Software Migration: A Theoretical Framework (A Grounded Theory approach on Systematic Literature Review)

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

Noname manuscript No.
(will be inserted by the editor)

Software Migration: A Theoretical Framework

A Grounded Theory approach on Systematic Literature Review

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

Received: date / Accepted: date

Abstract Software migration has been a research subject for a long time. Major research and industrial implementations were conducted, shaping not only the techniques available nowadays, but also a good part of Software evolution jargon. To understand systematically the literature and grasp the major concepts is challenging and time-consuming. Even more, research evolves, and it does based on the assumption that many words (such as migration) have a single well-known meaning that we all share. Since since these words meanings are rarely explicit, and their usage heterogeneous, these words end up polluted with multiple and many times opposite or incompatible meanings. In our quest to understand, share and contribute in this domain, we recognize this situation

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

as a problem. To tackle down this problem we propose a taxonomy on the subject as a theoretical framework grounded on a systematic literature review. In this study we contribute a bottom-up taxonomy that links from the object of a migration to the procedure nature migration, passing by migration drivers, objectives and approaches. We contribute a classification of all our readings, and a list of research directions discovered on the process of this study.

Keywords Software Reengineering · Migration · Modernization · Taxonomy.

1 Introduction

Software migration happens. With the fast innovation pace of the software industry, it happens more and more often. The research and industrial implementations of software migration evolves not only the software but also, the natural language we use to understand and communicate the knowledge required for conducting such processes. The wide and heterogeneous cases of migration, as well as the specificity of most of the approaches, threats the reusability of the existing knowledge, by polluting our language with multiple and/or incompatible definitions. "Legacy system" is a name used to refer widely different systems from different times [9, 1] as if these systems require exactly the same solutions. Even what we do understand by migration is unclear, when [9] points wrapping to surely not be a migration approach, and [12] cites many wrapping based migrations.

The urgency that often characterizes the migration projects seems to not allow the software engineers to go thought this wide and scattered literature looking for guides. This reality facilitates the production of a broad, scattered and hard to systematize literature, impacting on 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 recognize this situation as a problem

To tackle down this problem we propose a bottom-up taxonomy on the subject as a theoretical framework grounded on a Systematic Literature Review (SLR).

Taking into account that a software migration is a kind of software engineering project, we expect it to respond to similar cycle and problematic. Software engineering projects are required to produce results that respond to requirements and acceptance. These projects are also susceptible to risks and failure. Many works claim and demonstrate that iterative and incremental planning and implementation approaches are the key to mitigate these risks and to succeed such enterprises [23].

Such theoretical framework is based on a study driven by the following questions: Which elements and concepts are involved in a migration process? What are the existing processes for software migration? How are these processes incremental/iterative? What validations/verifications are proposed?

These questions enable us to contribute a taxonomy that covers the various concepts that characterize a migration: 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 recognized as key in software engineering: iterativity, incrementality and validity.

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

Following, we do explain 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 (subsection 4.7). We identify the threats to the validity of our study (section 5) and contribute a list of research directions on areas that we find to be yet unexplored (section 6). The article finishes with a conclusion on the study (section 7).

2 Systematic Literature Review: protocol definition

This section details the protocol followed for conducting the experiment.

2.1 Planning

The first phase of the protocol aims to cover three main aspects of the SLR: (I) to explain why it is important to conduct an SLR, by stating the research questions expected to respond with the study. (II) to expose the considerations of the construction of the search string used for gathering the relevant articles. (III) to consider the main aspects of the validation of the results.

2.1.1 Motivations

As stated in section 1, our motivation for conducting this SLR is to build a theoretical framework able to articulate and unify the different approaches proposed by the selected articles. This SLR aims to characterize the different elements of a migration, summarize and synthesize the different migration approaches, emphasizing on the process's characteristics, how the technical approaches allow incremental and iterative processes, and how are them validated or verified.

2.1.2 SLR 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 project of software migration that involves different kinds of migration: GUI Migration (Desktop to Web), Architectural migration (Monolithic to Microservice), and Language Migration. The intent of our study is to discover the different approaches, to elucidate the risks how to mitigate these risks, and to understand if the software migration processes respond to iterativity and incrementality as software engineering processes.

Research Context Following the method proposed by [21], we define the context of our research questions, to relate the different research questions and to relate the further decisions taken during the study. Our research questions arise from a more general question that is *What would be a valid theoretical framework that relates and gives meaning to the techniques, technologies and concepts that are required to achieve a migration process successfully, and that can systematically guide our research and reading of the large literature, driven specially by the implementation process features?*

Research questions definition The research questions and their contribution are listed in the Table 1, and explained below. Our goal is to apply qualitative analysis over the article selection, and to refine the qualitative study a theoretical framework, we propose four open qualitative research questions. Since we aim our study to be done from a "process" point of view, the research questions reinforce the direction of the study towards the process nature of a migration, and on how to achieve incrementality, iterativity and verifiability. Our first question RQ1 limits the study to software migration as a process. RQ2 denotes the importance of the identification of the different elements and their role in a migration. RQ3 bias our study towards the usage of incremental and iterative planning and implementation of such processes. This bias is due to our knowledge of different works claiming that iterativity and incrementality are key features to succeed in large and complex software engineering projects. Finally, question RQ4 biases the study towards the verifiability of the proposed solutions. This bias is due to narrow the study to those solutions that propose some sort of guarantee.

2.2 Search Query

Following the method proposed by [21], we build a keyword-based query, to gather of articles, based on the following steps:

(i)Obtain keywords from the context the research questions. (ii)Obtain keywords synonyms, to be able to widen the search. (iii) Build the search string using PICOC (Population, Intervention, Comparison, Outcomes, Context) [28]

RQ#	Question	Aim
RQ1	Which elements and concepts are involved in a migration process??	Link migration with the artefacts involved
RQ2	What are the existing processes for software migration??	Comprehend the procedural nature of Migration
RQ3	How are these processes incremental/iterative?	Link processes with planning
RQ4	What validations/verifications are proposed?	Link processes with guarantees

Table 1 SLR Research Questions

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

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

Contextualizing The PICOC technique, proposed by [28], aims to contextualize the query building based on the understanding of the elements of our study. This technique is essentially used in SLR in social sciences. Applied also on software engineering studies such as [30]. We followed his general mapping criteria for our points.

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

Intervention: How? The intervention or procedure 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 done 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 taxonomy able to classify the approaches proposed the analysed articles, including the approaches analysed by the surveys found during the SLR.

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

Source name	URL	Results
ACM Digital Library	http://dl.acm.org	150
IEEE Xplore	https://ieeexplore.ieee.org	8
IET Digital Library	https://digital-library.theiet.org	40
Springer	https://link.springer.com	580
Wiley Online Library	https://onlinelibrary.wiley.com	213
Science Direct	https://www.sciencedirect.com/	1
Total		992

Table 2 Search engines

Search string tuning For ensuring the relevance of the query we iterated by adding, removing or splitting keywords and synonyms and tested the query in google scholar.

The general parametrization of Google Scholar search for the test are:

- Date test: 29/10/2020
- Testing environment: Google Scholar [1]
- Time span: 2000-2020
- Excludes: cites and patents

The search string was tested and tuned up to obtain a minimal expected relevance. The title and abstract of each of the first 100 results of each test is screened and summarized. The query was considered tuned once we reached 76 relevant results out these 100 results. This proportion of relevancy has been accepted by other articles such as [30].

The final search string obtained by this process is the following:

("migration" OR "modernization") AND ("reengineering" OR "transliteration") AND ("software") AND ("iterative" OR "incremental") AND ("validation" OR "analysis" OR "verification" OR "solution")

2.3 Conducting the protocol: Articles Selection

After the tuning of the search string, we proceeded to search for articles on the search engines of the most popular article editors in the domain.

Table 2 lists the engines, their URL and the amount of articles matching the search string. These values correspond to the queries done the 29/10/2020.

2.4 Articles selection process

For selecting the articles, we firstly searched for repetitions. Not finding any, we moved forward to do a quick screening. The screening was based on the reading of titles and abstracts. At this point we took all articles related with

[1] http://scholar.google.com

software processes. This left us with 71 articles. From these 71 articles, we removed those that were grey literature (books, reports, etc) and those out of domain (finances by example), leaving 57 articles. From these 57 articles, we read firstly two general surveys [12,7] and a paper on the professional perception of software modernization [20]. All these three articles are meta articles. The first two surveys expose different software migration solutions. The third one exposes the industrial perception of what a software migration is about and what it is expected to achieve, which aligns with our industrial software migration context. From the 57 articles, we removed those that not seem to be directly by reading abstract introduction and conclusion, reducing the dataset to 27 articles. After the first phase of reading of these 27 articles, we run again the selection over the 57 articles, retrieving 3 articles, giving a total of 30 articles. After the application of the analysis methodology, we run again the screening over the 57 articles retrieving 0 articles. After the writing of the main taxonomy, we run again the screening over the 57 articles retrieving 0 articles.

We aim to produce a bottom-up taxonomy, and link it with more general and standard concepts. For achieving this, during the confection of the taxonomy we relied on support literature. We choose ISO IEC Software Standards, due to the international acceptance and the citation of it by some of our articles [1].

We rely on the documents ISO IEC 25010 [16], 42010 [17] 14764[15], 90003 [18] for those definitions related with quality, process and architecture. Widely used terms, but never explicitly defined.

The Table 3 includes the 30 articles obtained by the search string and fully included in this SLR. At the end of the table we find those articles added as support literature.

3 Conducting Protocol: Grounded Theory

To produce this bottom-up taxonomy grounded on the literature, inspired by [31,20], we decided to apply the grounded theory method over a systematic literature review, to be able to manifest what is explicitly and implicitly understood.

As we stated in section 2, our study aims to build a taxonomy based on SLR. We used the research questions to narrow down the articles to study. We use qualitative research to discover an emerging bottom-up taxonomy. For conducting this qualitative research, we choose to follow a Grounded Theory (GT) approach. GT is an exploratory research method that aims at discovering new perspectives and insights, rather than confirming existing ones [6] In order to have an open mind, reducing bias and let the knowledge emerge from the text, rather than find responses to strict pre-existing questions (which implies a bias on how to read and interpret content), we adopted a qualitative research strategy. The main two concepts used in our study are open coding and axial coding. The open coding process consists in breaking down the content into

#	Year	Title	Publisher
1	2019	GUI Migration using MDE from GWT to Angular 6: An Industrial Case [35]	IEEE
2	2018	An Approach for Creating KDM2PSM Transformation Engines in ADM Context: The RUTE-K2J Case [2]	ACM
3	2017	White-Box Modernization of Legacy Applications [13]	Springer
4	2016	A Survey on Survey of Migration of Legacy Systems [12]	ACM
5	2015	Modernization of Legacy Systems: A Generalized Roadmap [19]	ACM
6	2014	How do professionals perceive legacy systems and software modernization? [20]	
7	2014	A framework for architecture-driven migration of legacy systems to cloud-enabled software [1]	
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 [8]	
10	2012	Searching for model migration strategies [36]	ACM
11	2012	A lean and mean strategy for migration to services [29]	ACM
12	2010	Extreme maintenance: Transforming Delphi into C# [5]	IEEE
13	2009	Parallel iterative reengineering model of legacy systems [33]	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 [9]	Wiley & Sons
16	2007	OPTIMA: An Ontology-Based PlaTform-specIfic software Migration Approach [39]	IEEE
17	2007	Reversing GUIs to XIML descriptions for the adaptation to heterogeneous devices [11]	ACM
18	2005	Quality driven software migration of procedural code to object-oriented design [40]	IEEE
19	2004	Incubating services in legacy systems for architectural migration [38]	IEEE
20	2003	Network-centric migration of embedded control software: a case study [32]	IBM Press
21	2002	C to Java migration experiences [24]	IEEE
22	2002	A framework for migrating procedural code to object-oriented platforms [41]	IEEE
23	2000	A Survey of Legacy System Modernization Approaches [7]	DTIC [2]
24	1998	Code migration through transformations: an experience report [22]	IBM Press
25	1997	Lessons on converting batch systems to support interaction: experience report [10]	ACM
26	1997	Reverse engineering strategies for software migration (tutorial) [27]	ACM
27	1996	Strategic directions in software engineering and programming languages [14]	
28	1996	Rule-based detection for reverse engineering user interfaces [26]	IEEE
29	1995	Workshop on object-oriented legacy systems and software evolution [34]	ACM
30	1994	Knowledge-based user interface migration [25]	IEEE
–	2015	ISO IEC 90003 (ISO 9001 applied to Software) [18]	ISO
–	2011	ISO IEC 25010 (ex ISO IEC 9126)[16]	ISO
–	2011	ISO IEC 42010 [17]	ISO
–	2006	ISO IEC 14764 [15]	ISO

Table 3 Initial Dataset

different parts and labelling them with words or short phrases, with the goal of content discretisation. Axial coding consists of categorizing the found open codes.

Each of the articles has been read systematically two times in two phases. The first phase in the lapse of two weeks, taking overview notes of each reading. The second phase read has been assisted by the usage of qualitative research software MAXQDA2020[3]. Notes taken in the first phase are mean to be dismissed but expected to help to contextualize the researcher. The notes are available in the folder **articles** in the following GIT repository https://gitlab.inria.fr/sbragagn/slrmigration/.

During the second reading of each article, we applied open coding methodology at sentence / paragraph levels. The sort of codifications at the level of a document are by example *"migration: multiple actor problem"*, *"migration is related with decomposability"*, *"a legacy system may have not external information (doc, manual), or obsolete"*, etc.

After the reading of each article we incrementally reorganized the open coding codes into simple axial coding 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.

When this process is finished, we end up having 756 different codes, organized on a hierarchical but vague axial coding. The complete list of codes is available as **Coded Segments.html** in the following GIT repository https://gitlab.inria.fr/sbragagn/slrmigration/.

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 taxonomy axes. All this content is available in the appendix.pdf file in the following GIT repository https://gitlab.inria.fr/sbragagn/slrmigration/, and submitted in the HAL platform https://hal.inria.fr/hal-03169377.

4 A literature emergent bottom-up taxonomy

As explained by [37] taxonomies main utility is to communicate knowledge, provide a common vocabulary, and help 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 that are created using the top-down method use the existing knowledge structures and categories with established definitions. In contrast, the taxonomies that use the bottom-up approach are created using the available data such as experts' knowledge and literature. Since we did not find established definitions and taxonomies on the subject, we propose a bottom-up taxonomy, based on the analysis and synthesis of the selected literature. The crafting of the taxonomy responds to our first research question: Which elements and concepts are involved in a migration process?

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

Following we make explicit some basic definitions required to contextualize the taxonomy, to follow up defining the taxonomy, After we define characterize and define a taxonomy

4.1 Software System definitions

A migration is always applied to some level of an origin system. This software system is mostly named "Legacy system".

Software systems may have internal subsystems, and be contained by larger systems.

System Following the definition given by [17] 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 add also that all these entities and their relationships configure what we understand as the environment where our software takes place.

Software functional entity built from source code, able to produce a desired behaviour by interacting with other entities on the system. A software may respond to one or more concerns, such as User Interface, Data Storage, Intercommunication, or plain Functionality (calculations, predictions, etc).

Dependencies All artefacts required to be part of a system for a given software to be fully functional. E.g., libraries, frameworks, services, hardware.

Application Programming/Binary Interface While an API is usually a source code interface that an operating system, library, or service provides to support requests made by computer programs, an ABI defines how data structures or computational routines are accessed in machine code, which is a low-level, hardware-dependent format. Both of them can be considered as an architectural connector since those are the protocols to define and respect to enable interoperability.

Architecture & Design Following the definition given by [17], we recognize architecture to be the fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the 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 following the [17] comment on: "The architecture of a system is cognisant of the system in its environment; the environment determines the totality of influences on the system. One often-cited difference between architecture and design is this: **architecture is outwardly focused** on the system in its environment; whereas **design is inwardly focused** once the system boundaries are set".

Source code is the building material of the pieces of software in general. Source code is written in a programming language and it follows one or more paradigms that provides conceptual means to define functionalities, normally provided by the programming language. The source code responds to a design that organizes the internal concepts and allows the articulation of the produced software with the system through some exhibited API, and depends on other entities by using those entities API or ABI.

Design Patterns formalized best practices on the scope of a specific programming and architectural paradigm, that the programmer can use to solve common problems when designing an application or system. These patterns normally describe resilient and/or stable internal compositions of source code with a rather specific goal.

Paradigm understood as a set of conceptual tools provided by a programming language for writing the source code of a program. Thus defining the way in which the programmer conceives and perceives the program itself, affecting on which are the development assumptions and how the required semantics are expressed and mapped.

System Documentation All different kinds of documents that trace and support the implementation and evolution of a software and its usage, such as user and developer manuals, requirements reports, processes specifications, etc.

Software Quality According to [16] we talk about quality from three points of view. The quality is perceived internally by measuring the quality of source code and or architectural metrics, such as cohesion and coupling, test coverage or by the complementary support they may have, such as user documentation or architectural / development documentation, and the existence of knowledge on the maintaining organization. The quality is perceived externally by measuring its artefact behaviour. Finally, the quality is perceived in-use as the capacity of the software to accomplished requirements, to adapt to new changes. [16] also spots the inter-relationship of these qualities, making explicit that internal quality impacts on external quality, which impacts on quality in-use. E.g., [40] spots how the internal quality is important to enable new features, required to enable web technologies.

Software Modernity The modernity of a software is related with the distance in between the up-to-date techniques and technologies of software development, and those used during the development of the source code. An example would be if this software is or not able to profit from the usage of up to date technologies and concepts by example: IOT, Blockchain, microservices. E.g., [4] proposes to enable cloud computing on existing systems, or [26] who brings GUI to a text-based UI application.

Software Continuity The continuity of a piece of software (also persistence or permanence) is directly related to the resource allocation policy for its maintenance and evolution. Despite the modernity or the quality, a software continuity is related with how much this software is needed, and how many resources are the owners ready to afford for keeping it working. A direct implication of continuity is the increment of the investment value in multiple aspects: money, time and knowledge.

In an industrial context, systems that arrive to the decision of migration are relevant, and they are relevant due to their long continuity. E.g., [9] spots the importance of systems that runs 24/7. Also, [22] points that software that migrates "are often mission critical for the organization that owns and operates them".

4.1.1 Legacy System: A problematic permanent system

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

By decadence , we understand the continuous deterioration of the **internal inherent 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**. [22] states a really important fact on this aspect: "Some components of the system 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 affect negatively its overall performance.".

By obsolescence , we understand the changes of the environment where our software exists and how these changes affect the **external inherent 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, apparition of radically cheaper infrastructure, the deprecation of dependent software (libraries, compilers, etc), the out-of-production of required hardware platforms, changes on business legislations, etc. The obsolescence of the system **justifies and causes its evolution**. [32] exposes the urgency of system evolution in the context of a project that requires enabling network communication on a system that include embedded software, since this requirement implies hardware level modifications.

Legacy systems are normally systems that exhibit some grade of decadence and/or obsolescence at some part of the system. We find that the nomenclature Legacy System is too vague and not really revealing. As vague as proposed by

one of the interviews in [20], "My definition of a legacy system is systems and technologies that do not belong to your strategic technology goals".

Therefore, we propose to specify the kind of legacy system in terms of how are them affected by decadence and/or obsolescence. Since we defined decadence and obsolescence to affect correspondingly to internal or external parts qualities of the system, we propose a non exhaustive list of source-code centric internal and external parts of a system.

By external parts we refer to all the material and intellectual elements that may affect and or constraint the impacted source code. Internal parts we refer to the different aspects of the crafting quality that may affect and or constraint 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

We can then talk about (i) legacy system due to a 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.

4.2 Solution kinds

Analysing the reporting we split what is and what is not a migration, and what different kind of migrations emerge from different system parts, and which are their implications. Most of the times in complex problems we cannot easy match the outcome of one tool with the desire future of a piece of software. In all the cases, the solutions have specific objectives (we address objectives on subsubsection 4.3.1), and conducted to respond entirely or partially to one or more solution drivers (we address drivers on subsubsection 4.3.2).

We propose two large families of solutions first that include all possible solutions, in relation to the whole system. **Reengineering & Replacement**:

Figure 1 gives a general over view on the Solution's taxonomy. In grey, we find those nodes that are not further explored in this article. Those nodes are

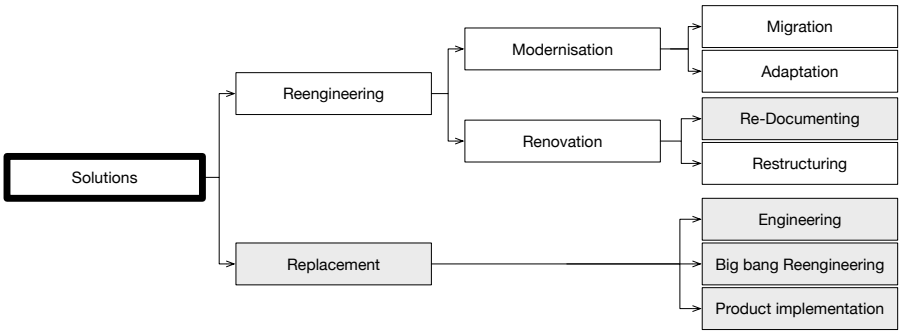

Fig. 1 Solution's Taxonomy Overview (In grey we find those nodes that are not further explored in this article).

not explored because the selected literature does not provide experience on this family, beyond acknowledging its existence. Nevertheless, their inclusion and definition is maintained to insist on what is not a migration.

Reengineering Is all process based on the modification of a previously existing system.

Modernization All processes that recover a system from **Obsolescence**, achieving a better integration with the environment and enhancing the external quality of our system. These processes affect external and internal elements of a Legacy System. *Adaptation* is all Modernization process that enables the usage of a new environment, without threatening the original environment. There are many kinds of adaptations, from e.g., (i) [14], proposing to compile C in C++, to be able to add new code on object-oriented fashion, to e.g., (ii) [32] proposing to modify hardware, or e.g., [11] who adapts a website to be rendered on different running devices. *Migration* is all Modernization process that moves from one environment to a target environment that is in relation of mutual exclusion (either for technological or strategical reasons) with the origin environment. There are many kinds of migrations, like source code translation proposed by [5, 22, 24], GUI migrations proposed by [35, 13, 25], or library migration [39, 8, 24]

Renovation We understand by Renovation all processes that recover a system from **Decadence**, achieving a 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 process issued over the source code (e.g., refactoring). *Re-Documenting* is all Renovation process that produces new or enhance existing documentations of the code such as writing manuals, specifying processes, formalizing requirements. "The spectrum of reengineering activities includes re-documentation, restructuring of source code, transformation of source code, abstraction recovery, and reimplementation." [27]

Replacement All processes that discard the existing system and establish a different one. *Engineering* is all Replacement process that creates a new system based on the understanding of 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 [5] *Product implementation* is all Replacement processes that implement and customize a Commercial off-the-shelf (COTS) system to solve the current requirements. E.g., [32] proposes as possibility an off-the-shelf product.

We can then talk about (i) legacy system due to a 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.

4.3 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 that is swung to deliver an impact to a small area of an object. Different kind 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 the usage of a hammer often relates to larger processes with more general targets: build a shelf, forge a sword, etc.

4.3.1 Objectives

We understand as objective the expected specific outcome of the application of a solution. In our SLR we found the following objectives:

Migrate Data Access Protocol : Modify the data accessing architecture.
Centralized to distributed database : Distribute and/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 organization and semantics from procedural to object oriented programming.
Translation : Translate source code from one language to another one.
UI Translation : Translate the UI representation from one model to another one.

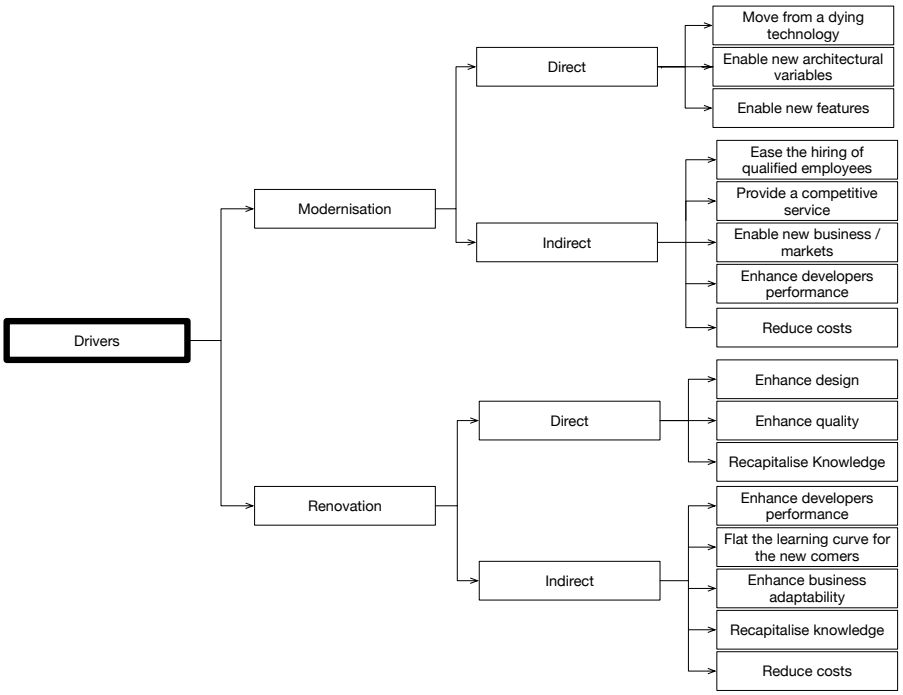

Fig. 2 Driver's Taxonomy Overview

Library Migration : Change the API used to delegate a concern to a given library/framwork.

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 embedded system to support networking : Implement network communication between devices.

Adapt batch to support interactive control : Adapt batch to support interactive control

4.3.2 Drivers

Overview Figure 2 gives a general overview on the Driver's taxonomy.

Reengineering processes are often expensive in time and money. The expected outcome is often a system that responds to exactly the same problematic, but differently. Large spending of resources for a system that does not solve new problems are often left for critical situations, where 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 subsubsection 4.1.1).

Our bottom-up taxonomy groups the findings on drivers into the groups of **Direct & Indirect** in the context of **Modernization & Renovation**. We focus then on the **Evolutionary** processes of **Modernization & 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 does not provide any experience or hard evidence on this family, beyond acknowledging its existence.

Direct drivers We understand by Direct drivers, all those decisions that find their reasons in the **immediate impact** of the application of a specific solution. Most of the drivers in this branch respond to strategic technological and/or system's quality objectives.

Indirect drivers We understand by indirect drivers, all those decisions that find their reasons in the **expected implications** of the impact of the application of a specific solution. Most of the drivers in this branch respond to strategic organizational objectives.

4.3.3 Modernization related drivers

- Direct
  - Move from a dying technology [35,8]
  - Enable new architectural variables (scalability, elasticity, availability) [1,4,19]
  - Enable new features (interactivity, run on new devices) [11] [24]
- Indirect
  - Ease the process of hiring qualified employees [34]
  - Provide a competitive service [19,1,4]
  - Enable new businesses / markets [11,38]
  - Enhance developers' performance [22]
  - Reduce costs [22]

4.3.4 Renovation related drivers

- Direct
  - Enhance architectural variables by design (scalability, elasticity, availability) [1,4,19]
  - Enhance design quality variables (decomposability, maintainability, understanding, reliability) [14,11,32]
  - Recapitalize knowledge [9]
- Indirect
  - Enhance developers' performance [27]
  - Flat the learning curve for newcomers [34]

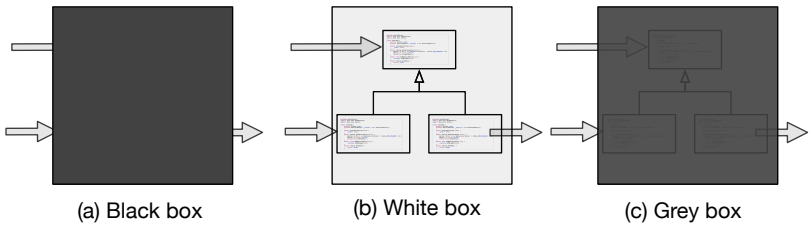

Fig. 3 Approaches

- Enhance business adaptability [26]
- Recapitalize knowledge[9]
- Reduce costs[9]

We can then talk about (i) Legacy system due to a third-party library obsolescence, requiring modernization to move out from a dying technology. (ii) Legacy system due to an obsolete architectural paradigm, requiring modernization because of the low availability of experts for hiring. (iii) Legacy system due to decadent source code, requiring renovation to run on new devices. (iv) Legacy system due to decadent design, requiring renovation to enhance the maintainability.

4.3.5 Objectives & Drivers mapping: Contribution

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 5 shows the Cartesian product between those objectives that have been mapped to the drivers by the literature. Please note that Table 5 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 on the proposed methods, those drivers are not included in the table. The table includes the acronym NER that stands for Not Explicit Relationship. This means that the work did not provide explicit link between solution and specific driver. In the other cases, the crossing points give us the Contribution of solution's objective to the driver.

4.4 Reengineering Approches

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 deep understanding of the origin system/subsystem, those based on the analysis of input and outputs [7] and those based on hybrid approaches.

4.4.1 Black-box Approaches

Black-box or external approaches(Figure 3 (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 on existing system. Black-box approaches are often based on wrapping techniques.

Wrapping consists of surrounding a piece of software with a software layer that hides unwanted complexity and exports a new interface. Wrapping is used to remove mismatches between the interface exported by a software artefact and the interfaces required by current integration practices. Since a wrapping impacts over devices aiming to enable communication, it is only applicable on the different levels of interoperability: Third party solutions, exhibited API/ABI, Architecture. Figure 4(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.

4.4.2 White-box Approaches

White-box or internal approaches(Figure 3 (b)) are named after the fact that they consider the internal composition of the system. Often based on an initial reverse engineering process required to gain a deep internal understanding of the origin system/subsystem. This process aims normally to identify components and relationships at different levels of abstraction (classes, patterns, dependencies etc). Automatic and semi-automatic white-box techniques normally are based on the production of representational models, such as metamodels or ontologies. These approaches are often imply high amount modifications on the existing system. White-box approaches are often based on transforming techniques.

Transforming consists on producing 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 directly or indirectly on the source code, it can be applied to all the different internal and external parts of software. Architecture, Design, Language, exhibited and used API/ABI, Paradigm, Deployment environment, Third party products. Figure 4(b) shows a schematic of a hypothetical transformed system. As the image shows, transforming implies to modify all the internal design, and even add or remove existing source code in order to articulate the system into the new environment.

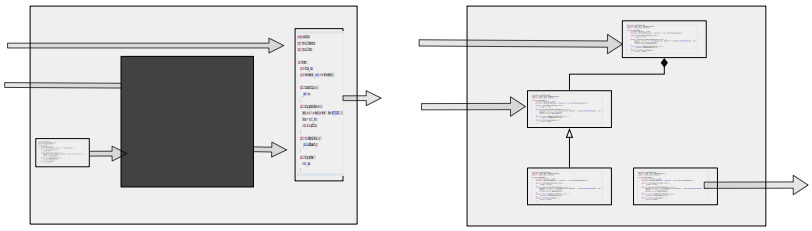

Fig. 4 Produced artefacts schematics

4.4.3 Grey-box Approaches

Grey-box or hybrid approaches (Figure 3 (c)) are those approaches that use internal approaches for enabling certain granularity on external approaches, or using external general 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 recognize parts of a system and decomposing it, enabling to wrap parts of a system instead of the full system [12]. We found the usage of the second kind of approach specially on modernization processes that are required to delegate what once was a concern of the system 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) [9].

4.5 Process

In section 2 #RQ2 expressed our concern of understanding what the proposed processes on migration are? In this subsection we give a framework to interpret the literature.

We distinguish the word *procedure* from the word *process*. By process, we do refer to the steps to follow, by procedure we understand the implementation and execution of the process.

Modernization & Renovation are often long and highly risky enterprises [29,20]. Such projects often deal with Legacy Systems that suffer from both Decadence and Obsolescence on multiple artefacts. Such projects often respond to multiple direct and indirect drivers expected to be satisfied. In short such projects are bounded to a lot of circumstantial variables, that impose the instrumentations of many times ad-hoc processes, what makes specially hard (if not impossible) to generalize practical processes (as practical process we understand an exhaustive definition able to fit all possible cases of modernization and renovation), but only some process form for the sake of knowledge organization.

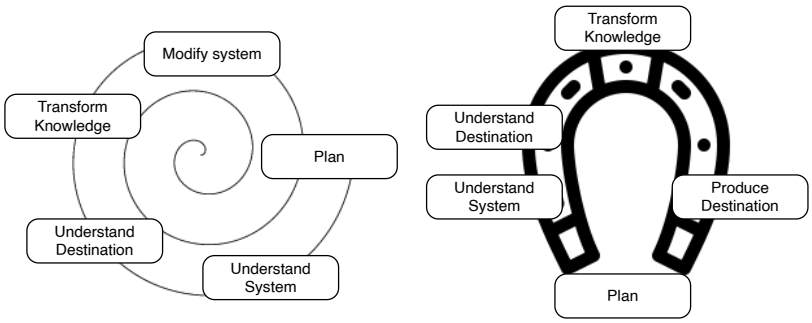

Fig. 5 (left) Spiralling Model (right) Horseshoe Model

According to our studies and experiments we recognize that in general Modernization & Renovation respond to two procedures forms shown on Figure 5. On Figure 5(right) we find the classical Horseshoe reengineering model [15]. This model is related with processes that takes as input a system and gives as output a new system that should comply with the old and new specifications.[36, 1, 19, 4]. *Disadvantages:* Due to the forking nature of the process, it is important to remark that this kind of process threats the maintenance and development of new features. Since the produced software takes more time to be delivered, it reduces Also, the ability to acquire feedback from users. Products may take much time to be implemented seen and valorized. *Advantages:* On the other hand it does not threaten the quality or stability of the origin system.

On Figure 5(left) we find the classical Spiralling forward-engineering model [15]. Related with the nature of a process that takes as input a system and gives as output the same system but modified. [38, 10] *Disadvantages:* Due to the continuous integrating nature of the process, is important to remark that this process threatens the stability and internal consistence of the system. *Advantages:* on the other hand, the feedback is guaranteed by the usage of the systematic delivery of the running system, and the products of this process are available earlier.

	Modernization		Renovation
Procedure	Migration	Adaptation	Restructuring
Horse Shoe	White-box / Grey-box	NF	Refactoring / Transform
Spiralling	Black-box / Grey-box	White-box / Black-box	Refactoring / Transform

Table 4 Procedure x Solution (NF: Not found)

As shown in Table 4, we find that Migration responds to a Horse Shoe process, due to the mutual exclusion nature of the migration. Adaptation on the other hand may responds to both. On the renovation side, under reengineering we find both kind of processes. Below we present each step.

Table 5 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.[38]	NER	Accessibility	Web access	NER	NER	Online market	NER
	Lean and Mean Industrial approach [29]		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 [11]	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. [32]	NER	Accessibility	Network Access	NER	NER	NER	NER
Client-Server To Web	Interoperability middleware for Cobol application wrapping [9]	NER	Interoperability	Web access	NER	Offering services online	NER	NER
Paradigm Change	Object Model Discovery based on source code patterns [41] [40]	NER	Modularity & Interoperability	Web access	NER	Offering services online	NER	NER
Translation	C to Java by patterns and grammatical translation [24]	NER	Interoperability	Reusability — Web Access	NER	NER	NER	NER
	PL/IX to C++ by patterns and grammatical translation [22]	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. [13]	Language	NER	NER	NER	Offering services online	NER	NER
	Model Driven Engineering: PSM to KDM. KDM To Code. [35]	Framework	NER	NER	NER	NER	NER	NER
	Knowledge-based GUI selective translation [25]	Interface	NER	Use windows GUI API	NER	NER	NER	NER
	Procedural code interaction patterns recognition for building interaction model [26]	Interface	NER	GUI	NER	NER	NER	NER
	Java AWT to XIML conversion [11]	NER	Accessibility	Web access	NER	NER	NER	NER
Library migration	Ontological matching for code rewritting [39]	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 [10]	NER	Interactivility	GUI Access	NER	NER	GUI tool market	NER

NER No Explicit Relationship in the articles considered

Plan Activities in this phase are normally conducted to define the reach and expectations of the process at operational level[18], including risk and feasibility assessment. [27] Recognizes that risk is related with planning "Minimizing the migration risk is a key requirement. The most common strategy is to follow an incremental approach to minimize the risk". [29] Remarks the importance of understanding "Associating costs and risks to core activities makes the core an even more powerful tool for planning how to do migration"

Understand Origin System Activities in this phase are normally 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 [29]), to computational models built from reverse engineering (as those proposed specially by model driven engineering [2,35,13]) or ontological methods [39], 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 prioritize tasks, etc [29,8]) to the input of automatic/semi-automatic algorithms with many usages such as code enhancement recommendations, language translation etc [36,5].

Understand Expectations of the Destination System Activities in this phase are normally conducted to acquire knowledge of the destination system. [1,4]. These activities are normally accomplished manually. The proposed activities are related to the understanding of how is the new system is going behave and to interact with the environment. This knowledge is required to choose a valid and optimal approach [1] for the process, estimating costs, times, risks and assessing task prioritization [29,8].

Transform Knowledge Activities in this phase are normally conducted to work over the acquired knowledge in terms of the process expectations. [1] These activities are accomplished manually, semi-automatically or automatically. The nature, size and order of the tasks change from a white-box approach to a black-box approach. Still, these activities range from the intellectual understanding (of the required transformations and re-structuration to instrument in order to accomplish the target expectations of the current process as proposed by [29], to leverage and actually transform computational models built during the previous step, to fit better on the destination restrictions [25,3], or [41] who uses clustering algorithms over models for proposing classes and methods in the context of procedural to object oriented migrations).

Modify system Specific for spiralling procedures. Activities in this phase are normally conducted to apply the transformed knowledge on the current system. These activities are accomplished manually, semi-automatically or automatically. The nature of the modification range from modifies manually some asset of the system (source code, documentation, etc) [29,3,10] to the automatic/semi-automatic modification of these assets [39].

Produce Destination Specific for horseshoe procedures. Activities in this phase are normally conducted to use the transformed knowledge for the production of a destination system. These activities are accomplished manually, semi-automatically or automatically. The nature of the production range from the manual creation of the destination system (based on the transformed knowledge), to the automatic/semi-automatic generation of this destination system [36,13,2]

4.6 Process planning

Planning is directly constraint by the ability of breaking down the process into tasks. The smaller and more independent the task can be, the better. In the context of modernization and renovation, this may not be always the case. In all our cases, the ability of splitting the workload into small and manageable task requires high level of decomposability, as pointed by [38,9,22], [33,5] and [1]. And the fact is that decomposability of a system, is related with source code qualities, such as coupling and cohesion (obtained metrics analysis). This means that a decomposable system is normally 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 most of the times a modernization process requires a tightly interleaving renovation process. [38]. And many other times, renovation is just too expensive on an obsolete environment, and therefore it requires a tightly interleaving modernization process [41].

In order to interleave this processes tightly enough to reduce risks, a highly documented and informed iterative strategic plan is required [5]. For obtaining 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 most important tasks-metrics is related with validatability and testability, what also requires decomposability to be possible.

This is why we conclude that for reducing the risks a virtuous circle in between each of these points is required. And this virtuous circle is highly likely to require the help reliable tooling [5,32,34]

On the process of planning, we recognize two different level of planning (as proposed by ISO 9001 [18]: Strategic and Operational.

4.6.1 Strategical planning

Strategical planning is situated on the overall vision of a project of Modernization & Renovation. At this level, the important activities are the recognition of "strategic" milestones [5,33], and their linking in terms of iterativity. Strategic milestones in the context of modernization may imply the recognition of which parts of the system to modernize, and in which order of priority acknowledging dependencies.

Iterativity is taken as a key property to make a migration into a possible process [5]. This feature is related with the way to define the project's roadmap. It is managed at strategical level. In order to respond the **first part of #RQ3**, according to the SLR, the most important pillars to ensure iterativity, on the context of Modernization & Renovation, are (i) Breaking the project into milestones. [5,33] (ii) Each of the milestones must be independent and testable. [5], (iii)The milestones must be efficiently prioritized. [33] [22] (iv) Each milestone should work on the refinement of the previous milestones. [1] (v) Instrumentation of feedback devices. [5,41]

4.6.2 Operational planning

Operational planning is situated on the vision of one specific iteration of a project of Modernization & Renovation. At this level, the important activities are the recognition of "operational" milestones, and their linking in terms of incrementality. Operational milestones in the context of modernization may imply the recognition of sprint-length tasks, along with tasks dependencies, priorities opportunities of parallelism [33], and the mapping to incremental change, and systematic validation of the results.

Incrementality is proposed for reducing operational risks [22]. This feature is related with the way to define the tasks to do in order to accomplish one strategical milestone. It feeds back to the strategical planning on how the milestone was accomplished. It is managed at operational level. In order to respond the **second part of #RQ3**, according to the SLR, the most important pillars of incrementality, in the context of Modernization & Renovation, are: (i) Deep and systematic understanding of the origin system is required for task measuring.[27,9] (ii) Tasks must be the result of coarse-grained decomposition of larger tasks. [1] (iii) Tasks must be measured and their impact on the next tasks understood.[5] (iv) Tasks outputs must be mergeable with the results produced before and those to be produced after [40] (v) Tasks outputs must be tested. [5] (vi) Instrumentation of feedback devices. [5]

Validatability is required as it is the main feedback for operational planning, informing evolution and increment accomplishment. Validability is managed at task level. In order to respond the **#RQ4**, according to the SLR, the most important pillars of validation and evaluation, in the context of Modernization & Renovation, are: (i) Unit testability. The task output must allow and instrument tests that proof their behaviour [5]. (ii) Integration testability. The task output must allow to be tested on the expected context of usage of the output [5]. (iii) Performance measurability. The task performance must be measured [33,9,22]. (iv) Comparability. On the context of automatic/semi-automatic transformation, the task must be comparable with the manual equivalent outcome [41,9]. (v) Correctness. On the context of automatic/semi-automatic transformation, the tasks must respond to correctness analysis and testing

[25], [5]. (vi) Soundness. On the context of automatic/semi-automatic transformation, the tasks must report the same results for equivalent objects. [22] (vii) Understandability. The result of a task must be interpretable, for further comparisons with the previous state/origin system. [40,9].

4.7 The impact over the Legacy system

A general definition of a reengineering solution (migration included) aware of the different elements and concepts involved in a software migration, that can be used to respond our **#RQ1** is: *Given a legacy system and a driver (which implies an evolution of the given legacy system), a solution is a process (subsection 4.5) that applies a specific method subsection 4.4 – which responds to a general approach (subsection 4.4)– in order to achieve an objective (subsubsection 4.3.1) that contributes to the satisfaction of the given driver (subsubsection 4.3.2), by impacting specific parts of the given legacy system(subsubsection 4.1.1).*

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

Migration solutions have been gathered and divided by approach in the following three tables. Black-box approaches in Table 6. 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 7. We can see in this table that most of the works are on how to enable architectures, such SOA, cloud, etc. White-box approaches are in Table 8. We can see in this table that the heterogeneous, from paradigm to architectural migrations. The amount of variables that are accessible from white-box are much more. Nevertheless, white-box approaches are more detailed, normally related with the ideas of risk and time-consuming.

Adaptation solutions have been gathered and divided by approach in the following three tables. In Table 9 and Table 10 we find the different classifications on Adaptation proposals. Table 9 holds the only black-box adaptation approach in our literature. This approach just bridges request to some internal and well-known service. Finally, our last Table 10 holds the whitebox approaches on adaptation. We find that the adaptation proposals are interesting since they tackle down problematic as software development assumptions, control, and hardware implications.

Table 6 Migration - black-box approach

Procedure	Objective	Solution	Data	GUI	Arch
Reengineering spiral	Migrate Data Access Protocol	Database Gateway [7]	X		X
		XML Integration [7]	X		X
	Centralized to distributed database	Database replication [7]	X		X
	Migrate Text to GUI	Screen Scrapping [7]		X	X

Table 7 Migration - grey-box approach

Procedure	Objective	Solution	Data	GUI	Func	DS	Arch
Horse Shoe	Migrate To Service	Object-Oriented Wrapping [7]			X	X	X
		Component-Oriented Wrapping [7]			X	X	X
		Service identification, code adaptation, wrapping and orchestration.[38]				X	X
		Lean and Mean Industrial approach [29]			X	X	X
		Design patterns to reuse architecture [3]				X	X
		MASHUP [12]			X	X	X
	Enable Cloud	SMART [12]			X	X	X
		REMICS [12]			X	X	X
	Client-Server To Web	Interoperability middleware for Cobol application wrapping [9]		X		X	X
Reengineering Spiral	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 [12]	X		X	X	

Arch Architecture DS Design
Func Functionality

Table 8 Migration - White-box

Procedure	Objective	Work	GUI	PD	Lg	3P	U-API	DS	Arch	MU	RT
Horse Shoe	Paradigm Change	Object Model Discovery based on source code patterns [41] [40]		X				X			
	Translation	C to Java by patterns and grammatical translation [24]		X	X	X					X
		PL/IX to C++ by patterns and grammatical translation [22]		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. [35]	X		X			X			
		Knowledge-based GUI selective translation [25]	X		X						
		Model Driven Engineering: PSM to KDM. KDM Modified. KDM To Code. [13]	X		X			X			
		Procedural code interaction patterns recognition for building interaction model [26]	X		X						
		Java AWT to XIML conversion [11]	X	X							
	Library migration	Ontological matching for code rewritting [39]				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 9 Adaptation - Black-box

Procedure	Objective	Work	GUI	Arch
Reengineering spiral	Enable web access	CGI Integration [7]	X	X

Table 10 Adaptation - White-box

Procedure	Objective	Work	GUI	Func	DS	Arch	MU	Ctrl	HW	RT
R. Spiral	Adapt batch to support interactive control	Lessons on converting a compleX software (compiler) to support interaction [10]			X	X	X	X		
	Adapt UI to multiple devices	Mix multiple representations of one UI and serve it according to screen's size [11]	X		X	X				
	Adapt embedded system to support networking	Assess and modify from hardware to software to add network capabilities. [32]		X	X	X			X	X

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

Table 11 Classified Articles - Part 2

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 [35]	GWT Web application	Move from dying technology	UI Translation	Migration	White-box	Transformation	Horse shoe
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	Horse shoe
White-Box Modernization of Legacy Applications [13]	Oracle forms application	Moving from dying technology	UI Translation	Migration	White-box	Transformation	Horse shoe
A Survey on Survey of Migration of Legacy Systems [12] (Survey paper)	N/A	Many	Many	Migration	All	All	All
Modernization of Legacy Systems: A Generalized Roadmap [19] (Meta paper)	N/A	Enable new architectural variables	Migrate To Service	Migration	All	All	All
How do professionals perceive legacy systems and software modernization? [20]	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	Horse shoe
Migrating Legacy Software to the Cloud with ARTIST [4]	N/A	Enable new architectural variables	Enable Cloud	Migration	Grey-Box	Wrapping	Horse shoe
Seeking the ground truth: a retroactive study on the evolution and migration of software libraries [8] (Meta paper)	N/A	N/A	Library Migration	Migration	White-box	Transformation	N/A
Searching for model migration strategies [36]	Object Model	N/A	N/A	Adaptation	White-box	N/A	Horse shoe
A lean and mean strategy for migration to services [29](Meta paper)	N/A	Enable new architectural variables	Migrate To Service	Migration	Grey-box	Wrapping	Horse shoe
Extreme maintenance: Transforming Delphi into C# [5]	Delphi application	Move from dying technology	Translation	Migration	White-box	Transformation	Horse shoe
Parallel iterative reengineering model of legacy systems [33] (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 [9]	Cobol application	Enable New Business / Markets	Migrate to service	Migration	Grey-box	Wrapping	Horse shoe
OPTIMA: An Ontology-Based PlaTform-specIfic software Migration Approach [39]	C/C++ application	Move from dying technology	Library Migration	Migration	White-box	Transformation	Horse shoe
Reversing GUIs to XIML descriptions for the adaptation to heterogeneous devices [11]	Java AWT Application	Enable new features	GUI Migration — GUI Adaptation	Adaptation after Migration	White-box	Transformation	Horse shoe

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

Table 12 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 [40]	Procedural Application	Enable new features	Paradigm change	Migration	White-box	Transformation	Horse shoe
Incubating services in legacy systems for architectural migration [38]	C/C++ Application	Enable new architectural variables	Migrate To Service	Migration	Grey-box	Wrapping	Horse shoe
Network-centric migration of embedded control software: a case study [32]	Embedded system	Enable new features	Adapt embedded system to support networking	Adaptation	White-box	Transformation	Spiral
C to Java migration experiences [24]	C application	Enable new features	Translation	Migration	White-box	Transformation	Horse shoe
A framework for migrating procedural code to object-oriented platforms [41]	Procedural Application	Enable new features	Paradigm change	Migration	White-box	Transformation	Horse shoe
A Survey of Legacy System Modernization Approaches [7] (Survey paper)	N/A	Many	Many	All	Black-box	Wrapping	All
Code migration through transformations: an experience report [22]	PL/IX Application	Move from dying technology	Translation	Migration	White-box	Transformation	Horse shoe
Lessons on converting batch systems to support interaction: experience report [10]	Batch application	Enable new features	Adapt batch to support interactive control	Adaptation	White-box	Transformation	Spiral
Reverse engineering strategies for software migration (tutorial) [27] (Meta paper)	N/A	N/A	N/A	Migration	Black-box	Wrapping	N/A
Strategic directions in software engineering and programming languages [14] (Meta paper)	N/A	N/A	Paradigm change	Migration	N/A	N/A	N/A
Rule-based detection for reverse engineering user interfaces [26]	Texte UI Application	Enable new features	UI Translation	Migration	White-box	Transformation	Horse shoe
Workshop on object-oriented legacy systems and software evolution [34] (Meta paper)	N/A	N/A	N/A	All	N/A	N/A	N/A
Knowledge-based user interface migration [25]	GUI Application	Move from dying technology	UI Translation	Migration	White-box	Transformation	Horse shoe

4.8 The taxonomy in action

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

5 Threats to validity

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

The article selection was done based on our understanding of what is related and what is not related taking as input title, and some times title and abstract. This selection threats the impact over the reproducibility of our experiment. To reduce the impact of this bias we run the screening of the articles many times during the writing process, including a 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 organize the reading and to have 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 bias of the researcher. 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 main author. The measures we took for reducing bias are: spacing the first lectures from the coding part, and spacing the process of writing 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 has been properly re-overviewed and analysed on relation to the ongoing taxonomy part.

6 Proposed Research

During our research we found unexplored or barely-explored ground.

Process risk assessment is recognized by most of the articles as one of the most important activity 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 neither systematic classification of risks, nor 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 in between runtime migration and library migration: whenever there is a runtime migration, a library migration becomes compulsory. Also seems to be a correlation in between language migration and runtime migration as well. To have a clear view of the modernization 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, planning, and as a guide for reuse.

Product risk assessment what ever the flavour of 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 specific form. We found only one work that takes the produced system into account during a modernization process [40], by ensuring that the produced quality responds to the expectation. We found none work on acceptance of the product or in the security risk of a hypothetical product of modernization. This may seem to be academic talk, but during migrations we get to use old code in new ways. These new ways surely were not part of the assumption on the development time. The can lead to large security breaches of multiple kinds, we can easily foresee from vulnerabilities denial of service to data leakage.

Metrics and planning during the study we find an explicit relationship in between decomposability and feasibility, but specially due to claims and not to statistical analysis or measuring devices. The link in between the system decomposability (by architecture and by design), the modernization approach and the procedure may be the link required to be able to recommend a specific kind of solution to a specific problem. It may be also a key to understand the material requirements of a smooth incremental modernization process.

Validation and verification Most of the works propose at best an evaluation of tools over a single system, which is not enough to generalize nor systematize. This may seem good enough industrially, but this talk also about the lack of modularity on the approaches in general, and the lack of reusability. Validation and verification may seem also an academic word, but even systematic testability seems neglected on the literature.

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

7 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 Horse shoe and Spiral processes For understanding "How are these processes incremental/iterative?" we summarize all the important planning aspects to have into account. Finally, for exposing "What validations/verifications are proposed?" we summarize the different approaches and what is required to use them.

We discover the lack of systematic bounds on the migration literature. We discover the impact of this lack on the exchange of knowledge, and research development due to the lack of unification. For tackling down 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 over the field.

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

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

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, p. 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. Brant, J., Roberts, D., Plendl, B., Prince, J.: Extreme maintenance: Transforming Delphi into C#. In: ICSM'10 (2010)

6. Charmaz, K.: Constructing grounded theory. sage (2014)
7. 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)
8. 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
9. 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
10. 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
11. 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
12. 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
13. 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
14. 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
15. ISO: International Standard – ISO/IEC 14764 IEEE Std 14764-2006. Tech. rep., ISO (2006)
16. ISO: International Standard – ISO/IEC 25010:2011 – Software engineering – Product quality. Tech. rep., ISO (2011)
17. 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
18. ISO: International Standard – ISO/ICE 90003:2018 – Software engineering – Product quality. Tech. rep., ISO (2015)
19. 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
20. 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)
21. Kitchenham, B., Charters, S.: Guidelines for performing systematic literature reviews in software engineering. Tech. rep., Department of Computer Science University of Durham (2007)
22. 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 Studies on Collaborative Research, CASCON '98, p. 13. IBM Press (1998)

23. Larman, C., Basili, V.R.: Iterative and incremental developments. a brief history. Computer **36**(6), 47–56 (2003). DOI 10.1109/MC.2003.1204375
24. 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)
25. 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/
26. 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)
27. 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
28. Petticrew, M., Roberts, H.: Systematic reviews in the social sciences: A practical guide. John Wiley & Sons (2008)
29. 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
30. Sepulveda, S., Diaz, J., Esperguel, M.: Systematic literature review protocol identification and classification of feature modeling errors (2020)
31. Shull, F., Singer, J., Sjøberg, D.I.: Guide to advanced empirical software engineering. Springer (2007)
32. 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)
33. 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)
34. 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
35. 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. URL https://hal.inria.fr/hal-02019015
36. 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
37. Zabardast, E., Gonzalez-Huerta, J., Gorschek, T., Šmite, D., Alégroth, E., Fagerholm, F.: Asset management taxonomy: A roadmap. arXiv preprint arXiv:2102.09884 (2021)
38. 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)
39. 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)
40. 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)
41. 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)