IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 11 February 2024, accepted 18 April 2024, date of publication 29 April 2024, date of current version 8 May 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3394732

SURVEY

Model-Driven Approaches for Reverse Engineering–A Systematic Literature Review

HANAN ABDULWAHAB SIALA[1,2], KEVIN LANO[1], AND HESSA ALFRAIHI[3]

[1]Department of Informatics, King's College London, WC2R 2LS London, U.K.
[2]Department of E-Commerce and Data Analysis, Faculty of Economics and Political Science, Tripoli University, Tripoli, Libya
[3]Department of Information Systems, College of Computer and Information Sciences, Princess Nourah bint Abdulrahman University, P.O. Box 84428, Riyadh 11671, Saudi Arabia

Corresponding author: Hanan Abdulwahab Siala (hanan.siala@kcl.ac.uk)

This work was supported by Princess Nourah Bint Abdulrahman University, Riyadh, Saudi Arabia, through the Princess Nourah Bint Abdulrahman University Researchers Supporting Project under Grant PNURSP2024R411.

ABSTRACT Many organizations depend on software systems to accomplish their daily tasks, but these systems need to be maintained and evolved to cope with various changes and requirements. Before starting to maintain and evolve software systems, it is necessary to understand them. Reverse engineering plays a crucial role in comprehending various aspects of software systems by extracting different models and diagrams that represent the structure and behaviour of software systems. This article presents a systematic literature review (SLR) to understand the current state of research in model-driven engineering (MDE) for reverse engineering software systems. The considered articles came from five electronic databases (Scopus, IEEE Xplore, Web of Science, ACM Digital Library, and Google Scholar), and were supplemented by additional articles recommended by experts and provided by manual snowballing. From 538 surveyed papers, 83 principal studies were selected, which present the main characteristics of 64 model-driven reverse engineering (MDRE) approaches. These approaches are analyzed and evaluated based on their objectives and characteristics. Additionally, research gaps and areas where more research is needed are also identified. Therefore, the review provides comprehensive answers to several widely interesting questions for researchers and practitioners who are considering using MDRE.

INDEX TERMS Application program, legacy system, model-driven reverse engineering (MDRE), model-driven re-engineering, software application.

I. INTRODUCTION

Many organizations experience growth and complexity in their software systems in order to meet new user requirements, to support new platforms and technologies, and to improve quality. In these cases, the original designs and architectures of the systems are no longer adequate to support the new functionalities that users demand and new technological innovations. These systems can become legacy systems if they are not properly maintained and evolved. Legacy software systems can be challenging to work with because of many factors, including complexity, a lack of documentation, and a limited understanding of the whole system. As a result, maintaining and evolving these systems is critical for any organization [1].

When the software systems of any organisation do not meet the requirements, the organisation needs to choose whether to evolve its legacy software systems to extend their useful life or to completely rewrite or redevelop the systems. It would usually be too expensive and time-consuming to completely replace or rewrite legacy systems. In such cases, it is better to evolve them by making changes to current software systems so that they can meet new requirements and continue to be used for an extended period amount of time. The more a software system is used, the more income the company can get from it.

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

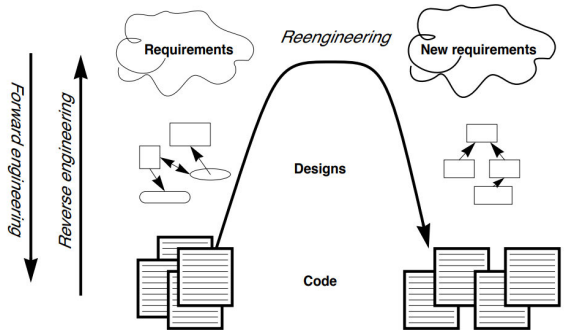

FIGURE 1. Reverse, forward, and re-engineering (extracted from [1]).

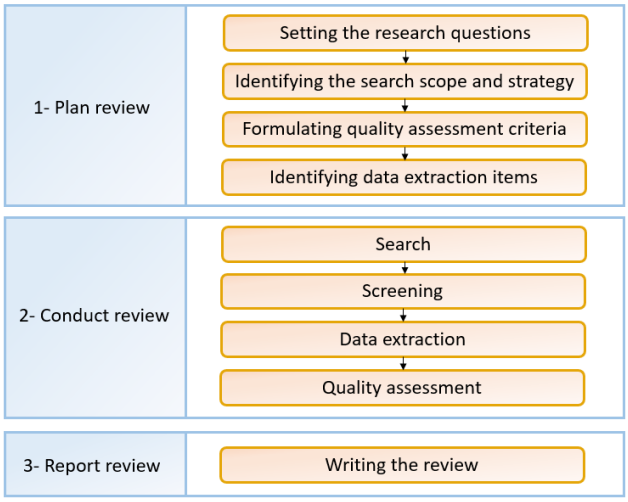

FIGURE 2. Research methodology of a systematic literature review (adapted from [10]).

Re-engineering aims to understand the existing software system, improve its quality, and migrate it to a new technology or platform. Chikofsky and Cross [2] defined re-engineering as "*the examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form.*". Re-engineering includes three stages: reverse engineering, restructuring, and forward engineering. Chikofsky and Cross [2] defined reverse engineering as: "*the process of analyzing a subject system to: a) identify the system's components and their interrelationships; and b) create representations of the system in another form or at a higher level of abstraction*". In a restructuring stage, the representation is mapped to another representation form at the same relative abstraction level. At this abstraction level, the external behaviour of the software system may be preserved while improving the internal structure (Refactoring), or new functionality may be added. Finally, a forward engineering stage, which is also defined by Chikofsky and Cross [2] as: "*the traditional process of moving from high-level abstractions and logical, implementation-independent designs to the physical implementation of a system*".

Despite "reverse engineering" and "re-engineering" referring to different techniques, they are closely related in the context of software development. While analyzing an existing software system to understand its structure and functionality is the main objective of reverse engineering, the objective of re-engineering involves transforming an existing software system by changing its structure, behaviour, or functionality. The aim is to improve its performance, maintainability, or other characteristics. Therefore, reverse engineering is a prerequisite for re-engineering to understand how these systems work before making any changes to them [1]. Fig. 1 illustrates the reverse engineering, forward engineering, and re-engineering concepts.

Model-driven engineering (MDE) [3] is one approach for assisting the reverse engineering process. MDE provides high-level abstractions that allow designers and developers to work with complex software systems through models. These models are essential for different software engineering activities, including forward engineering and reverse engineering [4], [5]. The transformation processes can be implemented using a model transformation chain during the three stages of re-engineering.

Model-driven re-engineering and model-driven reverse engineering (MDRE) are major application scenarios of MDE techniques, where models are employed in the representation, comprehension, and documentation of existing software systems. Models and model transformations can be generated automatically by different reverse engineering techniques and tools. As a result, the challenges of traditional reverse engineering are overcome by generating models of legacy software systems from different perspectives.

The Object Management Group (OMG) develops open standards to design and develop software systems using models. The OMG standards include Unified Modeling Language (UML) [6], Knowledge Discovery Metamodel (KDM) [7], and Object Constraint Language (OCL) [8]. The most widely used standard for modeling software systems is UML. UML includes several diagrams, each with its own distinct function. The most popular kind of UML diagram is a class diagram, which shows classes and their relationships in a software system. OCL is strongly connected to UML diagrams, where it is used as textual specifications within diagrams. UML and OCL representations, as stated by Lano et al. [9], are a good choice for presenting system abstractions since they are international standards, widely used, and supported by various tools.

Numerous MDRE approaches and tools have been developed to represent software systems through a collection of models that concentrate on various aspects of software systems (structure and behaviour). The existing MDRE approaches can be classified into two main frameworks: generic and specific. Generic approaches can be implemented across various application domains or technologies with dif-

TABLE 1. Databases used in the search process.

	Database	Link
1	*Scopus*	https://www.scopus.com/
2	*IEEE Xplore*	https://ieeexplore.ieee.org/
3	*Web of Science*(WoS)	https://www.webofscience.com/
4	*ACM Digital Library*	https://dl.acm.org/
5	*Google Scholar*	https://scholar.google.com/

ferent objectives, whereas specific approaches are designed for specific domains or technologies with well-defined objectives.

MDRE approaches can be completely automated, so human intervention is not needed during the reverse engineering process. Human intervention may be needed in one or more steps of the partially automated MDRE approaches. MDRE techniques can also be classified as *Static*, *Dynamic*, or *Hybrid*. In static techniques, information is extracted from the source code or binary code without executing it, whereas information is extracted from the system during runtime in dynamic techniques. Hybrid techniques comprise both static and dynamic techniques.

The rest of the paper is structured as follows: Section 2 defines the research methodology employed for planning, conducting, and reporting the systematic literature review. Section 3 provides a discussion of the results. Section 4 presents the related literature reviews. Threats to validity are discussed in Section 5. Finally, Section 6 concludes the work.

II. METHODOLOGY

According to Kitchenham's systematic literature review (SLR) approach [10], there are three main steps: planning, conducting, and reporting on the review. These steps along with the related activities are shown in Fig. 2.

A. PLANNING THE REVIEW

The following activities are included in this step: generating the research questions, establishing the search's scope and strategy, developing criteria for quality assessment, and selecting the items for data extraction.

1) IDENTIFYING THE STUDY OBJECTIVE AND SETTING RESEARCH QUESTIONS

Model-driven reverse engineering and model-driven re-engineering play a vital role in comprehending and evolving software systems in different domains. In this context, the goal of this study is to explore and gather knowledge about reverse engineering and re-engineering using MDE approaches. The research questions for this objective follow the PICOC (Population, Intervention, Comparison, Outcome, Context) criteria [11]:

- *Population:* Research publications presenting MDE approaches and techniques,
- *Intervention:* Reverse engineering/re-engineering approaches and practices,
- *Comparison:* Analysis of all reverse engineering/re-engineering approaches in the context of MDE using a pre-defined set of criteria,
- *Outcome:* The state of the art in reverse engineering and re-engineering approaches within the framework of MDE,
- *Context:* Research publications focused on proposing reverse engineering and re-engineering approaches in the context of MDE.

To achieve this, the research questions include two main research questions, with the first one consisting of two secondary research questions. The following are the questions:

RQ1 What are the existing model-driven reverse engineering approaches for software systems?

SQ1.1 What are the objectives of model-driven reverse engineering approaches?

SQ1.2 What are the general characteristics of model-driven reverse engineering approaches?

RQ2 What are the current model-driven re-engineering approaches for software systems?

2) IDENTIFYING THE SEARCH SCOPE AND STRATEGY

Two areas are included in the search scope: 1) the studies that are relevant to reverse engineering and re-engineering approaches; and 2) key studies (selected manually) that extract UML and OCL specifications from source code, due to the importance of both UML and OCL specifications as previously indicated.

To find relevant papers in the first area, we conducted automated searches on popular scientific databases. For the second area, we relied on a selection process guided by domain experts' opinions and forward citation searches. These searches involved looking for recent papers that cite previously selected papers, including those authored by the same authors. This approach helped us identify the main trends and contributions in the field without having to be exhaustive.

a: DATA SOURCES

The study was conducted across five key electronic databases to find scientific papers relating to the research questions. Table 1 shows the list of these databases and their links. Search engines were selected considering their comprehensive coverage, thoroughness, accessibility of available publications, and usage in other literature reviews within this area. We limited the considered papers' time period to those from 2000 to 2023 to ensure their continued relevance.

b: SEARCH QUERIES

The most important keywords relating to the research questions were identified to construct a search string. The search string was defined to include not only the most important keywords but also synonyms for these keywords. Code, programming language, application program, and

TABLE 2. Queries for five electronic databases.

No.	Databases	Query
1	*Scopus* *IEEE Xplore* *Web of Science* *ACM Digital Library*	`TITLE-ABS-KEY(("reverse engineering" OR "reengineering" OR "re engineering")AND ("model driven" OR "model based" OR MDRE) AND (code OR "programming language" OR "application program" OR "software application"))`
2	*Google Scholar*	`("reverse engineering" OR "reengineering" OR "re engineering") AND ("model driven" OR "model based" OR MDRE) AND (code OR "programming language" OR "application program" OR "software application")`

TABLE 3. Quality assessment checklist.

Code	Quality	Qualitative Score
C1	Is the problem of the study clearly defined?	Yes, No, Partially
C2	Is the relevant literature or related work adequately reviewed?	Yes, No, Partially
C3	Does the study suggest a novel approach?	Yes, No, Partially
C4	Does the study give a detailed description of the proposed approach?	Yes, No, Partially
C5	Is the effectiveness of the proposed approach assessed?	Yes, No
C6	Does the evaluation adequately assess the proposed approach?	Yes, No, Partially
C7	Is the contribution of the study obviously stated?	Yes, No, Partially
C8	Does the study clearly report its limitations?	Yes, No, Partially
C9	Does the study clearly report any future insights?	Yes, No, Partially

TABLE 4. Data Extraction Form.

Questions	Data Items
Documentation	Title, Authors, Year, Publisher, Database engine
RQ1	Objectives, Sources, Representations, Scope, Level of automation, Analysis technique, Evaluation technique, Tools/Framework, Platforms
RQ2	Input languages, Output languages, Intermediate representations (models), Re-engineering Technique, Why re-engineering?
RQ1, RQ2	Sources (input), Destinations (output), Objectives, Models, Techniques, Platforms

software application were considered to be synonyms for a programming language, whereas model-driven, model-based, and MDRE were also used to represent model-driven reverse engineering. The abstract queries presented in Table 2 were turned into concrete queries for the various languages that the electronic databases are using.

c: INCLUSION AND EXCLUSION CRITERIA

For deciding which papers to include or exclude, inclusion and exclusion criteria were applied. The following are the inclusion criteria:

1) The article is peer-reviewed (excluding books, posters, patents, slides, panels, tutorials, magazines, or gray literature).
2) The article must present or clarify reverse engineering or re-engineering approaches corresponding to queries 1 and 2 in Table 2.
3) At least one of the research questions ought to be addressed in the article.
4) The article is written in English.

The exclusion criteria are as follows:

1) The article lacks a focus on reverse engineering or re-engineering approaches and fails to offer any information that would address our research questions.
2) The article is either a subset or a duplicate of another paper that discusses approaches to reverse engineering or re-engineering unless it includes additional information.
3) Unavailable on the internet for downloading or cannot be accessed in full.
4) Ph.D., M.Sc., or Undergraduate theses.

3) QUALITY ASSESSMENT CRITERIA

Kitchenham's systematic literature review approach [10] was carried out by giving a qualitative assessment of each study using the checklist in Table 3.

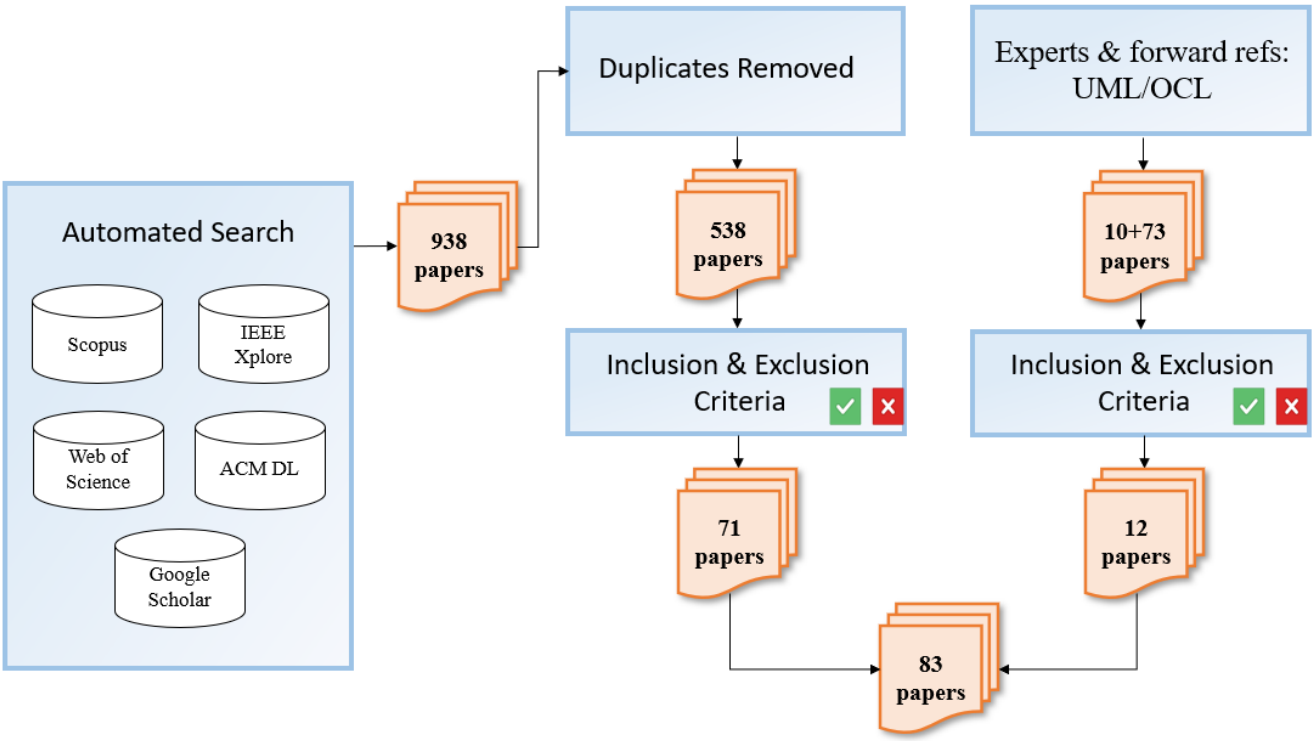

FIGURE 3. The screening process results.

TABLE 5. Results of the searches per databases.

Database	Results
Scopus	217
IEEE Xplore	108
Web of Science	172
ACM Digital library	74
Google Scholar (in any field)	300 (from 17,700)
Google Scholar (title only)	-
Google Scholar (title only*)	67
Total (with duplicates)	**938**
Total (unique papers)	**538**

* code, programming language, application program and software application, are excluded from the search.

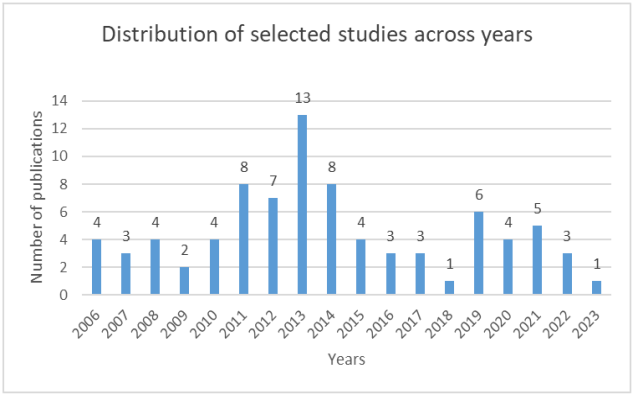

FIGURE 4. Retrieved studies from different years.

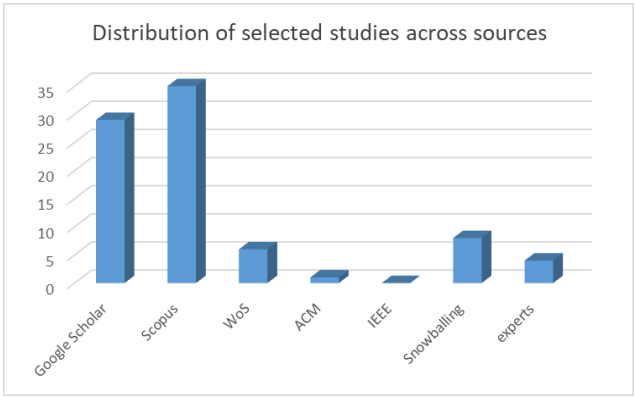

FIGURE 5. Retrieved studies from different sources.

4) IDENTIFYING DATA EXTRACTION ITEMS

The data extraction form (DEF) was utilized to retrieve relevant data from the selected articles. A spreadsheet with rows and columns was used to represent the DEF. Each row represents one reviewed article, and each column stores one data element required to address the research question. Research questions from the planning stage and their related extracted data are given in Table 4. To document the data, we gathered the following information for every article: the article's title, the names of the authors, the year it was published, the publisher, and the database engine that was utilized for retrieving the research paper.

B. CONDUCTING THE REVIEW

The study was conducted in April 2023 using the following procedures: searching, screening, extraction of data, and

TABLE 6. The selected studies and their sources and tools. (S:Scopus, IEEE Xplore, WoS:Web of Science, ACM, GS:Google Scholar, SB:Snowballing, Ex:Experts).

Code	Research studies	Ref.	Source	Tools
A1	**Bruneliere *et al.***, Barbier *et al.*	**[12]** [13]	GS	MoDisco, *Eclipse/EMF*
A2	**Favre L.**, Claudia *et al.*, Martinez *et al.*	**[14]** [15] [16]	S	*MoDisco, ATL, Eclipse/EMF*
A3	Warwas and Klusch	[17]	GS	*MoDisco, Eclipse/EMF, QVT*
A4	**Trias *et al.***	[18] **[19]** [20], [21]	GS S S	RE-CMS, *ATL, Xtext, Eclipse/EMF, EuGENia*
A5	Pérez-Castillo *et al.*	[22]	S	MARBLE, *QVT, JavaCC, Eclipse/EMF*
A6	Bergmayr *et al.*	[23]	GS	FREX, *MoDisco, ATL, Eclipse/EMF*
A7	Fleurey *et al.*	[24]	WoS	MIA
A8	Couto *et al.*	[25]	S	MapIt
A9	Barbier *et al.*	[26]	GS	BLU AGE®, *ATL, Eclipse/EMF*
A10	**Rodríguez-Echeverría *et al.***, Sosa *et al.*	**[27]** [28], [29] [30]	GS GS S	*MoDisco, ATL, QVT*
A11	**Sánchez Ramón *et al.***	**[31]** **[32]** [33]	GS S	UsiResourcer, *Eclipse/EMF, RubyTL*
A12	Sabir *et al.*	[34]	S	Src2MoF,*Eclipse/EMF*
A13	Kotekar C.	[35]	S	*GNU bison*
A14	Grosche *et al.*	[36]	S	
A15	Verhaeghe *et al.*	[37]	S	Casino
A16	Ichii *et al.*	[38]	S	POM/MC, *QVT, Eclipse/EMF, Acceleo*
A17	**Cosentino *et al.***	[39]	ACM	*MoDisco, ATL*
A18		**[40], [41]**	S	BREX, *Xtext, ATL, Portolan*
A19	Moutaouakkil and Mbarki	[42]	S	*glayzzle/php parser, QVT*
A20	Amendola and Favre	[43]	GS	*ObjectAid, ATL*
A21	**Garzón *et al.***, Lethbridge *et al.*	**[44]** [45]	S, GS	Umple
A22	**Silva *et al.*** Campos *et al.*	**[46]** **[47]** [48] [49]	S, GS GS	GUISURFER, *Haskell, QuickCheck, GraphViz, GLR parser generator, GraphTool*
A23	Bai and Liu	[50]	S	SyncTest, *SourceForge, Eclipse/EMF, Eclipse CDT*
A24	Li *et al.*	[51]	GS	XDRE
A25	Katsimpa *et al.*	[52]	S	Unnamed, *HTML TIDY*
A26	Zhang H.	[53]	WoS	J2UML, *JavaCC*
A27	Rodríguez *et al.*	[54]	WoS	SMoT, *Xtext*
A28	El Beggar *et al.*	[55]	GS	*ATL*
A29	Mzid *et al.*	[56]	GS	Unnamed, *GIMPLE, Ecore, XText, Xtend, GCC, Papyrus ALF Plugin*
A30	Alshanqiti *et al.*	[57]	S	VCE, *WindowTester, AspectJ, Henshin*
A31	Bork *et al.*	[58]	WoS	*Fujaba CASE tool*
A32	Béziers *et al.*	[59]	WoS	*MoDisco, Junit, ATL, TestNG, EMFTVM*
A33	Wagner *et al.*	[60]	S	*Elsa, jABC*
A34	Aho *et al.*	[61]	S	Murphy
A35	Sacramento *et al.*	[62]	GS	PARADIGM-RE
A36	Kappler *et al.*	[63]	GS	Java2PCM, *Eclipse/EMF*
A37	Morgado *et al.*	[64]	GS	ReGUI

TABLE 7. The selected studies and their sources and tools. (S:Scopus, IEEE Xplore, WoS:Web of Science, ACM, GS:Google Scholar, SB:Snowballing, Ex:Experts).

Code	Research studies	Ref.	Source	Tools
A38	Shah and Tilevich	[65]	GS	Androider
A39	Lamhaddab *et al.*	[66]	S	iSpecSnapshot, *Xpand*
A40	Briand *et al.*	[67]	GS	
A41	Salihu *et al.*	[68]	GS	AMOGA, *GATOR, Robotium*
A42	Bernardi *et al.*	[69]	S	DPF, *JDT, Xpand, Xtext, Eclipse/EMF, MoDisco*
A43	Gotti and Mbarki	[70]	S	*jdt parser, QVT*
A44	Reus *et al.*	[71]	WoS	*Grammatica, ArcStyler*
A45	Heidenreich *et al.*	[72]	GS	JaMoPP, *QVT*
A46	Candel *et al.*	[73]	S	Morpheus, *Eclipse Metrics 3 tools Eclipse/EMF, ATL, QVT, Acceleo, ClearSQL7*
A47	Methakullawat and Limpiyakorn	[74]	S	*Modelio, DOM*
A48	Sabiri *et al.*	[75]	GS	
A49	Fuhr *et al.*	[76]	GS	SOAMIG
A50	**Bergmayr *et al.*** Menychtas *et al.*	**[77]** [78]	GS	ARTIST tools
A51	Marah *et al.*	[79]	S	RE4TinyOS, DSML4TinyOS, *Acceleo, ANTLR, Eclipse/EMF*
A52	Yang *et al.*	[80]	S	C2AADL_Reverse ,*UPPAAL, AGREE environment*
A53	Pérez-Castillo *et al.*	[81]	S	Qrev, *ANTLR*
A54	Lano K.	[82]	S	*ANTLR, AgileUML*
A55	Lano *et al.*	[9]	Ex	*ANTLR, AgileUML, CSTL*
A56	Yaseen and Fawareh	[83]	SB	MVST, *PlantUML, Coco/R, JavaCC Parser, CSSFormatter, NBJAD*
A57	Cheers and Lin	[84]	SB	Unnamed, *JavaParser, JavaSymbolSolver*
A58	Zaidi *et al.*	[85]	SB	*SDEDIT, Eclipse/EMF*
A59	Fauzi *et al.*	[86]	SB	REVUML, *PlantUML, JavaParser*
A60	**Bouziane *et al.***, BAIDADA *et al.*	**[87]** [88]	SB	*AspectJ, MoDec*
A61	Labiche *et al.*	[89]	SB	*MDWorkbench, AspectJ API, JavaCC*
A62	**Nanthaamornphong *et al.***	**[90], [91]**	Ex	ForUML, *ArgoUML, ANTLR, Modelio, ATL, OFP*
A63	Alsarraj *et al.*	[92]	SB	RCUML
A64	korshunova *et al.*	[93]	Ex	CPP2XMI, *SQuADT, Columbus/CAN, DOT*

assessment of quality. This was done after the research questions were specified, and queries and database engines were identified.

1) SEARCH METHOD

Two search methods were utilized to retrieve articles that satisfy the conditions we determined:

1) The first search method involved searching electronic databases, and the search queries described in Section Search Queries were used. The method consists of the following steps:

   a) The papers were retrieved from the *Scopus*, *IEEE Xplore*, and *Web of Science* databases by using the first query across "title", "abstract", and "keyword" fields.
   b) With regard to the *ACM* database, the first query was also utilized, but the *ACM* database's advanced search engine has limitations when applying complex search queries. Therefore,

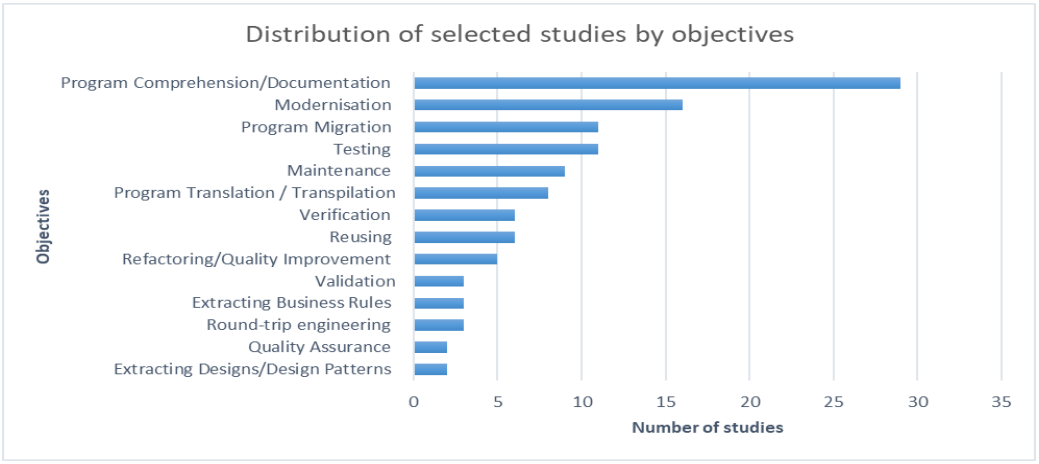

FIGURE 6. Distribution of objectives in the 64 selected approaches.

nine separate queries were executed, and their outcomes were manipulated to exclude duplicate papers.

c) The second query was applied to retrieve papers from *Google Scholar*. Here, the search cannot be limited to "title", "abstract", and "keyword" fields only. As a result, thousands of irrelevant papers are retrieved (with many false positives). This engine was utilized to run the following queries:

   i) The "anywhere" option was used in the query to retrieve papers. The number of retrieved papers was 17,700, so the results are ranked by relevance, and only the first 300 papers were considered.

   ii) No articles were retrieved when the same query was conducted using the "in the title" option. As a result, the query was modified by excluding "code", "programming language", "application program" and "software application" keywords from the search query. The later query retrieved 67 papers.

   The results from *Google Scholar* were checked, and any papers from unreliable sources were excluded from our dataset. The outcomes of *Google Scholar* can be regarded as an addition to the other, more trustworthy databases.

2) To gain a non-exhaustive overview of reverse and re-engineering approaches for extracting UML and/or OCL specifications from source code, a manual search was conducted. UML and OCL were used as the "starting set" for both the forward snowballing phase and for experts to seek about UML and/or OCL. This resulted in a dataset of ten primary articles on reverse and re-engineering approaches identified by experts and 73 more through forward search.

2) SCREENING

After the papers were collected from the previous stage, they were screened for relevancy, and the inclusion and exclusion criteria were employed according to each paper's abstract and conclusion. The introduction and method sections were also read to determine the relevance of a paper in case the abstract and conclusion sections provided insufficient information. Further, the full text of the paper was read if information remained insufficient to make a decision. Due to the difficulty of deciding the relevance of some papers, the guidance of an additional domain expert was requested for nine papers.

a: RESULT

Table 5 presents the results of running the queries listed in Table 2 on each database. The search engines found 938 papers, of which 71 were considered relevant and 867 were rejected, including 400 duplicates.[1] Applying a simple forward search and following the recommendation of an expert, 12 more papers were added to the accepted papers (we did not try to be exhaustive here). As a result, 83 papers were qualified for analysis. The outcomes of the screening process are depicted in Fig. 3. Afterward, the full text of the 83 relevant papers was read and summarized. The distribution across years is shown in Fig. 4, while the distribution across sources is shown in Fig. 5 respectively.

3) DATA EXTRACTION

To gather information relevant to our research questions, the data items shown in Table 4 were repeatedly extracted from each study that was chosen.

4) QUALITY ASSESSMENT PROCESS

Based on the criteria specified in Table 3, the studies gathered through the two search methods were evaluated. Our assessment primarily considered their relevance to the study

[1]The list of accepted and rejected papers is available in a spreadsheet at: https://bit.ly/SLR-MDRE

rather than their overall quality. All papers fulfilled at least three criteria and were included.

III. DISCUSSION

The selected studies were organized and presented in a way that made it easy to answer the research questions accurately. The discussion is divided into three subsections: model-driven reverse engineering approaches, model-driven re-engineering approaches, and problems and challenges.

A. MODEL-DRIVEN REVERSE ENGINEERING APPROACHES

To address the first research question regarding model-driven reverse engineering approaches (RQ1: What are the existing model-driven reverse engineering approaches for software systems?), we used the articles retrieved through the research queries in Table 2 and assessed against the quality assessment criteria in Table 3. The reverse engineering phase of the re-engineering approaches was handled similarly to other reverse engineering approaches and was included in this collection. Of the 83 articles, 64 distinct approaches, named A1 to A64, were identified and listed in Tables 6 and 7, along with the sources of these articles. Additionally, the tools associated with each approach are also presented in Tables 6 and 7. The primary article for each collection is highlighted in **bold**.

1) OBJECTIVES OF MODEL-DRIVEN REVERSE ENGINEERING APPROACHES

The studies were categorized according to their objectives to determine which objectives are frequently accomplished by MDRE approaches. This helped answer the secondary question SQ1.1, (What are the objectives of model-driven reverse engineering approaches?). Table 8 lists fourteen objectives along with the approaches that are associated with them. Program comprehension/documentation and modernization attract the highest number of approaches, with 29 approaches (A1, A6, A12, A13, A14, A19, A21, A24, A26, A27, A29, A30, A32, A33, A37, A39, A40, A42, A43, A44, A47, A49, A56, A57, A58, A60, A61, A62, and A63) and sixteen approaches (A5, A7, A9, A10, A11, A19, A20, A21, A29, A32, A43, A45, A48, A50, A53, and A55) falling under these categories, respectively. While eleven approaches (A4, A7, A11, A14, A15, A20, A38, A39, A48, A49, and A50) have program migration as their goal, eight approaches (A1, A27, A44, A45, A46, A49, A54, and A55) concentrate on translating programs from one language to another—a process known as program transpilation.

Software testing and maintenance come after the program comprehension/documentation goal. Eleven approaches (A2, A16, A22, A23, A32, A34, A35, A37, A41, A55, and A60) focus on software testing, while nine approaches (A8, A14, A23, A25, A32, A42, A56, A59, and A62) concentrate on software maintenance. Verifying software systems is the target of six approaches (A2, A14, A52, A55, A61, and A64), whereas the target of another six approaches (A3, A14, A25, A29, A36, and A42) is reusing software systems. Five approaches focus on refactoring/quality improvement (A1, A14, A21, A55, and A62), while only three approaches (A2, A14, and A64) aim to validate software systems.

In addition, three distinct approaches aim for roundtrip engineering (A7, A31, and A51), and three other approaches (A17, A18, and A28) target extracting business rules. Two other approaches focus on design pattern extraction (A8 and A42), and quality assurance is the aim of two other approaches (A1 and A40). It is worth noting that program comprehension and documentation are the most important goals, whereas design pattern extraction and quality assurance are the least important goals among the MDRE approaches, as shown in Figure 6.

2) CHARACTERISTICS OF MODEL-DRIVEN REVERSE ENGINEERING APPROACHES

Since every MDRE approach has unique properties and features, an ontology was defined to address the secondary question SQ1.2 (What are the general characteristics of model-driven reverse engineering approaches?). The ontology was derived from the one established by Cornelissen et al. [94] and was then refined to thoroughly classify all the MDRE approaches using a variety of criteria. The following are the key classification criteria:

1) Sources (the input),
2) Representations (the output),
3) Scope (whether the approach is generic or specific),
4) Automation (whether the approach is automated or semi-automated),
5) Technique (whether the technique is static, dynamic, or hybrid),
6) Evaluation (whether the approach is assessed using a case study, a comparison with other approaches, or a running example), and
7) Tools (whether the approach generates its own tool, uses existing tools, or does both).

It is important to highlight that some studies lacked sufficient details regarding their approaches. Consequently, this made it difficult to assess the characteristics of various approaches. Specifically, some studies did not explain how their approaches were evaluated or which tools they used.

a: SOURCES OF MODEL-DRIVEN REVERSE ENGINEERING APPROACHES

Table 9 shows the sources for MDRE approaches. These sources can be classified into three categories: code written using third-generation programming languages, including both procedural and object-oriented programming languages, along with web applications. The second category comprises code written using specialized programming languages like Quantum language, while mobile applications and Graphical User Interface (GUI) belong to the ''Other software technologies'' category. Any sources that do not fit into

TABLE 8. Objectives of model-driven reverse engineering approaches.

Objectives	Approaches
Program Comprehension/ Documentation	A1, A6, A12, A13, A14, A19, A21, A24, A26, A27, A29, A30, A32, A33, A37, A39, A40, A42, A43, A44, A47, A49, A56, A57, A58, A60, A61, A62, A63
Modernisation	A5, A7, A9, A10, A11, A19, A20, A21, A29, A32, A43, A45, A48, A50, A53, A55
Program Migration	A4, A7, A11, A14, A15, A20, A38, A39, A48, A49, A50
Program Translation / Transpilation	A1, A27, A44, A45, A46, A49, A54, A55
Testing	A2, A16, A22, A23, A32, A34, A35, A37, A41, A55, A60
Maintenance	A8, A14, A23, A25, A32, A42, A56, A59, A62
Round-trip engineering	A7, A31, A51
Reusing	A3, A14, A25, A29, A36, A42
Extracting Business Rules	A17, A18, A28
Extracting Designs/Design Patterns	A8, A42
Quality Assurance	A1, A40
Refactoring/Quality Improvement	A1, A14, A21, A55, A62
Verification	A2, A14, A52, A55, A61, A64
Validation	A2, A14, A64

TABLE 9. Sources of model-driven reverse engineering approaches.

Programs written using third-generation programming languages		
Sources		**References**
Procedural programming	COBOL	A9, A18, A28, A49, A55
	PL/SQL	A27, A44, A46, A47
	C	A13, A16, A29, A33
	Visual Basic 6	A55
Object Oriented programming	Java	A1,A2, A3, A5, A6, A8, A12, A17, A20, A21, A26, A29, A30, A31, A32, A36, A40, A43, A45, A49, A54, A56, A57, A58, A59, A61, A63
	C++	A23, A24, A29, A33, A56, A64
	Python	A25, A63
	Ruby	A21
	Fortran	A62
	Unspecific	A42, A60
Web Applications	ASP.NET	A25
	PHP	A4, A19, A21
	JSP	A1
	Unspecific	A10, A35
Programs written using specialized programming languages		
Q# (Quantum programs)		A53
AnsiC		A54
C (Special)		A14, A51, A52
Other Software technologies		
Mobile Applications		A39, A41
Graphical User Interfaces (GUIs)		A11, A15, A34, A37, A38
Other/Unspecific		A1, A3, A7, A11, A22, A25, A29, A31, A48, A50

* Various Python versions are supported by the A25 approach.

these categories are grouped together as ''Other/Unspecific'' resources.

The number of approaches for each programming language in the procedural languages category is almost equal, except for Visual Basic 6. Programs written in Visual Basic are the input only for the A55 approach. COBOL programs are the source for five approaches (A9, A18, A28, A49, and A55), whereas the input for A27, A44, A46, and A47 approaches is PL/SQL programs. C programs are input for A13, A16, A29, and A33 approaches.

In terms of object-oriented programming languages, Java is the most commonly used for reverse/re-engineering approaches, with 27 approaches that support it. C++ comes in second place, with only about one-fifth as many approaches as Java. Python has two approaches (A25 and A36), Ruby has one approach (A21), and Object-Oriented Fortran has one approach (A62). However, the approaches A42 and A60 do not specify which object-oriented programming languages they support.

With respect to web applications, the A25 approach uses ASP.NET applications as input, while the input of the A4, A19, and A21 approaches is PHP applications. Bruneliere et al. [12] use JSP pages as input for their approach (A1), and the A22 approach uses Java/Swing code, GWT

TABLE 10. The target representation of model-driven reverse engineering approaches.

Representations			References
Object Management Group Standards (OMG)	Unified Modeling Language	***UML Structural Diagrams***	
		Class diagram	A2, A8, A12, A20, A21, A23, A26, A31, A44, A47, A56, A62, A64
		Object diagram	A31, A56
		UML Behavioural Diagrams	
		Use case diagram	A2, A24, A63
		Activity diagram	A2, A12, A29, A49, A63, A64
		State Machine diagram	A2, A14, A21, A44, A49
		Sequence diagram	A2, A23, A26, A40, A56, A57, A58, A59, A60, A61, A62, A64
		Collaboration diagram	A44
		fUML	A6
		UML concepts	A7, A28, A53, A54, A55
	Knowledge Discovery Metamodel (KDM)		A1, A2, A4, A5, A19, A39, A43, A46, A53
	Object Constraint Language (OCL)		A2, A54, A55
	Meta-Object Facility (MOF)		A2, A44
	Interaction Flow Modeling Language (IFML)		A43
	Software Metrics Metamodel (SMM)		A1, A39
	Application Lifecycle Framework (ALF)		A29
	XML Metadata Interchange (XMI)		A23, A32, A35, A47, A49, A61, A62, A64
Other Representations	GUI model/UI Patterns		A22, A34, A35, A37, A41
	Business Rule		A17, A18, A28
	Web Modeling Language (WebML)		A10, A25
	Design Patterns		A8, A42
	Extensible Markup Language (XML)		A1, A12, A13, A33, A35, A38, A45, A47, A49, A61
	Abstract Syntax Tree (AST)		A2, A7, A12, A14, A21, A22, A23, A26, A27, A29, A32, A33, A36, A42, A43, A44, A51, A52, A53, A56, A59, A62, A64
	Abstract Syntax Tree Metamodel (ASTM)		A1, A4, A19, A39, A43, A46
	Other/Unspecific		A1, A3, A5, A9, A11, A13, A15, A16, A22, A30, A31, A32, A36, A43, A45, A48, A49, A50

source code, or Wx/Haskell code as input. However, the A10 and A35 approaches did not specify the type of web applications they used as input.

The category of specialized programming languages includes several approaches that deal with specific programming languages. The A53 approach analyzes quantum programs developed in Q#, which use qubits instead of bits. The A14 approach deals with embedded C code, whereas the A54 approach takes programs written in ANSI C, the standard version of the C programming language, as input among other source programming languages. The A51 approach deals with TinyOS programs written in nesC, a specialized programming language of TinyOS, where TinyOS is an operating system that is frequently used with wireless sensor networks (WSNs). On the other hand, the A52 approach focuses on multi-task C source code for complex embedded systems.

In the category of other software technologies, mobile applications serve as input for both the A39 and A41 approaches. However, it is worth noting that the A39 approach is specifically designed to address mobile applications developed exclusively for the iOS platform. Regarding GUIs, five approaches use GUIs as input for their processes. The A15 approach processes GUIs of desktop or web-based software systems as input, which are described using generic programming languages, TypeScript, or markup languages [37]. The A11 approach takes MS Windows resource files of GUIs as input, which contain the layout and the properties of GUI elements. In contrast, the A34, A37, and A38 approaches extract information related to GUIs during runtime.

Other MDRE approaches support multiple programming languages. For example, the A1 approach can handle Java, JEE (including JSP), and XML [12]. The A3 approach can reverse engineer Java and multi-agent systems developed in Jadex [17]. The A29 and A31 approaches support Java and other programming languages, such as C and C++. The A11 approach takes legacy GUIs of Rapid Application Development (RAD) software, such as those built with Oracle Forms, Visual Basic, or Delphi, as input. Similarly, the A22 approach takes various types of source code, including Java/Swing, wxHaskell, and C#, as input. The A25 approach takes ASP.NET as input, which includes popular programming languages such as Jscript.NET, Visual Basic.NET, and C#, as well as various versions of Python [52]. However, the A7, A48, and A50 approaches do not specify any particular programming language.

b: REPRESENTATIONS OF MODEL-DRIVEN REVERSE ENGINEERING APPROACHES

Table 10 shows the desired outcomes of MDRE approaches. The table includes various modeling languages from the

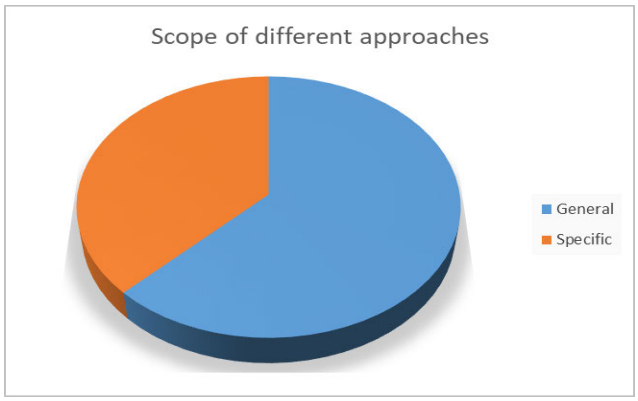

FIGURE 7. Scope of the selected studies.

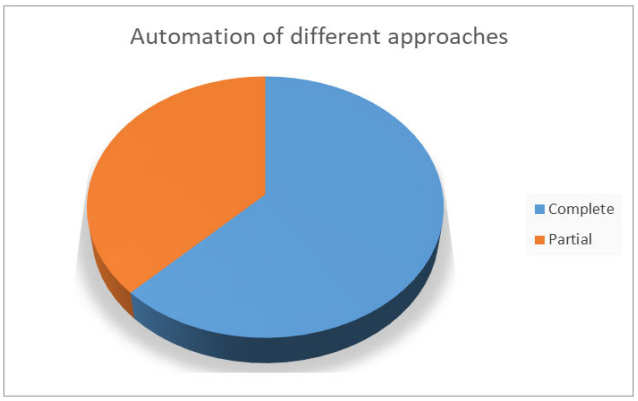

FIGURE 8. Automation of the selected studies.

OMG as well as other representations that fall under the "Other/Unspecific" category. It is worth noting that meta models are not included in the target representations, except for Extensible Markup Language (XML), Abstract Syntax Tree (AST), Abstract Syntax Tree Metamodel (ASTM) [95], which describes the structure of ASTs (metamodel), and those that belong to OMG standards.

From an OMG standard perspective, various diagrams have been utilized, with UML being the most popular modeling diagram and KDM ranks second. Structural and behavioural diagrams are the two main classifications of UML diagrams. The number of approaches that extract different types of UML diagrams from reverse engineering varies significantly. A comparison shows that class diagrams are the most widely supported type of diagrams, with 13 approaches available (A2, A8, A12, A20, A21, A23, A26, A31, A44, A47, A56, A62, and A64). Sequence diagrams are the second most widely supported type of diagrams, with 12 approaches available (A2, A23, A26, A40, A56, A57, A58, A59, A60, A61, A62, and A64). There are approximately twice as many approaches that extract class diagrams as there are approaches that extract activity diagrams (A2, A12, A29, A49, A63, and A64), three times as many approaches that extract class diagrams as there are approaches that extract state machine diagrams (A2, A14, A21, A44, and A49), and five times as many approaches that extract class diagrams as there are approaches that extract use case diagrams (A2, A24, and A63). In addition, the number of approaches that extract sequence diagrams is about double the number of approaches that extract activity diagrams or state machine diagrams, and there are four times as many approaches that extract sequence diagrams as there are approaches that extract use case diagrams. Object diagrams are extracted by the A31 and A56 approaches, while collaboration diagrams are exclusively extracted by the A44 approach. The fUML model is solely extracted by the A6 approach, while A7, A28, A53, A54, and A55 primarily focus on extracting UML concepts.

KDM is the second most widely used OMG standard after UML diagrams. The number of approaches that extract KDM

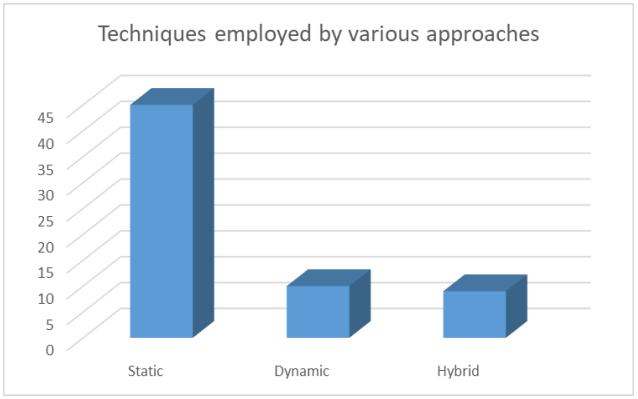

FIGURE 9. Techniques of the selected studies.

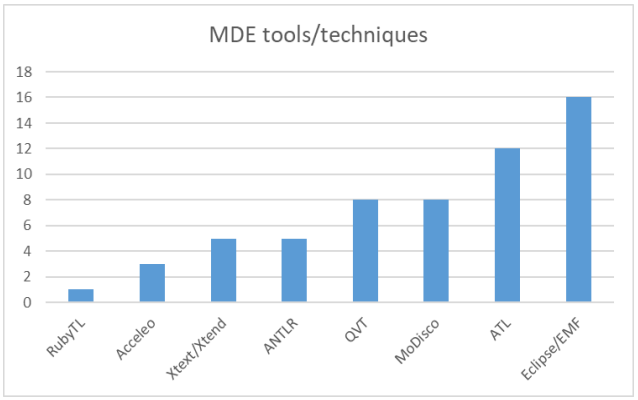

FIGURE 10. MDE tools used by MDRE approaches.

(A1, A2, A4, A5, A19, A39, A43, A46, and A53) is nearly identical to the number of approaches that use other OMG diagrams except for UML diagrams and XML Metadata Interchange (XMI). XMI is extracted by A23, A32, A35, A47, A49, A61, A62, and A64 approaches.

Some approaches focus on extracting different types of diagrams that belong to OMG standards. For example, the A54 and A55 approaches extract UML and OCL specifications, whereas the A44 approach extracts UML and Meta-Object Facility (MOF) diagrams. KDM and the

Interaction Flow Modeling Language (IFML) diagrams are extracted using the A43 technique, whereas activity diagrams and the Application Lifecycle Framework (ALF) diagrams are generated utilizing the A29 approach. Software Metrics Metamodel (SMM) and KDM diagrams are extracted by both the A1 and A39 approaches; however, the A2 approach extracts UML, KDM, OCL, and MOF.

Other representations from the presented approaches that do not belong to OMG standards can be summarized as follows: Regarding GUI models, several research approaches (A22, A34, A35, A37, and A41) utilize reverse engineering techniques to automatically generate GUI models. The A22 presents an approach to reverse engineer GUI code from any programming language to automatically generate GUI models. The A34 and A37 approaches extract GUI models using dynamic techniques. The A35 approach generates web application models by inferring UI patterns, while A41 proposes an approach for reverse engineering GUI models from Android apps.

Business rules are extracted from various source code using the A17, A18, and A28 approaches. The A17 approach extracts business rules from Java source code, while the A18 and A28 approaches extract them from COBOL source code. The A10 and A25 approaches employ reverse engineering to extract the conceptual model of web applications using the WebML notation. XML is extracted along with other representations in the A1, A12, A13, A33, A35, A45, A47, A49, and A61 approaches. On the other hand, the A38 approach extracts a general GUI model in XML format. The AST is a common representation extracted by many approaches. It is a result of the static analysis techniques used to analyze and understand the program. The following approaches extract AST: A2, A7, A12, A14, A21, A22, A23, A26, A27, A29, A32, A33, A36, A42, A43, A44, A51, A52, A53, A56, A59, A62, and A64. The ASTM is extracted by only six approaches: A1, A4, A19, A39, A43, and A46. The A4 approach defines an ASTM for the PHP programming language.

In addition to the previous outputs, several reverse engineering approaches generate different representations of source code that are not included in the aforementioned representations. In terms of C language code, the A13 approach automatically extracts Simulink models from existing C files, while the A16 approach derives Promela models for the SPIN model checker by analyzing the C source code. When examining reverse engineering techniques employed for Java programs, the A30 approach deduces visual contracts, while the A36 approach generates reusable performance models called RDSEFFs. On the other hand, the A1 and A45 approaches extract Java models. In the case of the COBOL language, the A9 approach generates various types of models that represent the structure, behaviour, and data of COBOL applications. For GUIs, the A22 approach extracts a Finite State Machine (FSM) model to depict the behaviour of GUIs from the source code of applications, while the A43 approach acquires GUI models and task models. In the case of multiagent systems, the A3 approach processes source code to extract the design of Belief, Desire, Intention (BDI) agents, enabling the reusability of multiagent system designs [17].

Other methodologies extract diverse representations. From the source code, business process models are derived using the A5 approach, while the output of the A32 approach is the dynamic model. The A15 approach extracts the source code model, which then maps to the GUI model, while the A11 approach derives a Concrete User Interface (CUI) model from resource files. The A48 approach extracts the business model, implementation, data model, and infrastructure model from legacy code. The A49 approach generates Data Flow Graphs (DFGs), which are then synchronized into TGraphs for software system migration. Generic Abstract Syntax Tree Metamodel (GASTM) was obtained using the A1 approach, with no regard for the language being used, whereas the outcomes generated by the A31 and A50 approaches have not been explicitly specified.

c: SCOPE

As shown in Fig. 7, 38% of the approaches were developed for specific systems with particular objectives, including A3, A4, A6, A9, A10, A11, A12, A14, A16, A17, A18, A19, A22, A28, A41, A43, A47, A51, A52, A54, A55, A61, A62, and A64, while 62% of approaches were developed for different application domains with different goals, including A1, A2, A5, A7, A8, A13, A15, A20, A21, A23, A24, A25, A26, A27, A29, A30, A31, A32, A33, A34, A35, A36, A37, A38, A39, A40, A42, A44, A45, A46, A48, A49, A50, A53, A56, A57, A58, A59, A60, and A63.

d: AUTOMATION

As illustrated in Fig. 8, more than three-fifths of the approaches (62%) are fully automated (A1, A4, A6, A8, A12, A14, A16, A17, A19, A20, A21, A22, A23, A24, A26, A29, A30, A32, A33, A34, A35, A37, A38, A39, A40, A41, A42, A43, A45, A46, A47, A51, A52, A54, A57, A59, A61, A62, A63, and A64). By contrast, less than two-fifths of the approaches require human intervention (A2, A3, A5, A7, A9, A10, A11, A13, A15, A18, A25, A27, A28, A31, A36, A44, A48, A49, A50, A53, A55, A56, A58, and A60).

e: THE TECHNIQUES USED

As seen in Fig. 9, all presented approaches use either static, dynamic, or a hybrid of the two techniques. Static analysis techniques are employed by 45 out of 64 MDRE approaches (A3, A4, A7, A8, A9, A10, A11, A12, A13, A14, A15, A16, A17, A18, A19, A21, A22, A23, A25, A26, A27, A28, A29, A31, A33, A36, A39, A42, A44, A45, A46, A47, A48, A50, A51, A52, A53, A54, A55, A56, A57, A59, A62, A63, and A64). In contrast, only ten out of sixty-four MDRE approaches utilize dynamic analysis techniques (A6, A24, A30, A34, A35, A37, A38, A40, A58, and A60). Additionally, nine MDRE approaches incorporate dynamic analysis techniques to complement the results of

TABLE 11. Evaluation of model-driven reverse engineering approaches.

Evaluations		References
Case study	Open source	A3, A4, A8, A15, A20, A21, A23, A30, A39, A41, A42, A45, A55, A62
	Industrial / Practical systems	A1, A3, A4, A5, A7, A8, A9, A10, A11, A12, A13, A14, A15, A16, A18, A22, A24, A25, A27, A29, A33, A35, A36, A37, A38, A40, A43, A44, A47, A49, A50, A51, A52, A53, A54, A55, A56, A57, A58, A61, A62, A64
Comparison		A8, A10, A12, A28, A36, A41, A42, A54, A57, A63
Running example		A2, A6, A14, A17, A18, A19, A21, A22, A26, A28, A31, A32, A46, A49, A58, A59, A60, A61, A63
Unknown/None		A34, A48

static techniques (A1, A2, A5, A20, A32, A41, A43, A49, and A61).

f: REVIEWING THE EVALUATION

The approaches were categorized according to the evaluation method. As shown in Table 11, all but two approaches clearly state how they assess their methods. They utilized case studies, comparisons with other tools, or running examples during the evaluation process.

Most of the approaches were evaluated using case studies. Of these, 42 approaches used industrial systems (A1, A3, A4, A5, A7, A8, A9, A10, A11, A12, A13, A14, A15, A16, A18, A22, A24, A25, A27, A29, A33, A35, A36, A37, A38, A40, A43, A44, A47, A49, A50, A51, A52, A53, A54, A55, A56, A57, A58, A61, A62, and A64), while only 14 approaches used open source systems (A3, A4, A8, A15, A20, A21, A23, A30, A39, A41, A42, A45, A55, and A62). By contrast, ten of the approaches (A8, A10, A12, A28, A36, A41, A42, A54, A57, and A63) used comparison with other approaches as well as other types of evaluation.

Assessment using running examples is the evaluation method for nineteen approaches (A2, A6, A14, A17, A18, A19, A21, A22, A26, A28, A31, A32, A46, A49, A58, A59, A60, A61, and A63). The A32 approach was evaluated by employing a Java project initially consisting of a small number of classes. Subsequently, the number of classes increased to thousands.

Notably, the A3, A4, A8, A15, A55, and A62 approaches use both types of case studies, whilst the A14, A18, A22, A49, A58, and A61 approaches used both running examples and practical case studies to assess their methods. Both Approaches A34 and A48 lacked any details about the evaluation process.

g: TOOLS/FRAMEWORK USED

Computer-Aided Software Engineering (CASE) tools are essential for the success of MDRE. Different commercial MDA tools have been developed. These tools offer forward engineering capabilities and, to some extent, support limited reverse engineering functionalities [96]. Some examples of commercial MDA tools include the Eclipse Modeling Framework (EMF) [97], Modisco [98], and ATL (Atlas Transformation Language). Eclipse/EMF is used to create models and generate Java code, Modisco is used to perform MDRE on legacy systems, and ATL is a model transformation language built on top of the Eclipse platform.

Among the collected approaches, some have developed their own tools for implementing their methods, while others have opted to use existing tools created for other MDRE approaches or different MDE purposes. Tables 6 and 7 provide details on new MDRE tools (listed in regular font) and co-existing tools (listed in *italics*) for each approach. Out of the 64 approaches, 42 have introduced new tools to implement their approaches (A1, A4, A5, A6, A7, A8, A9, A11, A12, A15, A16, A18, A21, A22, A23, A24, A25, A26, A27, A29, A30, A34, A35, A36, A37, A38, A39, A41, A42, A45, A46, A49, A50, A51, A52, A53, A56, A57, A59, A62, A63, and A64), while eight approaches have utilized the existing MDRE tool (MoDisco) for their implementation (A1, A2, A3, A6, A10, A17, A32, and A42).

MDE tools and other software tools are utilized by MDRE approaches to execute their tasks. Eclipse/EMF is employed by 16 approaches (A1, A2, A3, A4, A5, A6, A9, A11, A12, A16, A23, A36, A42, A46, A51, and A58). Additionally, ATL is used by 12 approaches (A2, A4, A6, A9, A10, A17, A18, A20, A28, A32, A46, and A62), and QVT is employed by 8 approaches (A3, A5, A10, A16, A19, A43, A45, and A46). Furthermore, ANTLR is utilized by five approaches (A51, A53, A54, A55, and A62), and Xtext/Xtend is chosen by five approaches (A4, A18, A27, A29, and A42). It is worth mentioning that there are three approaches (A14, A40, and A48) for which no information is available regarding the tools they used. The frequency of MDE tool usage by MDRE approaches is shown in Fig. 10.

B. MODEL-DRIVEN RE-ENGINEERING APPROACHES

To address the research question RQ2 (What are the current model-driven re-engineering approaches for software systems?), the articles retrieved by search queries in Table 2 were used. *Program migration* and *program translation* were selected out of all re-engineering activities for this study. These two activities were selected due to their higher popularity compared to other re-engineering tasks, their relatively well-defined nature, and their fundamental role in other re-engineering activities. Additionally, the number of articles retrieved for other re-engineering activities, not including program migration and program translation, during our research was relatively limited. Even though

TABLE 12. Destinations of program migration approaches.

Destinations		Approaches
Different platforms	Cloud Environment	A48, A50
	Mobile Platform	A20, A38, A39
	GUIs	A11, A15
	J2EE (from Mainframe)	A7
Web-based application	Content Management System (CMS)	A4
	Rich Internet Applications (RIA)	A10

program translation could be classified as a subset of program migration, we have opted to treat them separately. Program migration emphasizes changing the platform and technological base of a system, for example, from a desktop application to a system organized as web services, while program translation is concerned specifically with translating code between different programming languages, like Python and Java.

1) PROGRAM MIGRATION

To get a deeper understanding of the approaches considered for program migration, we classified the studies according to the platform they intended to migrate to. Even though web-based applications can fall under both program migration and program translation categories, we chose to categorize them as program migration since our focus in the upcoming sections is on program translation. As shown in Table 12, a total of ten approaches for program migration were identified from the 64 retrieved approaches. In terms of cloud migration, Sabiri et al. [75] (A48) and Bergmayr et al. [77] (A50) both sought to migrate legacy software systems to the cloud. Many researchers have also shown interest in mobile applications. Amendola and Favre [43] (A20) suggested utilizing the MDA approach to modify Customer Relationship Management (CRM) systems for mobile platforms. Shah and Tilevich [65] (A38) proposed an approach to assist with porting user interfaces between different mobile platforms and devices. Lamhaddab et al. [66] (A39) suggested an approach for porting mobile apps from iOS to Android, where a semantic mobile app model is automatically extracted and then analyzed using the iSpecSnapshot tool. The analysis results are then converted into specific target models, which are further transformed into the Android UI skeleton and functional specification documentation.

From a GUI perspective, Sánchez Ramón et al. [31], [32], [33] (A11) proposed a re-engineering approach for GUIs. The approach utilizes resource files that define the structure and behaviour of GUIs to derive a CUI model. The generated model is then edited to create a new GUI. Shah and Tilevich [65] (A38) also developed an approach for porting the GUI of a system to another GUI framework, platform, or device. Additionally, Verhaeghe et al. [37] (A15) introduced a framework-agnostic approach to migrate visual aspects of GUI from one framework to another. Fleurey et al. [24] (A7) presented their approach for migrating a large-scale banking system developed using the COOLGEN IDE, from Mainframe to J2EE.

With regard to web-based applications, Trias et al. [18], [19], [20], [21] (A4) proposed an approach for migrating from web applications to applications based on content management systems (CMS), while Rodríguez-Echeverrí et al. [27], [28], [29] and Sosa et al. [30] (A10) proposed their approach for modernizing Rich Internet Applications (RIAs) from non-model-based data-driven web applications.

2) PROGRAM TRANSLATION

Program translation can be considered a specific aspect of language migration that involves converting code between different programming languages and is currently receiving a lot of attention. In addition to the papers retrieved from previous search queries, we chose to incorporate additional articles recommended by experts and collected from digital libraries to further identify re-engineering approaches that support program translation. We decided to take into account the following considerations to obtain a comprehensive understanding of the approaches associated with program translation:

1) Whether these approaches use machine/non-machine learning techniques to perform program translation,
2) Whether these approaches are capable of manipulating programs written in multiple programming languages or if they are restricted to a single language,
3) Whether these approaches extract diagrams when translating programs between different programming languages.

a: PROGRAM TRANSLATION APPROACHES AND NON-MACHINE LEARNING TECHNIQUES

To gain an overall comprehension and a fair assessment of any non-machine learning program translation approach, we identified the programming languages that each approach can translate from and to. This allowed us to determine whether an approach can translate between multiple languages or is only designed for translating between two specific languages. Accordingly, we can classify the program translation approaches into two categories: ''one-to-one'' and ''one-to-many''. It is worth mentioning that the A1 approach was eliminated from this comparison because there is not enough information about the translation process.

On the one hand, there are four approaches from the retrieved articles that can be categorized into the ''one-to-one'' category. These approaches are A44, A45, A46, and A49. Each of these approaches takes programs written in one language and translates the desired program into another language.

1) Reus et al. [71] (A44) introduced an approach to reverse engineer legacy PL/SQL systems to language-independent UML models, such as class, statechart, and collaboration diagrams. They then offer code generation to transform these models into Java

classes. However, their approach only generates code stubs without any functionality.

2) Heidenreich et al. [72] (A45) proposed an approach to automatically migrate Java code to a newer version of the Java language. They created a tool called JaMoPP to create models of Java source code, such as UML models. They then used a standardized model transformation language to transform the models into a newer version of the Java language.
3) Candel et al. [73] (A46) introduced a model-driven re-engineering approach for migrating to Java code from PL/SQL monolithic code on triggers and program units. They used a systematic process to build a chain of model transformations that carried out the re-engineering. They used and evaluated the KDM in a code migration scenario.
4) Fuhr et al. [76] (A49) used model-driven technologies to present an adaptable migration process model and tool support. Their model was applied to two case studies: migration of a language from COBOL to Java and migration to a service-oriented architecture from a monolithic software system [76]. Both UML 2.0 activity diagrams and UML 2.0 state diagrams can be used in the migration process.

On the other hand, other approaches are not tied to a specific programming language, and they can translate programs written in one language into multiple other languages. Within this category, we found only three approaches, namely A27, A54, and A55.

1) Rodríguez et al. [54] (A27) proposed an approach to translate PL/SQL code that is integrated within Oracle Forms applications to Java or.Net languages. The proposed approach can be applied to other source and target languages.
2) Lano [82] (A54) proposed an approach for translating programs from Java or ANSI C to other languages using MDE. First, the source program is reverse-engineered into UML and OCL specifications. These specifications capture the essential structure and behaviour of the program without being tied to any specific language. Using AgileUML, the specifications are translated into code in a target language, including Java, ANSI C, Python, C++, Go, Swift, or C# [82].
3) Lano et al. [9] (A55) introduced an Agile Model-Driven Re-Engineering (AMDRE) approach. AMDRE first abstracts the existing software system into UML/OCL specifications, then applies refactoring and rearchitecting transformations to improve the system design's quality, and finally generates code in multiple target languages from the restructured and abstracted UML/OCL representations. Lano et al. used AMDRE to translate VB6/VBA code to Python 3 and COBOL code to Java and C#. AgileUML is used to provide code generators for ANSI C, C++, C#, Python, Go, Swift, and Java [9].

As we can see from the aforementioned information, some non-machine learning program translation approaches can manipulate programs written in multiple programming languages (A27, A54, and A55), while others are restricted to a single language (A44, A45, A46, and A49). It is also noteworthy that these approaches extract different types of diagrams as an intermediate representation. Additionally, Java stands out as the most prominent target language among these approaches.

b: PROGRAM TRANSLATION AND MACHINE LEARNING TECHNIQUES

Machine learning (ML) has been used to accomplish different tasks in software engineering, such as deployment, testing, and maintenance [99], and it has been extensively utilized in program translation as well. Various techniques have been used to translate programs between programming languages using either supervised or unsupervised machine learning, including:

1) Roziere et al. [100] proposed TransCoder, a highly accurate unsupervised machine translation model to automatically translate functions between Python, C++, and Java. The model was trained using a significant amount of source code from GitHub, and it uses a sequence-to-sequence architecture with an attention mechanism. It performs better than currently used rule-based translation techniques and operates without requiring human supervision.
2) Zhu et al. [101] introduced a new dataset called CoST containing parallel data of seven programming languages (C++, C#, Java, Javascript, PHP, Python, and C) at both the snippet level and the program level. By utilizing multilingual snippet translation pre-training and multilingual snippet denoising auto-encoding, the authors introduced a novel program translation model. The evaluation of the suggested model shows that it performs better than the baseline models for both snipped-level and program-level translation [101].
3) Ahmad et al. [102] proposed an unsupervised approach using code summarization and generation as an alternative to traditional back-translation techniques, which are used to ensure accuracy between the source and target languages. In code summarization, a model is trained on code snippets to produce natural language (NL) summaries. On the other hand, the model learns to generate code from NL summaries in code generation.
4) Rithy et al. [103] introduced XTest, a parallel multilingual corpus for code translation. It includes parallel programs of nine programming languages (Ruby, Go, Java, C++, C#, Javascript, PHP, Python, and C), along with test cases and problem statements in English. Using the XTest dataset, thirty systems that translate code through these languages were developed.

5) Chen et al. [104] proposed a tree-to-tree neural network that combines a tree encoder and a tree decoder to translate from a source tree to a target tree. They noticed that their approach performs better than other neural translation models, especially when using the attention mechanism.

In contrast to non-machine learning program translation approaches, the information mentioned above reveals that most of the presented machine learning program translation approaches have the ability to handle programs written in multiple programming languages. However, these approaches do not generate diagrams during the translation process. Instead, the translation results rely on the learned intermediate representation in the ML model derived from both the source code and target code found in the dataset.

C. THE PROBLEMS AND CHALLENGES

The studies faced several difficulties when applying their approaches. The main challenges are presented as follows:

a: ACCURACY

MDRE approaches need to be able to construct models that identify key components of software systems and remove unnecessary details. Providing complete and accurate models of software systems can be difficult, especially with large and complex software systems that have errors and/or inconsistencies.

b: HETEROGENEITY

MDRE approaches can face many difficulties, especially when dealing with a variety of programming languages, each with unique characteristics that distinguish it from the others. In addition, MDRE approaches deal with different software platforms, frameworks, architectures, and technologies. It is necessary to consider such language and platform variations to properly analyze the software system. However, this requires additional manual effort to construct or configure the MDRE tools.

c: SCALABILITY

MDRE approaches face a difficult challenge when dealing with real-world software systems and extracting the required information from the source code. This is because these systems have complex architectures or are developed in multiple programming languages. Additionally, more time and resources are required as the size and complexity of a software system grows.

d: HANDLING DYNAMIC BEHAVIOUR

It is a difficult task for MDRE approaches to capture runtime behaviours and dependencies between software components and include them in different models. Traditional reverse engineering approaches often prioritize static artifacts such as source code, making the extraction of behavioural diagrams such as UML sequence and communication diagrams extremely challenging.

e: UNDERSTANDING THE SEMANTICS OF THE SOURCE CODE

MDRE approaches need to be able to understand the semantics of the source code. This involves understanding the overall purpose of the software system, the goals of its different functions or methods, and the interactions between various components of the source code. Extracting this information automatically from source code can be very difficult, especially with large and complex software systems. Interactive (semi-automated) techniques involving input from system experts are typically necessary.

f: LACK OF MACHINE LEARNING TECHNIQUES FOR MDRE

There is a lack of MDRE approaches that use machine learning to extract specifications from source code. This appears to be a gap in current research and indicates the need to study several machine learning techniques. These techniques assist MDRE approaches by recognizing relevant information within source code, understanding its semantics, and dealing with multiple programming languages and frameworks. Machine learning, especially LLMs, can be used to reduce the human effort and various resources required to reverse engineer large and complex software systems.

g: PROGRAM TRANSLATION

Translating reverse-engineered models to target code with equivalent semantics to the source, particularly in the case of multiple target programming languages, presents a complex challenge. Some approaches address this problem by first abstracting precise UML and OCL representations from the source code and then using an MDE tool such as AgileUML to translate the program into the chosen target languages. A54 and A55 exemplify such an approach.

IV. RELATED WORK

Reverse engineering has been an active research topic over the last two decades. Several literature reviews about the topic have been published in the scientific literature. Following is an overview of their results and conclusions.

- Raibulet et al. [105] presented a comprehensive analysis of MDRE approaches in the literature from 2003 up to 2017. By conducting a systematic literature review, the authors aimed to answer three research questions concerning the metamodels used, the implemented tools, and the degree of automation in various MDRE approaches. The study highlighted 15 MDRE approaches and discussed their different features, such as automation, extensibility, and genericity. The authors provided some guidance concerning how to select an MDRE approach among the choices available.
- Lau and Arshad [106] classified the reverse engineering approaches within software product lines. The following

parameters acted as the basis for the classification: required input notation, generated output notation, analysis technique, required expertise, phase compatibility, and pre-requisite implementation [106]. They concluded that current reverse engineering approaches in software product lines mainly focused on variability management and feature locations.

- Assunção et al. [107] carried out a systematic mapping study (SMS) to present an overview of the current research on re-engineering existing systems to software product lines (SPLs). 119 relevant publications were analyzed and classified into six dimensions, which are: re-engineering phases, applied strategies, input artifacts, output artifacts, tool support, and types of systems involved in the evaluation [107]. The authors highlight open issues and areas that could be used as references for future research.
- Ghaleb et al. [108] conducted a systematic review to investigate the current reverse engineering approaches for program comprehension that aim to extract program behaviour and present it using sequence diagrams. They assessed these approaches by examining their features, limitations, and operations. The authors also suggested several enhancements to sequence diagrams to make them more clear and understandable, particularly for interactions between programs and various aspects of control flow.
- More recently, Luna-Herrera et al. [109] conducted a systematic mapping study (SMS) to identify reverse engineering approaches for understanding computer programs. Forty-eight studies were selected as primary studies, and ten different reverse engineering approaches for understanding computer programs were identified by analyzing the research's findings. Fifteen different types of artifacts were identified. The authors found that the most commonly used types of artifacts are visualization graphs, class diagrams, and sequence diagrams [109]. In addition, MDRE and graphical visualization are the most widely used approaches.

Unlike these aforementioned studies, our SLR stands out as it is fairly exhaustive in its investigation of model-driven reverse engineering, with partial coverage of model-driven re-engineering approaches. It contributes to providing an extensive investigation to answer different research questions concerning different facets of these approaches. In addition, it shows different program migration and program translation approaches that belong to the model-driven re-engineering context.

V. LIMITATIONS AND THREATS TO VALIDITY

In any systematic literature review, it is important to ensure the validity of the outcomes. This section outlines the steps taken to mitigate potential threats and minimize their impact. The discussion focuses on addressing four major categories of common validity threats: construct validity, internal validity, external validity, and conclusion validity [110], [111].

A. CONSTRUCT VALIDITY

The assessment of how well the operational measures being studied match the objectives and areas of investigation of the researchers is known as construct validity [110], [111].

- Threat to the inadequate definition of the concept: To address the potential threat of inadequately defining concepts, research questions were carefully formulated under the supervision of experts. Subsequently, search strings were developed, including all possible sets of keywords and synonyms identified in relevant articles corresponding to the research questions. To validate and enhance the effectiveness of the search strings, multiple iterative pilot searches were conducted. Also, the search strings were modified according to a specific query language used by each database engine. Moreover, inclusion and exclusion criteria were thoroughly checked and implemented to ensure the careful selection of primary studies.
- Threat to selection data sources: We carried out an SLR by performing automated searches across the most significant academic databases related to software engineering. These databases included *Scopus*, *IEEE Xplore* Digital Library, *Web of Science*, and the *ACM Digital Library*. In addition, we employed various queries on *Google Scholar* to enhance the comprehensiveness of our search and ensure the inclusion of relevant articles from diverse sources. To exhaustively cover papers related to reverse engineering techniques that abstract UML/OCL specifications from source code, forward snowballing was implemented (Fig. 3). Additionally, we manually incorporated primary articles recommended by experts, focusing on those addressing the use of machine learning techniques in the program translation process. This comprehensive approach resulted in the discovery of an extensive number of sources in our SLR, effectively mitigating threats related to the selection of data sources.

B. INTERNAL VALIDITY

Internal validity refers to threats that may have affected the results and other factors that were not adequately considered [110].

- Studies selection bias threat: Concerning the selection of primary studies, we detail in Section Search Method the steps taken to justify the systematic and unbiased selection. For example, if the number of search results exceeds 300, we chose the first 300, sorted by relevance, just like when we searched for papers in the *Google Scholar* database. Additionally, appropriate inclusion and exclusion criteria were identified and implemented. Each study undergoes screening and selection by the

first author, with co-authors involved in cases where the first author is uncertain about its relevance.

- Data extraction threat: Regarding the extraction of data from primary sources, we defined a data extraction form to guarantee reliable retrieval of relevant information and to verify its relevance to the research questions. Throughout the SLR process, the majority of data extraction was carried out by the first author. To mitigate possible bias or errors, we actively addressed any uncertainties collaboratively. Despite these efforts, a higher chance of bias remains when a single researcher is responsible for data extraction. To mitigate this, co-authors were engaged to conduct random checks and address any uncertainties at various stages of the SLR.
- Missing relevant studies threat: The search relied on metadata (abstract, title, and keyword), which may have missed studies that did not explicitly mention reverse engineering or re-engineering in these fields. While the metadata is provided by the paper authors, we must also consider the effectiveness of digital databases in classifying and indexing papers. The studies were gathered from academic indexing services, which are the primary data sources for our SLR.

C. EXTERNAL VALIDITY

The focus of external validity is on the degree to which the results of the study can be applied to different environments or cases [110], [111]. The content that is published in peer-reviewed journals and conferences might not accurately reflect what practitioners really use in the workplace, posing a challenge to generalizing the findings of the literature review. However, efforts were made in this study to address this threat by systematically and manually incorporating primary papers on both model-driven reverse engineering and model-driven re-engineering approaches. Additionally, our set of research questions is general enough to identify and classify these approaches. These questions are independent of any platform or specific programming language. Moreover, the statistical power threat did not appear, as the number of retrieved approaches (64) is high enough to generalize the results.

D. CONCLUSION VALIDITY

Issues that can impact a study's ability to get useful outcomes are referred to as threats to conclusion validity [110], [111]. Although these problems cannot be completely eliminated, they can be mitigated by answering the research questions and carefully analyzing the results of all primary studies involved in the research. In our study, the conclusion was reached after collecting and analyzing the results of each primary study, and this was done by both the first and second authors. The primary goal is to reduce the possibility of bias affecting SLR results.

VI. CONCLUSION

The number of software systems designed to fulfill user requirements is experiencing rapid growth. Consequently, the importance of using efficient MDRE and model-driven re-engineering approaches to understand and evolve those systems is crucial. Yet, the peer-reviewed scientific literature does not adequately cover the topics of MDRE and model-driven re-engineering. In our SLR, 83 publications were reviewed, and 64 MDRE and model-driven re-engineering approaches were assessed. The comprehensive review provides valuable insights for researchers and practitioners who are considering using MDRE and model-driven re-engineering to achieve different objectives, including program comprehension and program translation tasks. The review addresses the research questions as follows:

RQ1 *What are the existing model-driven reverse engineering approaches for software systems?* The question was answered by Tables 6 and 7, which list 64 different approaches (labeled A1 to A64) that were found by reviewing 83 studies. We identified that there appeared to be a peak of research activity in MDRE in the period 2011–2014 following the publication of the OMG's KDM [7] in 2011, and subsequently a reduced level of research activity comparable to the period 2006–2010.

SQ1.1 *What are the objectives of model-driven reverse engineering approaches?* Table 8 outlines the objectives of each approach, with the most common objectives being program comprehension/documentation, modernization, program migration and testing, maintenance, and program translation, listed in that order. The full comparison of the objectives is depicted in Fig. 6.

SQ1.2 *What are the general characteristics of model-driven reverse engineering approaches?* We found that object-oriented source languages, particularly Java, were the most frequent subjects of MDRE approaches. This is contrary to the focus of much mainstream re-engineering work on procedural code bases [112], [113], and is perhaps reflective of the bias towards Java in MDE research in general. We found that UML models, particularly class diagrams, were the most common target representations of MDRE, together with ASTs. The majority of MDRE approaches were fully automated and used static code analysis for reverse engineering. Most of the MDRE approaches were evaluated on real-world applications. The identified approaches use a wide range of tools, showing that there is not yet agreement on a particular optimal MDRE approach. A detailed explanation is available in the Characteristics of Model-Driven Reverse Engineering Approaches Section.

RQ2 *What are the current model-driven re-engineering approaches for software systems?* The answer to the question can be found by exploring the most popular re-engineering activities, which are program migration and program translation. There are ten

different approaches to program migration out of the 64 gathered, and their target platforms are listed in Table 12. Further details can be found in Program Migration Section. Regarding program translation, there are both machine and non-machine program translation approaches, and each approach has its unique characteristics. More details about these approaches are found in Sections Program Translation Approaches and Non-machine Learning Techniques and Program Translation and Machine Learning Techniques, respectively.

To address the challenges and limitations mentioned in this study, we suggest the following areas of investigation for future research:

- Enhancing the usability and scalability of MDRE approaches, particularly for systems with complex architectures, systems written in multiple programming languages, and systems that are frequently updated and changed.
- Because UML cannot represent all the necessary information for MDRE [105], it is important to supplement the extracted UML models with other models, such as OCL specifications.
- Using machine learning to support MDRE is a promising area of research, as there are very few approaches utilizing these techniques to support MDRE despite the growing use of ML approaches for other software engineering activities [114]. In particular, there is increasing use of ML in the field of program translation, as highlighted in Program Translation and Machine Learning Techniques Section.

Overall, we have observed several problems and challenges with MDRE, but we believe that the potential advantages of using machine learning techniques such as LLMs can help to address the identified disadvantages and limitations of MDRE. We expect that further research in the near future will therefore lead to innovative and effective MDRE approaches.

ACKNOWLEDGMENT

Hanan Abdulwahab Siala would like to thank the Libyan Ministry of Higher Education and Scientific Research for its financial support during her Ph.D. studies. Additionally, she would like to express her appreciation to her supervisors for their invaluable help and support. Finally, the authors would like to thank the anonymous reviewers whose comments improved this work.

REFERENCES

[1] S. Demeyer, S. Ducasse, and O. Nierstrasz, *Object-Oriented Reengineering Patterns*. Amsterdam, The Netherlands: Elsevier, 2002.

[2] E. J. Chikofsky and J. H. Cross, "Reverse engineering and design recovery: A taxonomy," *IEEE Softw.*, vol. 7, no. 1, pp. 13–17, Jan. 1990.

[3] S. Kent, "Model driven engineering," in *Proc. Int. Conf. Integr. Formal Methods (IFM)*, Turku, Finland. Berlin, Germany: Springer, May 2002, pp. 286–298.

[4] A. Van Deursen, E. Visser, and J. Warmer, "Model-driven software evolution: A research agenda," MoDSE Nantes, France, Tech. Rep. Ser. TUD-SERG-2007-006, 2007.

[5] P. Mohagheghi, W. Gilani, A. Stefanescu, M. A. Fernandez, B. Nordmoen, and M. Fritzsche, "Where does model-driven engineering help? Experiences from three industrial cases," *Softw. Syst. Model.*, vol. 12, no. 3, pp. 619–639, Jul. 2013.

[6] Object Manage. Group. (2017). *OMG Systems Modeling Language (UML), Version 2.5.1*. [Online]. Available: https://www.omg.org/spec/UML

[7] R. Pérez-Castillo, I. G.-R. de Guzmán, and M. Piattini, "Knowledge discovery metamodel-ISO/IEC 19506: A standard to modernize legacy systems," *Comput. Standards Interfaces*, vol. 33, no. 6, pp. 519–532, Nov. 2011.

[8] Object Manage. Group. (2014). *Object Constraint Language 2.4 Specification, OMG Document Formal*. [Online]. Available: https://www.omg.org/spec/OCL/2.4/About-OCL

[9] K. Lano, H. Haughton, Z. Yuan, and H. Alfraihi, "Program abstraction and re-engineering: An agile MDE approach," in *Proc. ACM/IEEE Int. Conf. Model Driven Eng. Lang. Syst. Companion (MODELS-C)*, Oct. 2023, pp. 211–220.

[10] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," School Comput. Sci. Math., Keele Univ. Durham Univ. Joint Rep., Keele, U.K., Tech. Rep. EBSE 2007-001, 2007.

[11] M. Petticrew and H. Roberts, *Systematic Reviews in the Social Sciences: A Practical Guide*. Hoboken, NJ, USA: Wiley, 2008.

[12] H. Brunelière, J. Cabot, G. Dupé, and F. Madiot, "MoDisco: A model driven reverse engineering framework," *Inf. Softw. Technol.*, vol. 56, no. 8, pp. 1012–1032, Aug. 2014.

[13] G. Barbier, H. Bruneliere, F. Jouault, Y. Lennon, and F. Madiot, "MoDisco, a model-driven platform to support real legacy modernization use cases," in *Information Systems Transformation*. Amsterdam, The Netherlands: Elsevier, 2010, pp. 365–400.

[14] L. Favre, "Formalizing MDA-based reverse engineering processes," in *Proc. 6th Int. Conf. Softw. Eng. Res., Manage. Appl.*, Aug. 2008, pp. 153–160.

[15] P. Claudia, M. Liliana, and F. Liliana, "Recovering use case diagrams from object oriented code: An MDA-based approach," in *Proc. 8th Int. Conf. Inf. Technol. New Generat.*, Apr. 2011, pp. 737–742.

[16] L. Martinez, C. Pereira, and L. Favre, "Recovering sequence diagrams from object-oriented code: An ADM approach," in *Proc. 9th Int. Conf. Eval. Novel Approaches Softw. Eng. (ENASE)*, Apr. 2014, pp. 1–8.

[17] S. Warwas and M. Klusch, "Making multiagent system designs reusable: A model-driven approach," in *Proc. IEEE/WIC/ACM Int. Conf. Web Intell. Intell. Agent Technol.*, vol. 2, Aug. 2011, pp. 101–108.

[18] F. Trias, V. de Castro, M. López-Sanz, and E. Marcos, "An ADM-based method for migrating CMS-based web applications: Extracting ASTM models from PHP code," in *Proc. Int. Workshop Softw. Evol. Modernization*, vol. 2. Angers, France: Scitepress, Jul. 2013, pp. 85–92.

[19] F. Trias, V. de Castro, M. Lopez-Sanz, and E. Marcos, "Migrating traditional web applications to CMS-based web applications," *Electron. Notes Theor. Comput. Sci.*, vol. 314, pp. 23–44, Jun. 2015.

[20] F. Trias, V. de Castro, M. López-Sanz, and E. Marcos, "A toolkit for ADM-based migration: Moving from PHP code to KDM model in the context of CMS-based web applications," in *Proc. 23rd Int. Conf. Inf. Syst. Develop. (ISD)*. Varaždin, Croatia: Association for Information Systems, Sep. 2014, p. 210. [Online]. Available: http://aisel.aisnet.org/isd2014/proceedings/WebBasedSystems/3

[21] F. Trias, V. de Castro, M. López-Sanz, and E. Marcos, "RE-CMS: A reverse engineering toolkit for the migration to CMS-based web applications," in *Proc. 30th Annu. ACM Symp. Appl. Comput.*, Apr. 2015, pp. 810–812.

[22] R. Pérez-Castillo, M. Fernández-Ropero, I. G. D. Guzmán, and M. Piattini, "MARBLE. A business process archeology tool," in *Proc. 27th IEEE Int. Conf. Softw. Maintenance (ICSM)*, Sep. 2011, pp. 578–581.

[23] A. Bergmayr, H. Bruneliere, J. Cabot, J. Garcia, T. Mayerhofer, and M. Wimmer, "FREX: FUML-based reverse engineering of executable behavior for software dynamic analysis," in *Proc. IEEE/ACM 8th Int. Workshop Model. Softw. Eng. (MiSE)*, May 2016, pp. 20–26.

[24] F. Fleurey, E. Breton, B. Baudry, A. Nicolas, and J.-M. Jézéquel, "Model-driven engineering for software migration in a large industrial context," in *Proc. 10th Int. Conf. Model Driven Eng. Lang. Syst. (MoDELS)*. Nashville, TN, USA: Springer, Sep./Oct. 2007, pp. 482–497.

[25] R. Couto, A. N. Ribeiro, and J. C. Campos, "A patterns based reverse engineering approach for Java source code," in *Proc. 35th Annu. IEEE Softw. Eng. Workshop*, Oct. 2012, pp. 140–147.
[26] F. Barbier, S. Eveillard, K. Youbi, O. Guitton, A. Perrier, and E. Cariou, "Model-driven reverse engineering of COBOL-based applications," in *Information Systems Transformation*. Amsterdam, The Netherlands: Elsevier, 2010, pp. 283–299.
[27] R. Rodríguez-Echeverría, J. M. Conejero, P. J. Clemente, J. C. Preciado, and F. Sánchez-Figueroa, "Modernization of legacy web applications into rich Internet applications," in *Proc. Current Trends Web Eng. Workshops Doctoral Symp., Tuts. (ICWE)*. Paphos, Cyprus: Springer, Jun. 2011, pp. 236–250.
[28] R. Rodríguez-Echeverría, J. M. Conejero, P. J. Clemente, M. D. Villalobos, and F. Sánchez-Figueroa, "Generation of WebML hypertext models from legacy web applications," in *Proc. 14th IEEE Int. Symp. Web Syst. Evol. (WSE)*, Sep. 2012, pp. 91–95.
[29] R. Rodríguez-Echeverría, V. M. Pavón, F. Macías, J. M. Conejero, P. J. Clemente, and F. Sánchez-Figueroa, "Generating a conceptual representation of a legacy web application," in *Web Information Systems Engineering—WISE*. Berlin, Germany: Springer, Oct. 2013, pp. 231–240. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-41154-0_17#citeas
[30] E. Sosa, P. J. Clemente, J. M. Conejero, and R. Rodríguez-Echeverría, "A model-driven process to modernize legacy web applications based on service oriented architectures," in *Proc. 15th IEEE Int. Symp. Web Syst. Evol. (WSE)*, Sep. 2013, pp. 61–70.
[31] Ó. S. Ramón, J. S. Cuadrado, and J. G. Molina, "Model-driven reverse engineering of legacy graphical user interfaces," *Automated Softw. Eng.*, vol. 21, no. 2, pp. 147–186, Apr. 2014.
[32] Ó. S. Ramon, J. S. Cuadrado, and J. G. Molina, "Reverse engineering of event handlers of RAD-based applications," in *Proc. 18th Work. Conf. Reverse Eng.*, Oct. 2011, pp. 293–302.
[33] Ó. S. Ramón, J. Vanderdonckt, and J. G. Molina, "Re-engineering graphical user interfaces from their resource files with UsiResourcer," in *Proc. IEEE 7th Int. Conf. Res. Challenges Inf. Sci. (RCIS)*, May 2013, pp. 1–12.
[34] U. Sabir, F. Azam, S. U. Haq, M. W. Anwar, W. H. Butt, and A. Amjad, "A model driven reverse engineering framework for generating high level UML models from Java source code," *IEEE Access*, vol. 7, pp. 158931–158950, 2019.
[35] C. Kotekar, "An approach to translate legacy 'C' code to Simulink® model using XML," in *Proc. Symp. Theory Model. Simulation DEVS Integrative M&S Symp.*, 2012, pp. 1–5.
[36] A. Grosche, B. Igel, and O. Spinczyk, "Transformation- and pattern-based state machine mining from embedded C code," in *Proc. 15th Int. Conf. Softw. Technol. (ICSOFT)* SciTePress, 2020, pp. 104–115.
[37] B. Verhaeghe, N. Anquetil, A. Etien, S. Ducasse, A. Seriai, and M. Derras, "GUI visual aspect migration: A framework agnostic solution," *Automated Softw. Eng.*, vol. 28, no. 2, p. 6, Nov. 2021.
[38] M. Ichii, T. Myojin, Y. Nakagawa, M. Chikahisa, and H. Ogawa, "A rule-based automated approach for extracting models from source code," in *Proc. 19th Work. Conf. Reverse Eng.*, Oct. 2012, pp. 308–317.
[39] V. Cosentino, J. Cabot, P. Albert, P. Bauquel, and J. Perronnet, "A model driven reverse engineering framework for extracting business rules out of a Java application," in *Proc. 6th Int. Symp. Rules Web Res. Appl. (RuleML)*, Montpellier, France. Berlin, Germany: Springer, Aug. 2012, pp. 17–31.
[40] V. Cosentino, J. Cabot, P. Albert, P. Bauquel, and J. Perronnet, "Extracting business rules from COBOL: A model-based framework," in *Proc. 20th Work. Conf. Reverse Eng. (WCRE)*, Oct. 2013, pp. 409–416.
[41] V. Cosentino, J. Cabot, P. Albert, P. Bauquel, and J. Perronnet, "Extracting business rules from COBOL: A model-based tool," in *Proc. 20th Work. Conf. Reverse Eng. (WCRE)*, Oct. 2013, pp. 483–484.
[42] A. Moutaouakkil and S. Mbarki, "PHP modernization approach generating KDM models from PHP legacy code," *Bull. Electr. Eng. Informat.*, vol. 9, no. 1, pp. 247–255, Feb. 2020.
[43] F. Améndola and L. Favre, "Adapting CRM systems for mobile platforms: An MDA perspective," in *Proc. 14th ACIS Int. Conf. Softw. Eng., Artif. Intell., Netw. Parallel/Distrib. Comput.*, Jul. 2013, pp. 323–328.
[44] M. A. Garzón, H. Aljamaan, and T. C. Lethbridge, "Umple: A framework for model driven development of object-oriented systems," in *Proc. IEEE 22nd Int. Conf. Softw. Anal., Evol., Reeng. (SANER)*, Mar. 2015, pp. 494–498.
[45] T. C. Lethbridge, A. Forward, and O. Badreddin, "Umplification: Refactoring to incrementally add abstraction to a program," in *Proc. 17th Work. Conf. Reverse Eng.*, Oct. 2010, pp. 220–224.
[46] J. C. Silva, J. Saraiva, and J. C. Campos, "A generic library for GUI reasoning and testing," in *Proc. ACM Symp. Appl. Comput.*, Mar. 2009, pp. 121–128.
[47] J. C. Silva, J. C. Campos, and J. Saraiva, "Combining formal methods and functional strategies regarding the reverse engineering of interactive applications," in *Proc. 13th Int. Workshop Interact. Systems. Design, Specification, Verification (DSVIS)* (Lecture Notes in Computer Science), vol. 4323. Dublin, Ireland: Springer, Jul. 2006, pp. 137–150. [Online]. Available: https://doi.org/10.1007/978-3-540-69554-7_11
[48] J. C. Silva, C. Silva, R. D. Gonçalo, J. Saraiva, and J. C. Campos, "The GUISurfer tool: Towards a language independent approach to reverse engineering GUI code," in *Proc. 2nd ACM SIGCHI Symp. Eng. Interact. Comput. Syst.*, Jun. 2010, pp. 181–186.
[49] J. C. Campos, J. Saraiva, C. Silva, and J. C. Silva, "GUIsurfer: A reverse engineering framework for user interface software," in *Reverse Engineering-Recent Advances and Applications*. Rijeka, Croatia: IntechOpen, 2012, ch. 2, pp. 31–54, doi: 10.5772/32931.
[50] X. Bai and T. Liu, "SyncTest: A tool to synchronize source code, model and testing," in *Proc. SEKE*, 2008, pp. 723–728.
[51] Q. Li, S. Hu, P. Chen, L. Wu, and W. Chen, "Discovering and mining use case model in reverse engineering," in *Proc. 4th Int. Conf. Fuzzy Syst. Knowl. Discovery (FSKD)*, vol. 4, Aug. 2007, pp. 431–436.
[52] T. Katsimpa, Y. Panagis, E. Sakkopoulos, G. Tzimas, and A. Tsakalidis, "Application modeling using reverse engineering techniques," in *Proc. ACM Symp. Appl. Comput.*, Apr. 2006, pp. 1250–1255.
[53] H. Zhang, "An approach for extracting UML diagram from object-oriented program based on J2X," in *Proc. Int. Forum Mech., Control Autom. (IFMCA)*. Atlantis Press, 2017, pp. 266–276. [Online]. Available: https://www.atlantis-press.com/proceedings/ifmca-16/25871847, doi: 10.2991/ifmca-16.2017.43.
[54] C. Rodríguez, K. Garcés, J. Cabot, R. Casallas, F. Melo, D. Escobar, and A. Salamanca, "Model-based assisted migration of Oracle forms applications: The overall process in an industrial setting," *Softw., Pract. Exper.*, vol. 51, no. 8, pp. 1641–1675, Aug. 2021.
[55] O. El Beggar, B. Bousetta, and T. Gadi, "Getting objects methods and interactions by extracting business rules from legacy systems," *J. Syst. Integr.*, vol. 5, no. 3, pp. 32–56, 2014.
[56] R. Mzid, A. Charfi, and N. Etteyeb, "Use of compiler intermediate representation for reverse engineering: A case study for GCC compiler and UML activity diagram," in *Proc. 10th Int. Conf. Model-Driven Eng. Softw. Develop.*, 2022, pp. 211–218.
[57] A. Alshanqiti, R. Heckel, and T. Kehrer, "Visual contract extractor: A tool for reverse engineering visual contracts using dynamic analysis," in *Proc. 31st IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Sep. 2016, pp. 816–821.
[58] M. Bork, L. Geiger, C. Schneider, and A. Zündorf, "Towards roundtrip engineering—A template-based reverse engineering approach," in *Proc. 4th Eur. Conf. Model Driven Archit. Found. Appl. (ECMDA-FA)*. Berlin, Germany: Springer, Jun. 2008, pp. 33–47.
[59] T. B. la Fosse, M. Tisi, and J.-M. Mottu, "Injecting execution traces into a model-driven framework for program analysis," in *Proc. Softw. Technol. Appl. Found. Collocated Workshops (STAF)*. Marburg, Germany: Springer, Jul. 2017, pp. 3–13.
[60] C. Wagner, T. Margaria, and H.-G. Pagendarm, "Analysis and code model extraction for C/C++ source code," in *Proc. 14th IEEE Int. Conf. Eng. Complex Comput. Syst.*, Jun. 2009, pp. 110–119.
[61] P. Aho, M. Suarez, T. Kanstrén, and A. M. Memon, "Industrial adoption of automatically extracted gui models for testing," in *Proc. EESSMod@MoDELS*, 2013, pp. 49–54.
[62] C. Sacramento and A. C. R. Paiva, "Web application model generation through reverse engineering and UI pattern inferring," in *Proc. 9th Int. Conf. Quality Inf. Commun. Technol.*, Sep. 2014, pp. 105–115.
[63] T. Kappler, H. Koziolek, K. Krogmann, and R. Reussner, "Towards automatic construction of reusable prediction models for component-based performance engineering," in *Software Engineering 2008*. Bonn, Germany: Gesellschaft für Informatik e.V., 2008, pp. 140–154.
[64] I. C. Morgado, A. Paiva, and J. P. Faria, "Reverse engineering of graphical user interfaces," in *Proc. 6th Int. Conf. Softw. Eng. Adv.*, Barcelona, Spain, 2011, pp. 293–298.

[65] E. Shah and E. Tilevich, "Reverse-engineering user interfaces to facilitateporting to and across mobile devices and platforms," in *Proc. Compilation Workshops DSM, TMC, AGERE!, AOOPES, NEAT, VMIL*, Oct. 2011, pp. 255–260.

[66] K. Lamhaddab, M. Lachgar, and K. Elbaamrani, "Porting mobile apps from iOS to Android: A practical experience," *Mobile Inf. Syst.*, vol. 2019, pp. 1–29, Sep. 2019.

[67] L. C. Briand, Y. Labiche, and J. Leduc, "Toward the reverse engineering of UML sequence diagrams for distributed Java software," *IEEE Trans. Softw. Eng.*, vol. 32, no. 9, pp. 642–663, Sep. 2006.

[68] I. A. Salihu, R. Ibrahim, and A. Mustapha, "A hybrid approach for reverse engineering GUI model from Android apps for automated testing," *J. Telecommun., Electron. Comput. Eng. (JTEC)*, vol. 9, no. 3, pp. 45–49, 2017.

[69] M. L. Bernardi, M. Cimitile, and G. Di Lucca, "Design pattern detection using a DSL-driven graph matching approach," *J. Softw., Evol. Process*, vol. 26, no. 12, pp. 1233–1266, Dec. 2014.

[70] Z. Gotti and S. Mbarki, "GUI structure and behavior from Java source code analysis," in *Proc. 4th IEEE Int. Colloq. Inf. Sci. Technol. (CiSt)*, Oct. 2016, pp. 251–256.

[71] T. Reus, H. Geers, and A. van Deursen, "Harvesting software systems for MDA-based reengineering," in *Proc. 2nd Eur. Conf. Model Driven Archit. Found. Appl. (ECMDA-FA)*. Bilbao, Spain: Springer, Jul. 2006, pp. 213–225.

[72] F. Heidenreich, J. Johannes, J. Reimann, M. Seifert, C. Wende, C. Werner, C. Wilke, and U. Assmann, "Model-driven modernisation of Java programs with JaMoPP," in *Proc. 1st Int. Workshop Model-Driven Softw. Migration (MDSM), 5th Int. Workshop Syst. Quality Maintainability (SQM)*, 2011, pp. 8–11.

[73] C. J. F. Candel, J. G. Molina, F. J. B. Ruiz, J. R. H. Barceló, D. S. Ruiz, and B. J. C. Viera, "Developing a model-driven reengineering approach for migrating PL/SQL triggers to Java: A practical experience," *J. Syst. Softw.*, vol. 151, pp. 38–64, May 2019.

[74] N. Methakullawat and Y. Limpiyakorn, "Reengineering legacy code with model transformation," *Int. J. Softw. Eng. Appl.*, vol. 8, no. 3, pp. 97–110, 2014.

[75] K. Sabiri, F. Benabbou, H. Moutachaouik, and M. Hain, "Towards a cloud migration framework," in *Proc. 3rd World Conf. Complex Syst. (WCCS)*, Nov. 2015, pp. 1–6.

[76] A. Fuhr, A. Winter, U. Erdmenger, T. Horn, U. Kaiser, V. Riediger, and W. Teppe, "Model-driven software migration: Process model, tool support, and application," in *Migrating Legacy Applications: Challenges in Service Oriented Architecture and Cloud Computing Environments*. IGI Globa, 2013, pp. 153–184. [Online]. Available: https://www.igi-global.com/chapter/model-driven-software-migration/72216, doi: 10.4018/978-1-4666-2488-7.ch007.

[77] A. Bergmayr, H. Brunelière, J. L. C. Izquierdo, J. Gorroñogoitia, G. Kousiouris, D. Kyriazis, P. Langer, A. Menychtas, L. Orue-Echevarria, C. Pezuela, and M. Wimmer, "Migrating legacy software to the cloud with ARTIST," in *Proc. 17th Eur. Conf. Softw. Maintenance Reeng.*, Mar. 2013, pp. 465–468.

[78] A. Menychtas, C. Santzaridou, G. Kousiouris, T. Varvarigou, L. Orue-Echevarria, J. Alonso, J. Gorronogoitia, H. Bruneliere, O. Strauss, T. Senkova, B. Pellens, and P. Stuer, "ARTIST methodology and framework: A novel approach for the migration of legacy software on the cloud," in *Proc. 15th Int. Symp. Symbolic Numeric Algorithms Sci. Comput.*, Sep. 2013, pp. 424–431.

[79] H. Marah, G. Kardas, and M. Challenger, "Model-driven round-trip engineering for TinyOS-based WSN applications," *J. Comput. Lang.*, vol. 65, Aug. 2021, Art. no. 101051.

[80] Z. Yang, Z. Qiu, Y. Zhou, Z. Huang, J.-P. Bodeveix, and M. Filali, "C2AADL_Reverse: A model-driven reverse engineering approach to development and verification of safety-critical software," *J. Syst. Archit.*, vol. 118, Sep. 2021, Art. no. 102202.

[81] R. Pérez-Castillo, L. Jiménez-Navajas, and M. Piattini, "QRev: Migrating quantum code towards hybrid information systems," *Softw. Quality J.*, vol. 30, no. 2, pp. 551–580, Jun. 2022.

[82] K. Lano, "Program translation using model-driven engineering," in *Proc. IEEE/ACM 44th Int. Conf. Softw. Eng. Companion (ICSE-Companion)*, May 2022, pp. 362–363.

[83] A. Yaseen and H. Fawareh, "Visualization and synchronization of object-oriented programs using re-engineering approach," *Int. J. Eng. Technol.*, vol. 12, no. 8, pp. 1239–1246, Jan. 2019.

[84] H. Cheers and Y. Lin, "Reverse engineering UML sequence diagrams for program comprehension activities," in *Proc. 5th Int. Conf. Innov. Technol. Intell. Syst. Ind. Appl. (CITISIA)*, Nov. 2020, pp. 1–10.

[85] T. Ziadi, M. A. A. d. Silva, L. M. Hillah, and M. Ziane, "A fully dynamic approach to the reverse engineering of UML sequence diagrams," in *Proc. 16th IEEE Int. Conf. Eng. Complex Comput. Syst.*, Apr. 2011, pp. 107–116.

[86] E. Fauzi, B. Hendradjaya, and W. D. Sunindyo, "Reverse engineering of source code to sequence diagram using abstract syntax tree," in *Proc. Int. Conf. Data Softw. Eng. (ICoDSE)*, Oct. 2016, pp. 1–6.

[87] E. M. Bouziane, C. Baidada, and A. Jakimi, "Towards a dynamic analysis of legacy systems for reverse-engineering interaction diagrams," *J. Theor. Appl. Inf. Technol.*, vol. 98, no. 12, pp. 2174–2184, 2020.

[88] C. Baidada, E. M. Bouziane, and A. Jakimi, "A new approach for recovering high-level sequence diagrams from object-oriented applications using Petri nets," *Proc. Comput. Sci.*, vol. 148, pp. 323–332, Jan. 2019.

[89] Y. Labiche, B. Kolbah, and H. Mehrfard, "Combining static and dynamic analyses to reverse-engineer scenario diagrams," in *Proc. IEEE Int. Conf. Softw. Maintenance*, Sep. 2013, pp. 130–139.

[90] A. Nanthaamornphong and A. Leatongkam, "Extended ForUML for automatic generation of UML sequence diagrams from object-oriented Fortran," *Sci. Program.*, vol. 2019, pp. 1–22, Feb. 2019.

[91] A. Nanthaamornphong, K. Morris, and S. Filippone, "Extracting UML class diagrams from object-oriented Fortran: ForUML," in *Proc. 1st Int. Workshop Softw. Eng. High Perform. Comput. Comput. Sci. Eng.*, 2013, pp. 9–16.

[92] R. G. Alsarraj, A. M. Altaie, and A. A. Fadhil, "Designing and implementing a tool to transform source code to UML diagrams," *Periodicals Eng. Natural Sci. (PEN)*, vol. 9, no. 2, pp. 430–440, Mar. 2021.

[93] E. Korshunova, M. Petkovic, M. G. J. van den Brand, and M. R. Mousavi, "CPP2XMI: Reverse engineering of UML class, sequence, and activity diagrams from C++ source code," in *Proc. 13th Work. Conf. Reverse Eng.*, Oct. 2006, pp. 297–298.

[94] B. Cornelissen, A. Zaidman, A. van Deursen, L. Moonen, and R. Koschke, "A systematic survey of program comprehension through dynamic analysis," *IEEE Trans. Softw. Eng.*, vol. 35, no. 5, pp. 684–702, Oct. 2009.

[95] G. Deltombe, O. L. Goaer, and F. Barbier, "Bridging KDM and ASTM for model-driven software modernization," in *Proc. SEKE*, 2012, pp. 517–524.

[96] L. Favre, "MDA-based reverse engineering," in *Reverse Engineering—Recent Advances and Applications*. Rijeka, Croatia: IntechOpen, 2012, ch. 3, doi: 10.5772/32473.

[97] Eclipse EMF. (2023). *Eclipse EMF—Eclipse Modeling Framework*. Accessed: Oct. 24, 2023. [Online]. Available: http://www.eclipse.org/emf/

[98] Eclipse MoDisco. (2023). *Eclipse MoDisco—Model Discovery*. Accessed: Oct. 24, 2023. [Online]. Available: http://www.eclipse.org/Modisco

[99] D. Zhang and J. J. Tsai, *Machine Learning Applications in Software Engineering*, vol. 16. Singapore: World Scientific, 2005.

[100] B. Roziere, M.-A. Lachaux, L. Chanussot, and G. Lample, "Unsupervised translation of programming languages," in *Proc. 34th Int. Conf. Neural Inf. Process. Syst. (NIPS)*. Red Hook, NY, USA: Curran Associates, 2020, pp. 20601–20611, Art. no. 1730.

[101] M. Zhu, K. Suresh, and C. K. Reddy, "Multilingual code snippets training for program translation," in *Proc. AAAI Conf. Artif. Intell.*, 2022, pp. 11783–11790.

[102] W. U. Ahmad, S. Chakraborty, B. Ray, and K.-W. Chang, "Summarize and generate to back-translate: Unsupervised translation of programming languages," in *Proc. 17th Conf. Eur. Chapter Assoc. Comput. Linguistics*. Dubrovnik, Croatia: Association for Computational Linguistics, May 2023, pp. 1528–1542. [Online]. Available: https://aclanthology.org/2023.eacl-main.112

[103] I. J. Rithy, H. H. Shakil, N. Mondal, F. Sultana, and F. M. Shah, "XTest: A parallel multilingual corpus with test cases for code translation and its evaluation*," in *Proc. 25th Int. Conf. Comput. Inf. Technol. (ICCIT)*, Dec. 2022, pp. 623–628.

[104] X. Chen, C. Liu, and D. Song, "Tree-to-tree neural networks for program translation," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.* Red Hook, NY, USA: Curran Associates, 2018, pp. 2552–2562.

[105] C. Raibulet, F. A. Fontana, and M. Zanoni, "Model-driven reverse engineering approaches: A systematic literature review," *IEEE Access*, vol. 5, pp. 14516–14542, 2017.

[106] K.-K. Lau and R. Arshad, "A concise classification of reverse engineering approaches for software product lines," in *Proc. 11th Int. Conf. Softw. Eng. Adv. (ICSEA)*, 2016, pp. 31–38.
[107] W. K. G. Assunção, R. E. Lopez-Herrejon, L. Linsbauer, S. R. Vergilio, and A. Egyed, "Reengineering legacy applications into software product lines: A systematic mapping," *Empirical Softw. Eng.*, vol. 22, no. 6, pp. 2972–3016, Dec. 2017.
[108] T. A. Ghaleb, M. A. Alturki, and K. Aljasser, "Program comprehension through reverse-engineered sequence diagrams: A systematic review," *J. Softw., Evol. Process*, vol. 30, no. 11, p. e1965, Nov. 2018.
[109] Y. A. Luna-Herrera, J. C. Pérez-Arriaga, J. O. Ocharán-Hernández, and Á. J. Sanchéz-García, "Comprehension of computer programs through reverse engineering approaches and techniques: A systematic mapping study," in *Proc. Int. Conf. Softw. Process Improvement*. Springer, 2022, pp. 126–140.
[110] R. Feldt and A. Magazinius, "Validity threats in empirical software engineering research—An initial survey," in *Proc. SEKE*, 2010, pp. 374–379.
[111] A. Ampatzoglou, S. Bibi, P. Avgeriou, M. Verbeek, and A. Chatzigeorgiou, "Identifying, categorizing and mitigating threats to validity in software engineering secondary studies," *Inf. Softw. Technol.*, vol. 106, pp. 201–230, Feb. 2019.
[112] A. De Marco, V. Iancu, and I. Asinofsky, "COBOL to Java and newspapers still get delivered," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Sep. 2018, pp. 583–586.
[113] H. Sneed, "Migrating from COBOL to Java: A report from the field," in *Proc. IEEE 26th ICSM*. Timişoara, Romania: IEEE Press, Sep. 2011, pp. 1–7.
[114] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang, "Large language models for software engineering: A systematic literature review," 2023, *arXiv:2308.10620*.

HANAN ABDULWAHAB SIALA received the M.Sc. degree in computer science from Libyan Academy, Libya. She is currently pursuing the Ph.D. degree in computer science with King's College London, U.K. Her thesis focused on extracting UML and OCL specifications from source code using LLMs. She was a Lecturer with Tripoli University, Libya. She contributed to the design and development of complex information systems, including systems for the social security fund within the public administration and municipalities and systems for all educational institutions in Libya that serve students, employees, and teachers. Her research interests include model-driven development, unified modeling language, and software re-engineering.

KEVIN LANO contributed to the foundation of the model-driven engineering (MDE) field in the 1990s via his work in reverse engineering, formal methods, and methods integration. He has initiated novel software engineering approaches, such as constraint-driven development (CDD) and the synthesis of code generators from examples (CGBE). He has been active in improving the precision of UML and in applying software engineering principles to transformation construction. His major contribution has been to provide comprehensive AgileUML tools for MDE. His recent work has focused on the development of simplified and agile processes for MDE, re-engineering using MDE, and the application of MDE in finance.

HESSA ALFRAIHI is an Assistant Professor in the Department of Information Systems, College of Computer and Information Sciences, Princess Nourah bint Abdulrahman University, Saudi Arabia. She has completed her Ph.D. in Software Engineering at Kings College London in 2020. At present, Dr. Alfraihi's research endeavors are centered around cutting-edge areas of software engineering, with a primary focus on model-driven software engineering, agile development, and machine learning. Her work in these domains demonstrates her visionary approach to tackling complex software development challenges through innovative methods and tools.