which elements and concepts are involved in a migration process?" We offer a taxonomy that involves the process. For "What are the existing processes for software migration?" We investigate the Horseshoe and

Attached processes To understand “How are these processes incremental/iterative?” we summarise all the essential planning aspects to consider. Finally, for exposing “What validations/verifications are proposed?” we summarise the different approaches and what is required to use them.

We discover the lack of systematic bounds on the migration and modernisation literature. We discover the impact of this lack on the exchange of knowledge and research development due to the lack of unification. For tackling this problem, we decided to define a theory based on the existing work towards to unification of the subject and the development of a large vision of the field.

We recognise that reengineering works are issued over legacy systems to contribute to the satisfaction of expected drivers.

Much work is still needed to achieve a complete unification of the subject. We did the first step by defining a profile on the object of modernisation, a taxonomy in the context of software reengineering describing the kind of solutions, the reasons, the general approaches, the processes, and many of the available concrete techniques with their concrete material objectives. We studied the extracted insight on achieving the different planning features recognised by the literature as critical for achieving a successful process. We finally proposed five different paths for possible research.

Table 4 Found mappings between objectives and drivers

Direct Driver / Objective	Work	Dying technology	Arch. variable	Features	Ease the process of hiring	Competitive service	Enable businesses	Reduce costs
Migrate To Service	Service identification, code adaptation, wrapping and orchestration.[46]	NER	Accessibility	Web access	NER	NER	Online market	NER
	Lean and Mean Industrial approach [36]		Interoperability	API Access	NER	NER		NER
Enable Cloud	MDM approach for cloudify software [4]	NER	Elasticity & Scalability	NER	NER	Availability enhances service	NER	Pay-As-You-Go
Adapt UI to multiple devices	Mix multiple representations of one UI and serve it according to screen's size [15]	NER	Accessibility	Device aware UI Rendering	NER	NER	Portable devices market	NER
Adapt embedded system to support networking	Assess and modify from hardware to software to add network capabilities. [38]	NER	Accessibility	Network Access	NER	NER	NER	NER
Client-Server To Web	Interoperability middleware for Cobol application wrapping [12]	NER	Interoperability	Web access	NER	Offering services online	NER	NER
Paradigm Change	Object Model Discovery based on source code patterns [49] [48]	NER	Modularity & Interoperability	Web access	NER	Offering services online	NER	NER
Translation	C to Java by patterns and grammatical translation [31]	NER	Interoperability	Reusability — Web Access	NER	NER	NER	NER
	PL/IX to C++ by patterns and grammatical translation [27]	Language	Modularity & Stability	Web access	Ease code understanding	Offering services online	NER	NER
	Delphi to C# by general and specialized rules based transformation	Language	NER	NER	NER	NER	Fusion two companies' systems	NER
UI Translation	Model Driven Engineering: PSM to KDM. KDM Modified. KDM To Code. [17]	Language	NER	NER	NER	Offering services online	NER	NER
	Model Driven Engineering: PSM to KDM. KDM To Code. [41]	Framework	NER	NER	NER	NER	NER	NER
	Knowledge-based GUI selective translation [32]	Interface	NER	Use windows GUI API	NER	NER	NER	NER
	Procedural code interaction patterns recognition for building interaction model [33]	Interface	NER	GUI	NER	NER	NER	NER
	Java AWT to XIML conversion [15]	NER	Accessibility	Web access	NER	NER	NER	NER
Library migration	Ontological matching for code rewritting [47]	Operative System	NER	NER	NER	NER	Deploy on more devices	NER
Adapt batch to support interactive control	Lessons on converting a complex software (compiler) to support interaction [13]	NER	Interactivity	GUI Access	NER	NER	GUI tool market	NER

NER No Explicit Relationship in the articles considered

Table 5 Migration - black-box approach

Process	Objective	Solution	Data	GUI	Arch
Attached	Migrate Data Access Protocol	Database Gateway [10]	X		X
		XML Integration [10]	X		X
	Centralized to distributed database	Database replication [10]	X		X
	Migrate Text to GUI	Screen Scrapping [10]		X	X

Table 6 Migration - grey-box approach

Process	Objective	Solution	Data	GUI	Func	DS	Arch
Detached	Migrate To Service	Object-Oriented Wrapping [10]			X	X	X
		Component-Oriented Wrapping [10]			X	X	X
		Service identification, code adaptation, wrapping and orchestration.[46]				X	X
		Lean and Mean Industrial approach [36]			X	X	X
		Design patterns to reuse architecture [3]				X	X
		MASHUP [16]			X	X	X
	Enable Cloud	SMART [16]			X	X	X
		REMICS [16]			X	X	X
	Client-Server To Web	Interoperability middleware for Cobol application wrapping [12]		X		X	X
Attached	Migrate data management to RDBMS	Gateway Approaches, used to decouple the risk of data migration from the functional migration. Data access is interoperable through gateways with the system and target system [16]	X		X	X	

Arch Architecture DS Design
Func Functionality

Table 7 Migration - White-box

Process	Objective	Work	GUI	PD	Lg	3P	U-API	DS	Arch	MU	RT
Detached	Paradigm Change	Object Model Discovery based on source code patterns [49] [48]		X				X			
	Translation	C to Java by patterns and grammatical translation [31]		X	X	X					X
		PL/IX to C++ by patterns and grammatical translation [27]		X	X	X					X
		Delphi to C# by general and specialized rules based transformation			X	X	X			X	X
	UI Translation	Model Driven Engineering: PSM to KDM. KDM To Code. [41]	X		X			X			
		Knowledge-based GUI selective translation [32]	X		X						
		Model Driven Engineering: PSM to KDM. KDM Modified. KDM To Code. [17]	X		X			X			
		Procedural code interaction patterns recognition for building interaction model [33]	X		X						
		Java AWT to XIML conversion [15]	X	X							
	Library migration	Ontological matching for code rewritting [47]				X	X				X
	Enable Cloud	MDM approach for cloudify software [4]						X	X		
	KDM - PSM	KDM2PSM [2]						X			

PD Paradigm Lg Language 3P Third Party U-API Used API DS Design Arch Architecture MU Memory Usage RT Runtime

Table 8 Adaptation - Black-box

Process	Objective	Work	GUI	Arch
Attached	Enable web access	CGI Integration [10]	X	X

Table 9 Adaptation - White-box

Process	Objective	Work	GUI	Func	DS	Arch	MU	Ctrl	HW	RT
R. Attached	Adapt batch to support interactive control	Lessons on converting a compleX software (compiler) to support interaction [13]			X	X	X	X		
	Adapt UI to multiple devices	Mix multiple representations of one UI and serve it according to screen's size [15]	X		X	X				
	Adapt embedded system to support networking	Assess and modify from hardware to software to add network capabilities. [38]		X	X	X			X	X

Func Functionality DS Design Arch Architecture MU Memory Usage Ctrl Control flow HW Hardware RT Runtime

Table 10 Classified Articles - Part 1

Article	Legacy System	Main driver	Main Objective	Solution Kind	Approach Kind	Approach	Process
GUI Migration using MDE from GWT to Angular 6: An Industrial Case [41]	GWT Web application	Move from dying technology	UI Translation	Migration	White-box	Transformation	Detached
An Approach for Creating KDM2PSM Transformation Engines in ADM Context: The RUTE-K2J Case [2]	N/A	N/A	KDM to PSM transformation	Migration	White-box	Transformation	Detached
White-Box Modernisation of Legacy Applications [17]	Oracle forms application	Moving from dying technology	UI Translation	Migration	White-box	Transformation	Detached
A Survey on Survey of Migration of Legacy Systems [16] (Survey paper)	N/A	Many	Many	Migration	All	All	All
Modernisation of Legacy Systems: A Generalized Roadmap [24] (Meta paper)	N/A	Enable new architectural variables	Migrate To Service	Migration	All	All	All
How do professionals perceive legacy systems and software modernisation? [25]	Many	Many	N/A	N/A	N/A	N/A	N/A
A framework for architecture-driven migration of legacy systems to cloud-enabled software [1]	N/A	Enable new architectural variables	Enable Cloud	Migration	Grey-Box	Wrapping	Detached
Migrating Legacy Software to the Cloud with ARTIST [4]	N/A	Enable new architectural variables	Enable Cloud	Migration	Grey-Box	Wrapping	Detached
Seeking the ground truth: a retroactive study on the evolution and migration of software libraries [11] (Meta paper)	N/A	N/A	Library Migration	Migration	White-box	Transformation	N/A
Searching for model migration strategies [42]	Object Model	N/A	N/A	Adaptation	White-box	N/A	Detached
A lean and mean strategy for migration to services [36](Meta paper)	N/A	Enable new architectural variables	Migrate To Service	Migration	Grey-box	Wrapping	Detached
Extreme maintenance: Transforming Delphi into C# [6]	Delphi application	Move from dying technology	Translation	Migration	White-box	Transformation	Detached
Parallel iterative reengineering model of legacy systems [39] (Planification Paper)	N/A	N/A	N/A	All	N/A	N/A	N/A
Can design pattern detection be useful for legacy system migration towards SOA? [3]	Object oriented application	Enable new architectural variables	Migrate To Service	Migration	N/A	N/A	N/A
Developing legacy system migration methods and tools for technology transfer [12]	Cobol application	Enable New Business / Markets	Migrate to service	Migration	Grey-box	Wrapping	Detached
OPTIMA: An Ontology-Based PlaTform-specIfic software Migration Approach [47]	C/C++ application	Move from dying technology	Library Migration	Migration	White-box	Transformation	Detached
Reversing GUIs to XIML descriptions for the adaptation to heterogeneous devices [15]	Java AWT Application	Enable new features	GUI Migration — GUI Adaptation	Adaptation after Migration	White-box	Transformation	Detached

N/A Not applies All All the options of the taxonomy are to be found in this article Many More than one option of the taxonomy are to be found in this article

Table 11 Classified Articles - Part 2

Article	Legacy System	Main driver	Main Objective	Solution Kind	Approach Kind	Approach	Process
Quality driven software migration of procedural code to object-oriented design [48]	Procedural Application	Enable new features	Paradigm change	Migration	White-box	Transformation	Detached
Incubating services in legacy systems for architectural migration [46]	C/C++ Application	Enable new architectural variables	Migrate To Service	Migration	Grey-box	Wrapping	Detached
Network-centric migration of embedded control software: a case study [38]	Embedded system	Enable new features	Adapt embedded system to support networking	Adaptation	White-box	Transformation	Attached
C to Java migration experiences [31]	C application	Enable new features	Translation	Migration	White-box	Transformation	Detached
A framework for migrating procedural code to object-oriented platforms [49]	Procedural Application	Enable new features	Paradigm change	Migration	White-box	Transformation	Detached
A Survey of Legacy System Modernisation Approaches [10] (Survey paper)	N/A	Many	Many	All	Black-box	Wrapping	All
Code migration through transformations: an experience report [27]	PL/IX Application	Move from dying technology	Translation	Migration	White-box	Transformation	Detached
Lessons on converting batch systems to support interaction: experience report [13]	Batch application	Enable new features	Adapt batch to support interactive control	Adaptation	White-box	Transformation	Attached
Reverse engineering strategies for software migration (tutorial) [34] (Meta paper)	N/A	N/A	N/A	Migration	Black-box	Wrapping	N/A
Strategic directions in software engineering and programming languages [19] (Meta paper)	N/A	N/A	Paradigm change	Migration	N/A	N/A	N/A
Rule-based detection for reverse engineering user interfaces [33]	Texte UI Application	Enable new features	UI Translation	Migration	White-box	Transformation	Detached
Workshop on object-oriented legacy systems and software evolution [40] (Meta paper)	N/A	N/A	N/A	All	N/A	N/A	N/A
Knowledge-based user interface migration [32]	GUI Application	Move from dying technology	UI Translation	Migration	White-box	Transformation	Detached

References

1. Ahmad, A., Babar, M.A.: A framework for architecture-driven migration of legacy systems to cloud-enabled software. In: Proceedings of the WICSA 2014 Companion Volume, WICSA '14 Companion. Association for Computing Machinery, New York, NY, USA (2014). DOI 10.1145/2578128.2578232. URL https://doi.org/10.1145/2578128.2578232
2. Angulo, G., Martín, D.S., Santos, B., Ferrari, F.C., de Camargo, V.V.: An approach for creating kdm2psm transformation engines in adm context: The rute-k2j case. In: Proceedings of the VII Brazilian Symposium on Software Components, Architectures, and Reuse, SBCARS '18, pp. 92–101. Association for Computing Machinery, New York, NY, USA (2018). DOI 10.1145/3267183.3267193. URL https://doi.org/10.1145/3267183.3267193
3. Arcelli, F., Tosi, C., Zanoni, M.: Can design pattern detection be useful for legacy systemmigration towards soa? In: Proceedings of the 2nd International Workshop on Systems Development in SOA Environments, SDSOA '08, p. 63 to 68. Association for Computing Machinery, New York, NY, USA (2008). DOI 10.1145/1370916.1370932. URL https://doi.org/10.1145/1370916.1370932
4. Bergmayr, A., Bruneliere, H., Izquierdo, J.L.C., Gorronogoitia, J., Kousiouris, G., Kyriazis, D., Langer, P., Menychtas, A., Orue-Echevarria, L., Pezuela, C., et al.: Migrating legacy software to the cloud with artist. In: 2013 17th European Conference on Software Maintenance and Reengineering, pp. 465–468. IEEE (2013)
5. Bianchi, A., Caivano, D., Marengo, V., Visaggio, G.: Iterative reengineering of legacy systems. IEEE Transactions on Software Engineering **29**(3), 225–241 (2003)
6. Brant, J., Roberts, D., Plendl, B., Prince, J.: Extreme maintenance: Transforming Delphi into C#. In: ICSM'10 (2010)
7. Brodie, M.L., Stonebraker, M.: Migrating Legacy Systems. Morgan Kaufmann (1995)
8. Charmaz, K.: Constructing grounded theory. sage (2014)
9. Chikofsky, E., Cross II, J.: Reverse engineering and design recovery: A taxonomy. IEEE Software **7**(1), 13–17 (1990). DOI 10.1109/52.43044. URL http://dx.doi.org/10.1109/52.43044
10. Comella-Dorda, S., Wallnau, K., Seacord, R.C., Robert, J.: A survey of legacy system modernization approaches. Tech. rep., Carnegie-Mellon univ pittsburgh pa Software engineering inst (2000)
11. Cossette, B.E., Walker, R.J.: Seeking the ground truth: A retroactive study on the evolution and migration of software libraries. In: Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, FSE '12, pp. 55:1–55:11. ACM, New York, NY, USA (2012). DOI 10.1145/2393596.2393661. URL http://doi.acm.org/10.1145/2393596.2393661
12. De Lucia, A., Francese, R., Scanniello, G., Tortora, G.: Developing legacy system migration methods and tools for technology transfer. Software: Practice and Experience **38**(13), 1333–1364 (2008). DOI https://doi.org/10.1002/spe.870. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.870
13. DeLine, R., Zelesnik, G., Shaw, M.: Lessons on converting batch systems to support interaction: Experience report. In: Proceedings of the 19th International Conference on Software Engineering, ICSE '97, p. 195 to 204. Association for Computing Machinery, New York, NY, USA (1997). DOI 10.1145/253228.253267. URL https://doi.org/10.1145/253228.253267
14. Demeyer, S., Ducasse, S., Nierstrasz, O.: Object-Oriented Reengineering Patterns. Morgan Kaufmann (2002)
15. Di Santo, G., Zimeo, E.: Reversing guis to ximl descriptions for the adaptation to heterogeneous devices. In: Proceedings of the 2007 ACM Symposium on Applied Computing, SAC '07, p. 1456 to 1460. Association for Computing Machinery, New York, NY, USA (2007). DOI 10.1145/1244002.1244314. URL https://doi.org/10.1145/1244002.1244314
16. Ganesan, A.S., Chithralekha, T.: A survey on survey of migration of legacy systems. In: Proceedings of the International Conference on Informatics and Analytics, ICIA-16. Association for Computing Machinery, New York, NY, USA (2016). DOI 10.1145/2980258.2980409. URL https://doi.org/10.1145/2980258.2980409
17. Garcés, K., Casallas, R., Álvarez, C., Sandoval, E., Salamanca, A., Viera, F., Melo, F., Soto, J.M.: White-box modernization of legacy applications: The oracle forms case study. Computer Standards & Interfaces pp. 110–122 (2017). DOI https://doi.org/10.1016/j.csi.2017.10.004
18. Goedicke, M., Zdun, U.: Piecemeal legacy migrating with an architectural pattern language: A case study. Journal of Software Maintenance and Evolution: Research and Practice **14**(1), 1–30 (2002)
19. Gunter, C., Mitchell, J., Notkin, D.: Strategic directions in software engineering and programming languages. ACM Comput. Surv. **28**(4), 727 to 737 (1996). DOI 10.1145/242223.242283. URL https://doi.org/10.1145/242223.242283
20. ISO: International Standard – ISO/IEC 14764 IEEE Std 14764-2006. Tech. rep., ISO (2006)
21. ISO: International Standard – ISO/IEC 25010:2011 – Software engineering – Product quality. Tech. rep., ISO (2011)
22. ISO: Iso/iec/ieee systems and software engineering – architecture description. ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000) pp. 1–46 (2011). DOI 10.1109/IEEESTD.2011.6129467
23. ISO: International Standard – ISO/ICE 90003:2018 – Software engineering – Product quality. Tech. rep., ISO (2015)
24. Jain, S., Chana, I.: Modernization of legacy systems: A generalised roadmap. In: Proceedings of the Sixth International Conference on Computer and Communication Technology 2015, ICCCT '15, p. 62 to 67. Association for Computing Machinery, New York, NY, USA (2015). DOI 10.1145/2818567.2818579. URL https://doi.org/10.1145/2818567.2818579
25. Khadka, R., Batlajery, B.V., Saeidi, A.M., Jansen, S., Hage, J.: How do professionals perceive legacy systems and software modernization? In: Proceedings of the 36th International Conference on Software Engineering, pp. 36–47 (2014)
26. Kitchenham, B., Charters, S.: Guidelines for performing systematic literature reviews in software engineering. Tech. rep., Department of Computer Science University of Durham (2007)
27. Kontogiannis, K., Martin, J., Wong, K., Gregory, R., Müller, H., Mylopoulos, J.: Code migration through transformations: An experience report. In: Proceedings of the 1998 Conference of the Centre for Advanced Stud-

ies on Collaborative Research, CASCON '98, p. 13. IBM Press (1998)
28. Krüger, J., Lausberger, C., von Nostitz-Wallwitz, I., Saake, G., Leich, T.: Search. review. repeat? an empirical study of threats to replicating slr searches. Empirical Software Engineering **25**(1), 627–677 (2020)
29. Larman, C., Basili, V.R.: Iterative and incremental developments. a brief history. Computer **36**(6), 47–56 (2003). DOI 10.1109/MC.2003.1204375
30. Lehman, M., Belady, L.: Program Evolution: Processes of Software Change. London Academic Press, London (1985)
31. Martin, J., Muller, H.A.: C to java migration experiences. In: Proceedings of the Sixth European Conference on Software Maintenance and Reengineering, pp. 143–153. IEEE (2002)
32. Moore, Rugaber, Seaver: Knowledge-based user interface migration. In: Proceedings 1994 International Conference on Software Maintenance, pp. 72–79. IEEE Comput. Soc. Press (1994). DOI 10.1109/ICSM.1994.336788. URL `http://ieeexplore.ieee.org/document/336788/`
33. Moore, M.M.: Rule-based detection for reverse engineering user interfaces. In: Proceedings of WCRE'96: 4rd Working Conference on Reverse Engineering, pp. 42–48. IEEE (1996)
34. Müller, H.A.: Reverse engineering strategies for software migration (tutorial). In: Proceedings of the 19th International Conference on Software Engineering, ICSE '97, p. 659 to 660. Association for Computing Machinery, New York, NY, USA (1997). DOI 10.1145/253228.253799. URL `https://doi.org/10.1145/253228.253799`
35. Petticrew, M., Roberts, H.: Systematic reviews in the social sciences: A practical guide. John Wiley & Sons (2008)
36. Razavian, M., Lago, P.: A lean and mean strategy for migration to services. In: Proceedings of the WICSA/ECSA 2012 Companion Volume, WICSA/ECSA '12, p. 61 to 68. Association for Computing Machinery, New York, NY, USA (2012). DOI 10.1145/2361999.2362009. URL `https://doi.org/10.1145/2361999.2362009`
37. Shull, F., Singer, J., Sjøberg, D.I.: Guide to advanced empirical software engineering. Springer (2007)
38. de Souza, P., McNair, A., Jahnke, J.H.: Network-centric migration of embedded control software: a case study. In: Proceedings of the 2003 conference of the Centre for Advanced Studies on Collaborative research, pp. 54–65 (2003)
39. Su, X., Yang, X., Li, J., Wu, D.: Parallel iterative reengineering model of legacy systems. In: 2009 IEEE International Conference on Systems, Man and Cybernetics, pp. 4054–4058. IEEE (2009)
40. Taivalsaari, A., Trauter, R., Casais, E.: Workshop on object-oriented legacy systems and software evolution. SIGPLAN OOPS Mess. **6**(4), 180 to 185 (1995). DOI 10.1145/260111.260276. URL `https://doi.org/10.1145/260111.260276`
41. Verhaeghe, B., Etien, A., Anquetil, N., Seriai, A., Deruelle, L., Ducasse, S., Derras, M.: GUI migration using MDE from GWT to Angular 6: An industrial case. In: 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER'19), pp. 579–583. Hangzhou, China (2019). DOI 10.1109/SANER.2019.8667989
42. Williams, J.R., Paige, R.F., Polack, F.A.C.: Searching for model migration strategies. In: Proceedings of the 6th International Workshop on Models and Evolution, ME '12, p. 39 to 44. Association for Computing Machinery, New York, NY, USA (2012). DOI 10.1145/2523599.2523607. URL `https://doi.org/10.1145/2523599.2523607`
43. Wolfswinkel, J.F., Furtmueller, E., Wilderom, C.P.: Using grounded theory as a method for rigorously reviewing literature. European journal of information systems **22**(1), 45–55 (2013)
44. Wu, B., Lawless, D., Bisbal, J., Richardson, R., Grimson, J., Wade, V., O'Sullivan, D.: The butterfly methodology: A gateway-free approach for migrating legacy information systems. In: Proceedings. Third IEEE International Conference on Engineering of Complex Computer Systems (Cat. No. 97TB100168), pp. 200–205. IEEE (1997)
45. Zabardast, E., Gonzalez-Huerta, J., Gorschek, T., Šmite, D., Alégroth, E., Fagerholm, F.: Asset management taxonomy: A roadmap. arXiv preprint arXiv:2102.09884 (2021)
46. Zhang, Z., Yang, H.: Incubating services in legacy systems for architectural migration. In: 11th Asia-Pacific Software Engineering Conference, p. 196 to 203. IEEE (2004)
47. Zhou, H., Kang, J., Chen, F., Yang, H.: Optima: an ontology-based platform-specific software migration approach. In: Seventh International Conference on Quality Software (QSIC 2007), pp. 143–152. IEEE (2007)
48. Zou, Y.: Quality driven software migration of procedural code to object-oriented design. In: 21st IEEE International Conference on Software Maintenance (ICSM'05), pp. 709–713. IEEE (2005)
49. Zou, Y., Kontogiannis, K.: A framework for migrating procedural code to object-oriented platforms. In: Proceedings Eighth Asia-Pacific Software Engineering Conference, p. 390 to 399. IEEE (2001)