Information and Software Technology 140 (2021) 106693

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Developing Mobile Applications Via Model Driven Development: A Systematic Literature Review

Md. Shamsujjoha [a,*], John Grundy [a], Li Li [a], Hourieh Khalajzadeh [a], Qinghua Lu [b]

[a] Department of Software Systems and Cybersecurity, Faculty of Information Technology, Monash University, Melbourne, Australia
[b] Data61, CSIRO, Sydney, Australia

ARTICLE INFO

Keywords:
Systematic Literature Review
Model Driven Development
Mobile App
Tools and Techniques

ABSTRACT

Context: Mobile applications (known as "apps") usage continues to rapidly increase, with many new apps being developed and deployed. However, developing a mobile app is challenging due to its dependencies on devices, technologies, platforms, and deadlines to reach the market. One potential approach is to use **M**odel **D**riven **D**evelopment (MDD) techniques that simplify the app development process, reduce complexity, increase abstraction level, help achieve scalable solutions and maximize cost-effectiveness and productivity.
Objective: This paper systematically investigates what MDD techniques and methodologies have been used to date to support mobile app development and how these techniques have been employed, to identify key benefits, limitations, gaps and future research potential.
Method: A Systematic Literature Review approach was used for this study based on a formal protocol. The rigorous search protocol identified a total of 1,042 peer-reviewed academic research papers from four major software engineering databases. These papers were subsequently filtered, and 55 high quality relevant studies were selected for analysis, synthesis, and reporting.
Results: We identified the popularity of different applied MDD approaches, supporting tools, artifacts, and evaluation techniques. Our analysis found that architecture, domain model, and code generation are the most crucial purposes in MDD-based app development. Three qualities – productivity, scalability and reliability – can benefit from these modeling strategies. We then summarize the key collective strengths, limitations, gaps from the studies and made several future recommendations.
Conclusion: There has been a steady interest in MDD approaches applied to mobile app development over the years. This paper guides future researchers, developers, and stakeholders to improve app development techniques, ultimately that will help end-users in having more effective apps, especially when some recommendations are addressed, e.g., taking into account more human-centric aspects in app development.

1. Introduction

In 2021, the number of smartphone users exceeded 3.8 billion, and approximately 66% of the world population had a mobile device such as cell phone, tablet, or a cellular-enabled IoT device [1,2]. In addition, mobile phone usage for different purposes had an average increase of 7.71% per year over the last three years [3]. In 2018, more than 205 billion mobile apps were downloaded from the app repositories [4]. The revenue earned by the mobile apps is expected to reach $935 billion in 2023, compared to $365 billion earned in 2018 [5]. Developing a modern mobile app is not a trivial exercise [6]. Key steps in app development include requirements gathering, platform selection, target users identification, constraint mapping, and problem modeling. During design, app developers draw approximate **U**ser **I**nterface (UI) sketches and may use a prototyping tool to create the model aspects of the visual design and the navigation flow. The architecture of the app is designed based on needed functionality and user interface mockups. Finally, the app is coded and the clients download, use and provide feedback on it. This is a highly iterative process and continues throughout the development lifecycle.

Model **D**riven **D**evelopment (MDD) techniques can help developers build an app more efficiently, as they enable code synthesis through a model transformation process. MDD has shown to be successful in many **S**oftware **E**ngineering (SE) domains to improve productivity, increase the quality of the outcome, provide tools for formal analysis, minimize manual implementation effort, provide more reliability, flexibility, and

* Corresponding author.
E-mail addresses: md.shamsujjoha@monash.edu, dishacse@yahoo.com (M. Shamsujjoha), john.grundy@monash.edu (J. Grundy), li.li@monash.edu (L. Li), hourieh.khalajzadeh@monash.edu (H. Khalajzadeh), qinghua.lu@data61.csiro.au (Q. Lu).

https://doi.org/10.1016/j.infsof.2021.106693
Received 27 March 2021; Received in revised form 14 July 2021; Accepted 19 July 2021
Available online 31 July 2021
0950-5849/

easy maintenance [7,8]. However, some researchers have shown that while many MDD-based mobile app development approaches offer useful domain patterns and tools, many of the models they use are relatively low-level. This can be a problem because they can then become very large and cumbersome to work with, require a lot of modeling effort, and do not abstract away from code-level details. Additionally, while many are sufficient to describe information for basic app generation, often only user interface or basic data aspects can be modeled and generated, and they do not contain adequate information to realize the full implementation while still require very detailed modeling [9,10]. Thus, these approaches are often hard to reuse, need expert app developer input, do not leverage similarities across platforms, and require extensive post-generation app testing and tuning.

We conducted a detailed investigation into existing research approaches used for mobile app development based on model-driven development, to identify strengths, limitations, and key directions for future work in the area. To do this, we chose to use a **S**ystematic **L**iterature **R**eview (SLR), locating and synthesizing relevant academic literature. This SLR will benefit the readers interested in app development in three key ways: (i) to understand existing MDD methodologies, techniques, and tools for app development; (ii) to compare different ways of app modeling and generation; and (iii) to identify key research gaps, potential future work, and enhancement possibilities for existing MDD based mobile app development approaches.

Based on our findings, 20 out of 55 selected studies aimed to make the app development process more flexible and faster, primarily to manage models, data, and services. We also found that the reusable code generation components in 8 of the selected studies was shown to reduce product development time and consequently costs for cross-platform, multi-platform or multi-version app development. Additionally, 18 studies proposed a new method, framework, tool, or languages to increase development efficiency. Our analysis also found that architecture, domain model and code generation are the most crucial purposes in MDD based app development, and three qualities, productivity, scalability and reliability can benefit from these modeling strategies. We found that a substantial proportion of the selected studies (25.45%) focus mainly on interface development instead of full mobile app features. Most approaches have been applied to examples and use-cases from academia (80%) and only a few from industry. One potential reason is that MDD approaches are not sufficiently mature and flexible for industrial app development. Another is that many industry apps can be built sufficiently well without using MDD approaches, or by only using basic MDD approaches e.g. generate skeleton code only [10].

Guided by our findings, we listed ten significant limitations in the selected studies, discussed in Section 4.3.2. We found that eleven out of the fifty-five selected studies are not suitable for use at the professional level either because they (i) are not extendable to other than very narrow app usage domains; (ii) only partial apps can be generated and their code cannot be modified; or (iii) the generator creates code with significant performance and security issues. We also found that another 20 studies need to describe the development processes proposed and analyze their work more thoroughly to be applicable in the real world. We also found that three studies are not suitable for large-scale app development, two developed GUIs separately from the other app components, and two do not specify how the tool and model get their target app requirements. From these analyses, we recommended seven high-priority potential future research areas. The key contributions of the paper include:

- We defined an SLR protocol following Kitchenham's guideline [11] and found 1042 papers potentially related to this topic. Subsequent filtering resulted in 55 primary studies selected for analysis, synthesis, and reporting.
- We extracted information from selected primary studies, carried out a meta-analysis, and present key strengths and limitations with the corresponding discussion.
- We provide guidance for mobile app developers, stakeholders, and researchers who want to better understand what MDD techniques and methodologies have been used to date to support mobile app development, and how these techniques have been employed.
- We identify a set of key recommended research directions to address the limitations of MDD based mobile app development schemes and discuss their impacts.

The rest of this paper is organized as follows. Section 2 briefly discusses key related work. We then present our SLR-based research methodology and data synthesis in Sections Section 3. In Section 4, we provide detailed answers to our key research questions as evidenced from the selected primary studies. Section 5 discusses threats to validity for this SLR. Finally, Section 6 concludes the paper.

2. Background and related work

Although MDD approaches for mobile app development have a long history, this review study is the first large-scale systematic review that accesses existing approaches to provide their classification for identifying gaps, limitations and discussing current trends and future challenges. In this review, we were interested in evaluating existing MDD-based approaches that have investigated mobile app development to date. This section presents some necessary background and key related works required to understand our review and the analysis presented in the following sections.

2.1. Model-driven development and related surveys

The terms **M**odel **D**riven **D**evelopment (MDD), **M**odel **D**riven **S**oftware **D**evelopment (MDSD), and **M**odel **D**riven **E**ngineering (MDE) are often used interchangeably in the literature. The MDD takes a high-level model and successively refines it down to lower-level models, eventually to executable code and/or configurations to produce software. A wide variety of MDD approaches, techniques and tools exist. All of them share a common approach of abstracting aspects of software into high-level models and using tools to synthesize code from these models, rather than writing code by hand. MDD tools have been developed for a great range of application domains, including web applications, user interfaces, test case generation, embedded systems, different domain-specific applications, and mobile app generation [10].

The Object Management Group has developed a standard and defined a **M**odel **D**riven **A**rchitecture (MDA) for MDD, used by many MDD approaches. MDA contains a set of rules and tools for problem modeling and defining the solutions. There are three types of models in MDA: (i) **C**omputation **I**ndependent **M**odel (CIM) for the business requirements; (ii) **P**latform **I**ndependent **M**odel (PIM) for system architecture; and (iii) **P**latform **S**pecific **M**odel (PSM) for model transformations.

Software **P**roduct **L**ines (SPL), **S**oftware **F**actories (SF) and Domain Specific Languages (DSL) are sometimes considered to be kinds of MDD/MDSD approaches. SPL reuses pre-built software artifacts for development, whereas SF draws parallel models with traditional manufacturing processes where software is assembled from pre-made parts. These techniques have also been used to create mobile apps, where features are identified in the domain analysis [10].

A detailed survey on MDSD is presented [12] that illustrates MDSD essential elements and relationships between them, e.g., modeling languages, domain knowledge, meta-models, formal methods, model transformations, and standards. In [13], Liddle et al. discuss how MDD approaches work in practice with examples and use cases. An interesting architecture-centric MDSD (AC-MDSD) approach is presented in [14]. The authors discuss the economic advantages of AC-MDSD over other approaches. Recently, Brambilla et al. [15] discuss the impact of MDD approaches in practice, especially for software professionals. This book is a good read for novice software engineers since it explains

MDD's basic principles and techniques. It also discusses how MDD can provide an agile and flexible tool, and how to select the right set of MDSD instruments for a specific project. In [16], two companies who are willing to adopt the principles of MDD are examined. This case study analysis results explain the differences in requirements for MDD in these organizations. It also discusses the factors that influence decision upon adoption, the potentially suitable modeling notation for each of the companies, and the conditions that should be fulfilled to increase the chances of success, i.e., in adopting MDD in current industries.

2.2. State of the art approaches for mobile app development and related surveys

There exist many tools and frameworks for mobile app development. Surveys from Heitkötter et al. [17] and Willocx et al. [18] present two detailed and excellent reviews on this topic. However, Barnett et al. [10] demonstrate that most mobile app development approaches reviewed in these survey papers (i) do not consider or only partially consider the technical domain model of the app, (ii) most model analysis is absent, and (iii) provide little guidance on how to construct the underlying meta-model used, if present at all. Some key examples of leading edge approaches for mobile app development are summarized below.

- **Nitrogen:** Nitrogen is a codeless and cloud-based development platform for enterprises. Due to the codeless environment, it provides limited flexibility and does not support all concepts of mobile app development for its users.
- **App Inventor:** App Inventor is a tool that helps children to build mobile apps based on visual programming languages. It simplifies mobile app development by hiding all implementation details. It consists of a meta-model for the concepts exposed to the developers but does not consider the technical domain's conceptual concerns, such as hardware constraints, event handling, network connection, etc.
- **Appcelerator:** Appcelerator is a widely used framework for mobile app development. It uses a model-based tool and uses JavaScript to build apps that run on multiple platforms, but does not specify an underlying meta-model.
- **Xamarin:** Xamarin is a cross-platform framework for mobile app development based on the C# programming language. It wraps the underlying mobile platform API so that developers can build the functionality that they desire.
- **Smart Maker Authoring Tool:** Smart Maker Authoring Tool enables non-developers to develop mobile apps and webs for their work via basic concepts. The user of this tool does not require any prior knowledge in programming or coding. The tool also follows the no-code/low-code development principles and tries to improve the application structure and operation mechanism for implementing the desired app/web functions.
- **Cordova:** Cordova is a widely used hybrid app development framework that hides platform-specific details. Ionic is built on top of Cordova and includes core UI components for building hybrid mobile apps that look like native apps. There are several third-party generators used for generating parts of the mobile app.
- **WebRatio:** WebRatio is a commercial tool that uses MDD for mobile app development based on the **O**bject **M**anagement **G**roup (OMG) extended standard **I**nteraction **F**low **M**odeling **L**anguage (IFML). The main strength of WebRatio is that it can generate cross-platform hybrid mobile apps using the Cordova framework.
- **MobiA:** MobiA is a graphical tool for health monitoring application development. This tool's target user is health professionals, and hence, technical details of app development are hidden.

There also exist several reverse engineering-based tools to support mobile app testing, UI artifacts modeling and tools for code recommendation [10]. In [19], Wasserman et al. point out many key **S**oftware **E**ngineering (SE) issues for mobile app development. Some real-world challenges for mobile app development are illustrated in [20]. Barnett et al. showed **D**omain **S**pecific **L**anguages (DSLs) can be used to generate useful mobile apps in industry [10]. One significant advantage of using a DSL is that the solutions can be expressed in the idiom and at the level of abstraction of the problem domain [21]. In [22], an evaluation framework is presented to analyze the current no-code/low-code app development platforms such as AppArchitect, EachScape, Form.com, iBuildApp, OutSystems, PhoneGap, RhoMobile, and SenchaTouch. The authors also showed the impact of some platforms at different app development life-cycle stages. A review of current representative low-code/no-code development platforms is presented in [23] that showed the required classification on the existing low-code platforms. The analysis results claim to help end-users selecting the most appropriate platforms based on their requirements. A survey and a SLR on MDD for mobile apps were presented in [24] and [25], respectively. However, these two review studies used a limited subset of existing works, and we decided that a more detailed and rigorous investigation was needed.

3. Research methodology

This section defines our **S**ystematic **L**iterature **R**eview (SLR) protocol based on the guidelines provided by Kitchenham et al. [11] and our previous experiences [26–29]. A high level workflow diagram is shown in Fig. 1. The review protocol development for this study was carried out by the first author under the close supervision of the remaining authors, who are experienced in performing and supervising SLRs in software engineering. The first author was also responsible for the initial study selection, i.e., searching and study accumulation, quality and quantitative assessment, study filtration, data extraction with the supervision of other authors. The extracted data were synthesized using meta-analysis techniques from the 55 original articles (summarized in Appendix A).

3.1. Research questions

Our objective was to analyze the existing research on "why" and "how" MDD techniques influence mobile app development, and what research gaps exist in these domains. Thus, we formulated three key RQs to answer this. Petticrew at el. [30] show that five elements, i.e., **P**opulation, **I**nterventions, **C**omparison, **O**utcomes, and **C**ontext (PICOC), can be used to direct the formation of RQs for a searchable study. The PICOC for this SLR is shown in Table 1, following the guidelines of Petticrew et al. modified for software engineering taxonomies [11].

Table 1
PICOC for this SLR.

Population	The literature on model driven development (MDD)
Intervention	Mobile apps development techniques, method, process and tools
Comparison	Comparison among interventions for analysis
Outcomes	The consequence of MDD for mobile apps development
Context	*Include:* MDD techniques, process, language for mobile app and tool development *Exclude:* IoT, threats, privacy, malware, hardware and communication

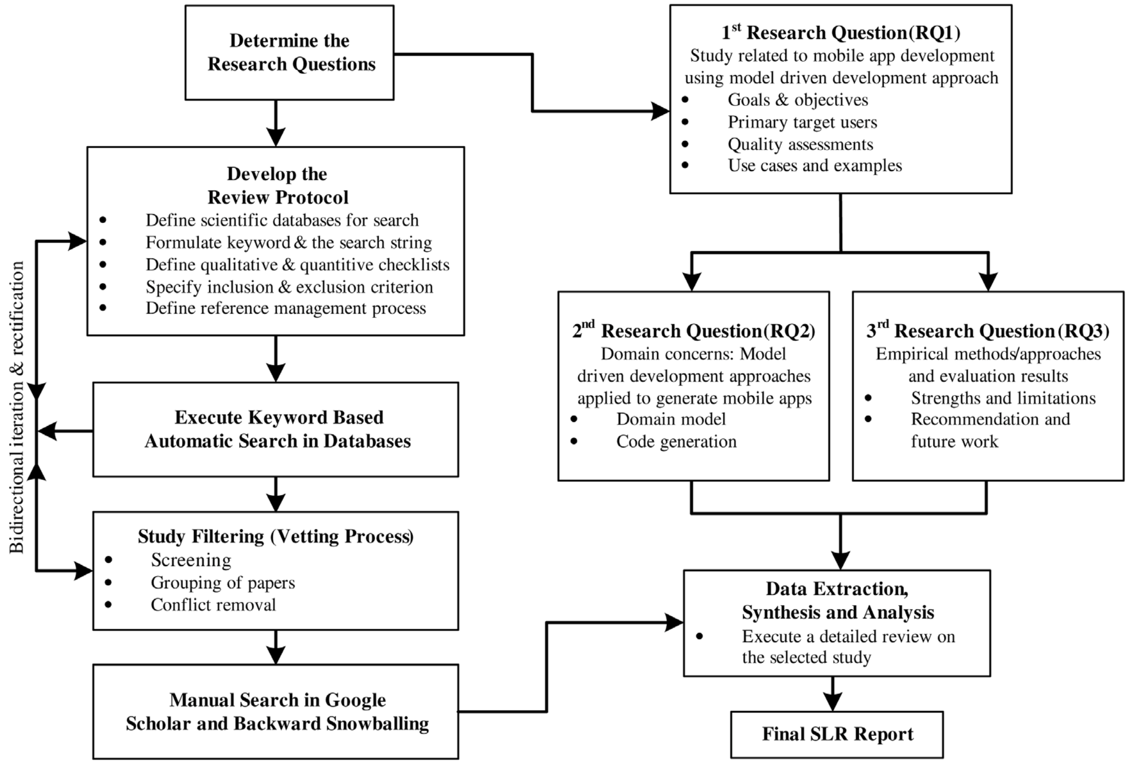


Fig. 1. Architectural block diagram for this SLR.

RQ1 What are the main goals and objectives for generating mobile apps using model driven approaches?

- **RQ_1-$SubRQ_A$** What are the goals and objectives for each research paper reviewed?
- **RQ_1-$SubRQ_B$** Who are the target end-users of the tools and generated apps?
- **RQ_1-$SubRQ_C$** Is the study applied to academic or industrial problems or both?

RQ2 What model-driven approaches have been applied to date to generate mobile apps?

- **RQ_2-$SubRQ_A$** What are the main domain model(s) used by the researchers?
- **RQ_2-$SubRQ_B$** What are the code generation steps? How is it accomplished?

RQ3 Which empirical methods are used in the selected studies to evaluate MDD based app development approaches, and what are the results obtained?

- **RQ_3-$SubRQ_A$** How were the studies evaluated?
- **RQ_3-$SubRQ_B$** What are the strengths and limitations of the selected studies?
- **RQ_3-$SubRQ_C$** What are our recommendations for future work in this area?

3.2. Search strategy

We developed a strategy to search for papers that target mobile app development using MDD. The goal was to find as many primary study papers as possible. Our strategy consisted of three parts: search string identification (Section 3.2.1), automatic search in electronic database (Section 3.2.2) and snowballing using google scholar (Section 3.2.3).

Table 2
Concepts and search terms explanation.

Main terms	Supportive search terms
Concept 1 (Co1): MDD	Model Driven Development, Model Driven Software Development, Model Driven Engineering, Domain Specific Language, Domain Specific Modeling Language, Domain Specific Visual Language, Platform Independent Model, Computation Independent Model, Platform Specific Model, Code, Transformation, Rule Based Transformation, Unified Modeling Language, Model Driven Architecture, Generate, Generator.
Concept 2 (Co2): Mobile	Platforms: Android, iOS, Windows, BlackBerry, Symbian, webOS, Ubuntu Touch, Tizen. Device: Mobile, Smartphone, Tablet, Cellphone, Cellular telephone.
Concept 3 (Co3): App Development	Generated Applications, Developed Tools, Used Tools, Improvement, Outcomes, Framework.

3.2.1. Search string formulation

Relevant primary studies for this SLR were identified based on the RQs defined in Section 3.1. With the assistance of the PICOC approach (Table 1), our search terms were divided into three primary concepts, as shown in Table 2. These concepts helped us to set a well-formulated search string. We also used synonyms, abbreviations, and alternative spellings of search terms to increase the number of relevant research papers. We used truncation and wildcard operators to save time and effort in finding these alternative keywords. Moreover, different supplementary key terms or phrases discovered during search iterations were added to the supportive search terms list to enhance our search strategy. For example, the supportive search terms 'Code' and 'Transformation' are applied with wildcard operator '*' and database search operators NEAR, respectively. Our supposition is that they will collect all relevant articles that contains no-code/low-code and product line related articles. When constructing the final search query, the identified keywords, their alternatives and related terms were linked with Boolean AND (&&), OR (||) and NOT (¬) operators. The OR

Table 3
Inclusion criteria.

ID	Detail criterion
IC_1	Full text of Conference papers, Journal articles and Book chapters that comply with three concepts defined in Table 2.
IC_2	Entire papers are written in English and use academic literature references.
IC_3	Studies that propose a solution or partial solution for Mobile apps development using an abstract model, languages, code generation and tools.
IC_4	Papers available in an electronic format i.e., doc, docx, pdf, HTML, ps.

operator was used to concatenate the synonyms; AND to concatenate the major concepts; and NOT to reduce the unwanted contents (UC) as follows:

$$[\{(C_{11}\|C_{12}\|\ldots\|\ C_{1n})\mathbf{AND}(C_{21}\|C_{22}\|\ldots\|\ C_{2n})\mathbf{AND}(C_{31}\|C_{32}\|\ldots\|\ C_{3n})\}$$
$$\mathbf{NOT}(UC_1\|UC_2\|\ldots\|\ UC_n)] \tag{1}$$

where $C_{11\ldots1n}$, $C_{21\ldots2n}$, and $C_{31\ldots3n}$ ε Co1, Co2 and Co3 of Table 2, respectively; and $UC_1 \ldots UC_n$ refers the **Exclude Context** as define in Table 1. During search we exclude following terms: ‘Malware, Classification, Clustering, Cloud, Wearable, Network, Test, IoT, Energy and Bug’.

3.2.2. Automatic search in electronic databases for scientific literature

We performed searches using four electronic databases for publications without any time range: *ACM Digital Library*, *IEEE Xplore*, *Science Direct*, and *Springer Link*. We chose these databases because they contain most high quality, peer-reviewed papers in Computer Science and Software Engineering including MDD and Mobile app development. We chose to ignore some of the secondary indexing search engines like SCOPUS and INSPEC because they contain a large number of duplicate studies. However, we also used snowballing from located study references via Google Scholar to find additional studies and make our review more comprehensive. We ignored the terms related to IoT, Malware, Energy, and Testing from the search query using Boolean NOT operator due to the following reasons:

- We were exclusively interested in MDD for Mobile apps development domain
- We were interested in app modeling and generation rather than test case generation and bug fixing. Test case generation might be an interesting topic to look in the future.
- IoT requires extra hardware and is beyond the scope of this study.
- Energy issues and malware detection are not related to the concepts defined in Table 2.

3.2.3. Snowballing using Google scholar

Our database searches yielded a large set of primary papers. We also manually searched Google Scholar using the primary and supportive terms defined in Table 2 as we did not want to miss any relevant existing study and wanted to make sure that the final set of papers is complete. We analyzed the references from the finally selected studies to also check for any potentially missed primary studies.

3.2.4. Selection of papers: Inclusion and exclusion criterion

Tables 3 and 4 present the **I**nclusion **C**riteria (IC) and **E**xclusion **C**riteria (EC) that have been used to identify the studies of this SLR, respectively.

Table 4
Exclusion criteria.

ID	Detail criterion
EC_1	Gray literature, Workshop articles, posters, books, work in-progress proposals, key notes, editorial, secondary or review studies.
EC_2	Discussions papers and opinion papers, as well as Surveys that do not include any solution defined is IC_3
EC_3	Short papers less than three pages, irrelevant and low quality studies that do not contain considerable amount of information to extract
EC_4	Papers discussing on MDD or similar terms but not regarding Mobile apps development e.g., mobile Malware, Testing or IoT focused
EC_5	Mobile app development without MDD, beyond the scope, no real work or implementation
EC_6	Conference papers and book chapters if an extended or recent journal version is available (from same authors), and when full version is unavailable.

3.2.5. Collection and filtering of the studies

Our filtration process is summarized in Fig. 2. Initially we ran the formatted query on four major databases that returned 1031 research papers. We then applied filtering and classified the studies found [11]. In our initial filtration process, we removed 44 papers due to being duplicated articles, editorial or key notes. After reading the title, abstract, conclusion and skimming through the introduction, methodology and results, we applied our exclusion criterion defined in Table 4, and 873 further papers were removed. During the third step of filtration, we applied inclusion criteria and removed 63 papers as these studies did not meet any ICs shown in Table 3. In parallel, we did a manual search and found only 11 papers that meet all three concepts defined in Table 2 but not contain any unwanted content (UC) defined in Eq. (1). After applying ICs and ECs, three out of eleven papers were selected. Finally, we did a cross-check and ended up with 55 papers as our primary set of studies for analysis after completing the filtration process.

3.2.6. Quality assessment

We used 1 to 5 numeric score – Very Poor, Inadequate, Moderate, Good and Excellent – **Q**uality **C**hecking (QC) applied to each study using following six questions (QC_1 to QC_6). We label a paper as a poor quality paper if its average value for all QCs is ≤2.00, otherwise we use the qualitative information (discussed in Section 3.2.7) to decide this.[1]

QC_1: Is the study highly relevant to the research and concepts defined in Tables 1 and 2, i.e. clearly uses an MDD based technique to generate mobile apps?

QC_2: Does the study clearly explain the methodology that accomplishes its goals?

QC_3: Does the study provide sufficient information on data collection, prototyping and/or algorithms used?

QC_4: Is there a clear outcome and results analysis reported?

QC_5: Are study limitations and possible future work adequately described?

QC_6: What is the citation count and quality of the venue where the study was published?

[1] These poor quality papers were dropped from our set of primary studies without further investigation.

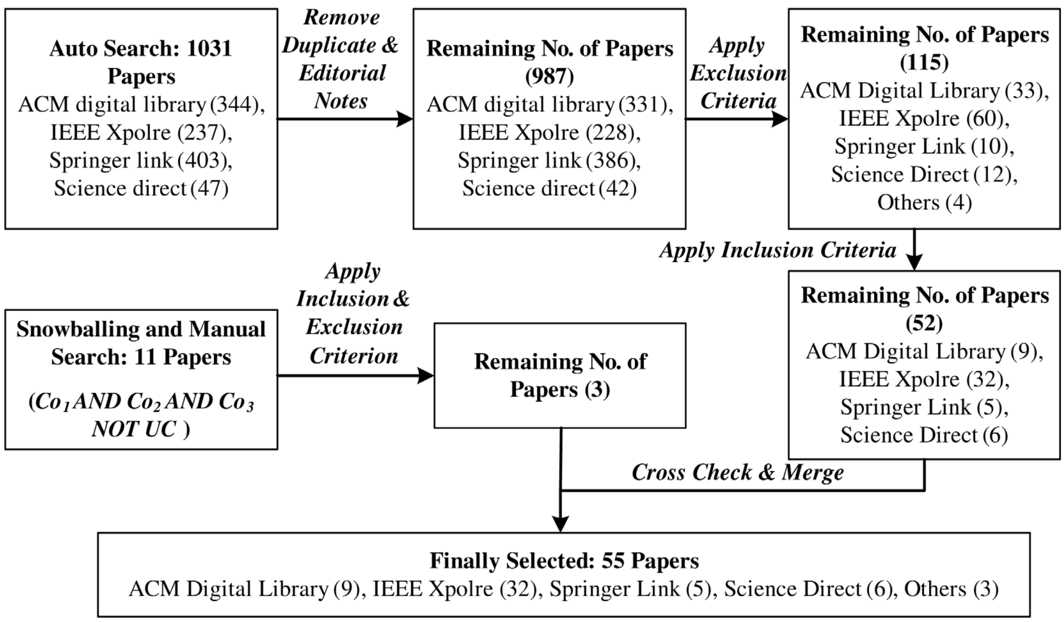


Fig. 2. Primary study selection process steps.

3.2.7. Qualitative information to be extracted from each paper

We extracted the following fifteen key information items from each primary selected paper, forming its **Q**ualitative **I**nformation (QI):

QI_1: Publication details — authors, title, date, venue, citation count, publisher.

QI_2: What are the main goals and objectives of this study?

QI_3: Is the user/case study from Academia or Industry?

QI_4: What domain(s) are the generated apps targeting e.g. retail, travel, entertainment.?

QI_5: Who are the target end-users of the tool e.g. business analysts, app developers, end users, etc?

QI_6: What are the underlying model(s) (domain models and architecture) used?

QI_7: How are the target app requirements specified?

QI_8: How is the code generator implemented?

QI_9: Does it produce complete or partial output (generates a full app or only generates a part of an app)?

QI_{10}: Can the generated apps be hand-modified?

QI_{11}: How was the study evaluated?

QI_{12}: Is the tool scalable to large apps?

QI_{13}: Does the study (tool) generate a quality output (app)? How is this measured?

QI_{14}: What are the main strengths and limitations of the presented work?

QI_{15}: What are the identified research gaps and future work ideas?

3.2.8. Reference management and screening tool

We used EndNote X9 tool for reference management and screening the studies because it facilitates easy removal of double entries and keeps track of papers by summarizing essential facts, e.g., title, authors, abstract, keywords, venue, date, and page numbers.

3.3. Data extraction and synthesis

During data extraction, we downloaded all primary studies and grouped the papers as per the theme, contribution, authors, and **E**lectronic **D**atabase (ED) name in this order. An identity code was formulated and assigned to every individual study. The list of papers with their identity code is available in Appendix A. We followed the following steps to counter the biases during data extraction:

- Initially, the first author of this paper extracted data for two papers from each selected ED and stored the results in a google sheet. The remaining authors of this report cross-checked these data, and the necessary correction was applied.
- Then the first author extracted data for another twenty selected studies, and similar cross-checking was performed until all of the authors reached agreement and the outcome did not vary more than 5% for anyone. At the end of this step, the review protocol was finalized to incorporate the changes.
- In the third step, the first author re-extracted the data from previously examined studies as well as the remaining twenty-seven studies as per the revised protocol. The extracted data were sequentially cross-checked by the remaining authors (once each) to minimize extraction bias and omissions.
- Finally, all data was stored in a google sheet for analysis and synthesis.

4. Evaluation results and analysis

We extracted qualitative, quantitative and mixed data from the selected 55 primary studies. We also used visualization tools and meta-analysis techniques to present our analysis, especially to answer the research questions defined in Section 3.1. Fig. 3 show the year of publication for all selected studies. Appendix A contains the full list of references for the selected studies. We found, there were 6 journal papers, 1 book chapter, 4 symposium papers, 3 workshop articles and 41 conference papers. All of the selected primary studies were published between 2005 and 2019. We collected the paper list in 2020. Hence, there may be papers published after our search. For example, two very new relevant papers [31,32] have now been published in 2021. The first paper [31] describes mobile app synthesis from UML models applying the UML toolset to generate native apps for Android

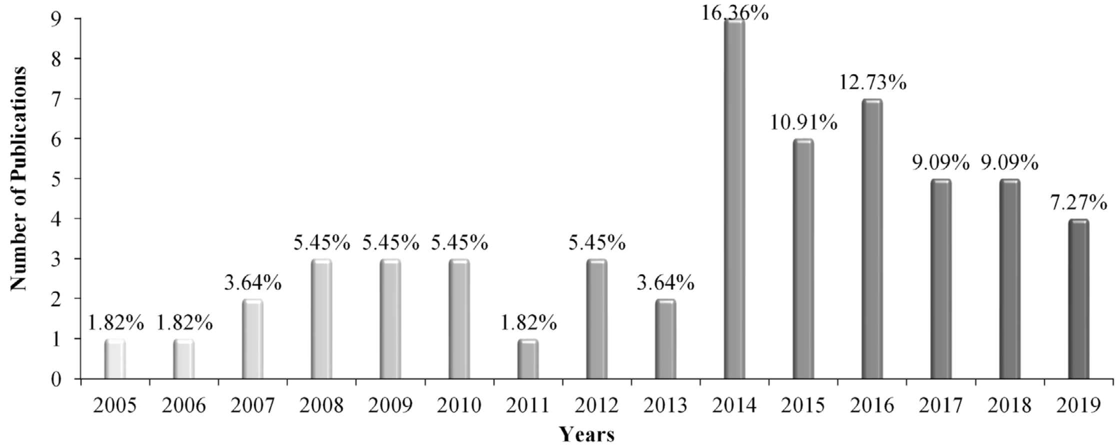
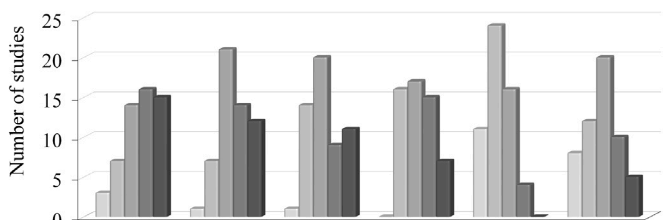


Fig. 3. Number of publications per year.



	QC_1	QC_2	QC_3	QC_4	QC_5	QC_6
Score 1	3	1	1	0	11	8
Score 2	7	7	14	16	24	12
Score 3	14	21	20	17	16	20
Score 4	16	14	9	15	4	10
Score 5	15	12	11	7	0	5

Fig. 4. Distribution of quality assessment score.

or iOS platforms. The second article [32] developed an excellent framework called MAndroid to generate Android-based classic multiplayer 2D board games. Overall, we found at least one study in each year since 2005, and in 2014, we found the highest number of studies. Although there is a considerable increase in the number of studies from 2013 to 2014, after 2014, it has now leveled off or decreased slightly.

We defined six quality assessment criteria for the primary studies (described in Section 3.2.6). Their distribution is shown in Fig. 4 and individual scores for each question are attached in Appendix B. We scored more than 60% of the selected studies as ≥3 for all the questions except QC_5 – Future work summary. This suggests many studies poorly identify and define the appropriate future scope for MDD-based approaches to mobile app development. However, 81.82% of the selected studies clearly answer QC_1 with score ≥3, with clear motivation and objectives. QC_3 – data collection and algorithms clearly described and QC_4 – outcome and analysis also score well overall. QC_2 – methodology and chosen approach are good to excellent in most studies. Most studies have good citations and appear in good to excellent venues – QC_6.

As an example, ED52 proposes a model for conducting early usability evaluation for mobile apps generated with an MDE tool. Although the goal is clear, it does not align well with the methodology presented later in the paper. More specifically, it defines the usability metrics and corresponding sub-characteristics, but the method contains only applicability discussion of the proposal for a 'Car Rental System'. We thus scored it 2 for QC_1. We also scored 2 for questions QC_2 to QC_6 for this study, since the paper lacks details about its data collection, does not discuss the used algorithm, presents few findings that do not match with the goal, and future work is not explained but the summary with study limitations is provided.

In contrast, ED30 presents JustModeling, which is a MDD based approach for business app development. This research has a clear-cut objective that well-aligns with the goal of MDD. It also presents a novel model/methodology, excellent implementation and currently is in use and therefore, we scored it 5 for QC_1. The methodology and goal align (≥90%) highly (score 5 for QC_2); it describes clear and appropriate data collection procedures and algorithms (score 5 for QC_3); has clear findings and explanation for analysis (score 4 for QC_4); discusses summaries but its limitations and future works are not illustrated appropriately, for example, it does not take into account coding the class methods, e.g., method codes need to be inserted manually by the developers after automatic code generation steps, and it misses structural component (score 3 for QC_5); and the work is published in Brazilian Symposium on Computing Systems Engineering which is a moderate venue in software engineering (score 3 for QC_6).

4.1. RQ1: What are the main goals and objectives for generating mobile apps using model driven approaches?

The first research question in our SLR tried to identify the motivation behind each selected study. Overall, the studies aimed to add more flexibility in mobile app development through high-level modeling, consequently increasing productivity. We also tried to find the target end-users and applications areas. We found that most approaches were applied to examples and use cases from academia and only a few from industry or in collaboration. We present our analysis and finding on these in the following subsections, answering three related sub-research questions.

4.1.1. RQ_1-$SubRQ_A$ What are the goals and objectives for each research paper reviewed?

Mobile app development steps are similar to traditional software development steps that begin with platform selection, target user identification, constraint mapping, data collection, implementation and testing. However, the data collection may not be required for some trivial apps, e.g., a calculator or photo viewer apps. Model Driven Development methods enable the synthesis of a mobile app through a model transformation process ending up with code generation [10]. The objective is ultimately to raise the abstraction level and increase the level of automation. Thus, it improves productivity, increases the quality of the apps, reduces the risks, and provides tools for formal analysis. Several works [17,33] present novel frameworks and tools for mobile app development but not all are based on MDD techniques. To better understand limitations with current mobile app development approaches using MDD, this SLR identifies the main goals and objectives for generating mobile apps using MDD approaches.

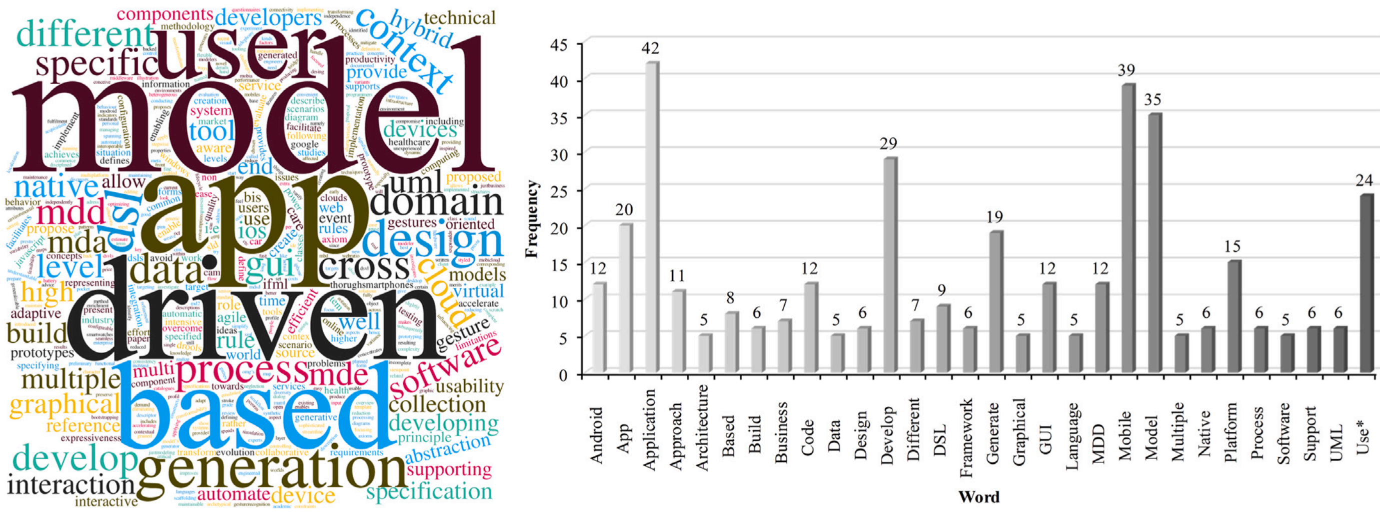


Fig. 5. Selected studies (a) Word Cloud from objective and title (b) Frequently used words to describe goals and objectives.

Table 5
Identified common objectives and goals in the selected studies.

Attributes: Goal and objectives	Studies	%
Flexibility: Provide more flexibility through executable models as an integral part and for app evolution.	ED1 ED2 ED3 ED4 ED5 ED6 ED16 ED18 ED22 ED30 ED32 ED33 ED34 ED35 ED38 ED46 ED54	30.90%
Efficiency: Reduce app development time and cost and hence increase productivity.	ED2 ED4 ED5 ED7 ED16 ED31 ED33 ED34 ED37 ED38	18.18%
Reliability: Integrating design models and approaches to support reliability and corresponding analysis.	ED2 ED3 ED5 ED10 ED13 ED15 ED20 ED23 ED25 ED29 ED32 ED34 ED52	23.63%
Reuse: Ensure reusable methods to generate the apps or a proportion of apps using DSL, DSVL, framework or templates.	ED8 ED10 ED11 ED12 ED17 ED18 ED19 ED21 ED23 ED28 ED30 ED31 ED45	23.63%
Quality: Increase the quality of the developed app.	ED1 ED14 ED36 ED42 ED47	9.09%

We extracted the main goals and objectives from the text of the 55 selected studies. We then performed thematic coding analysis and found that the primary studies use in total around five hundred words to illustrate the goals of their studies. A word cloud illustrating the key phrases from the extracted text is shown in Fig. 5(a), and distribution of the most frequent words with *frequency≥5* for this word cloud is shown in Fig. 5(b). Phrases such as 'Application, Develop, Mobile, Model and Use' are very commonly used by the researchers, as to be expected in the sense that these works try to address concerns in this domain. 'Code, Generate and Platform' are also used frequently which highlights key issues to address in code generation and platform selection. However, the frequencies of words 'Architecture, Data, Language, Native, Multiple and Software' are much lower. We also found that 72.72% of the selected studies map to a set of five common goals, shown in Table 5.

One-third of the primarily selected studies try to increase the flexibility of app development process through use of MDD techniques. However, we found only one of the project development principles is flexible in most cases (selected studies), while others are rigid. For example, study ED5 aims to accelerate Android app development following the **C**reate, **R**ead, **U**pdate and **D**elete (CRUD) pattern. Here, **Q**uery **V**iew **T**ransformation (QVT) is used, transforming the UML class diagram (PIM) to the Android model (PSM) at the metamodel level. Then Acceleo tool generates the code from the PSM through an MVC pattern implementation. However, it is limited to a particular kind of variability and does not support modeling and variant generation for mobile domain-specific hardware and software features. Thirteen studies have explored improving reusability in app development. Initially, most of these studies build a library of app components or templates. Then, this library is used to help in generating a complete app. For example, ED19 proposes the RUMO framework for UI design. It creates different versions of pre-built template files to address the issue of different versions of a platform or multiple reusable templates for multiple platforms. Each template file is responsible for creating the source code for the desired platform.

We found only five selected studies aimed to increase product quality exclusively. This is a small number compared to other parameters presented in Table 5, in the sense that all the quality attribute parameters are very hard to achieve due to the laborious synthesis process in a single project. We also identified that maintainability among the software product quality attributes was implicitly addressed, but compatibility, functionality, and effectiveness still need further attention. For example, ED42 investigates a model-based approach to support non-technical users creating data collection tools according to their actual needs. Here, task models are described in constraints on task execution as temporal relations between sub-tasks to increase the quality. However, the effect of integrating the data collection method with the tool is not shown.

We also found that a sub-set of studies address more than one common goal summarized in Table 5. Graphical distribution of these studies in terms of common goal attainments using a Venn diagram is shown in Fig. 6. In addition, we found 27.27% of studies did not address any of these five common goals. The main objectives and goals of these 15 papers are shown in Table 6.

The first study ED9 combines aspect-oriented development techniques with MDD for app development. It provides a set of techniques to modularize crosscutting behavior. The main aim is to manage the complexity of contexts, e.g., environmental factors, device limitations, and connectivity. The study ED24 keeps the **D**omain-**S**pecific **V**isual **L**anguages (DSVL) and **D**omain-**S**pecific **T**ransformation **L**anguages (DSTL) in sync for the app development and code generation. Though this is an improvement on previous code generators, it aims to help the experienced developers, whereas the studies ED26

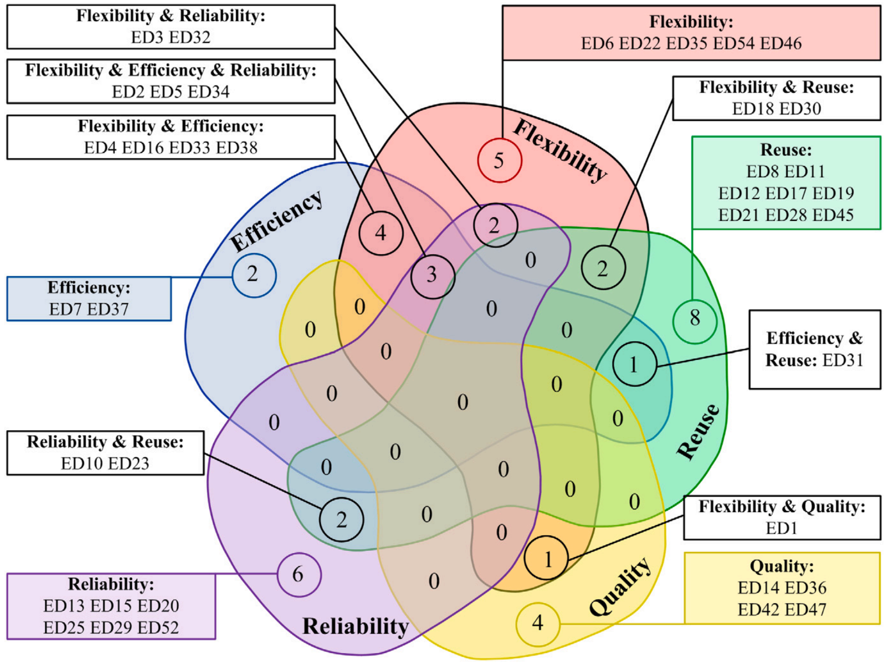


Fig. 6. Venn diagram represents the overlapping studies in common goals attainments.

Table 6
Identified uncommon objectives and goals in the selected studies.

Study	Primary objectives	Study	Primary objectives
ED9	Better manage complexity	ED24	Scaffolding for a mobile app generation
ED26	Evaluate app feasibility to an archetypal case	ED27	Utilize independent language to generate code
ED39	Streamline prototyping process	ED40	Ease computing for Hybrid app development
ED41	Enable single code base usages for all platforms	ED43	Support collaborative design
ED44	Specifying behavior of data collection process	ED48	Generate synthetic emulation code
ED49	Enable users to create their own DSL	ED50	Specify domain and user interaction models
ED51	Integration of different modeling environments	ED53	Create inter-operable apps
ED55	Experimental development		

and ED39 target inexperienced developers to start producing mobile apps. The main goal of study ED27 is to use an independent modeling language to generate native app code. However, there is no standard procedure for confirming the completeness of GUI specified by the language shown. Hence, we did not consider this as a potential candidate for the productivity of Table 5. Similarly, study ED55 aims to easy model process, but the prototype is far from productive usage.

4.1.2. RQ_1-$SubRQ_B$ Who are the target end-users of the tools and generated apps?

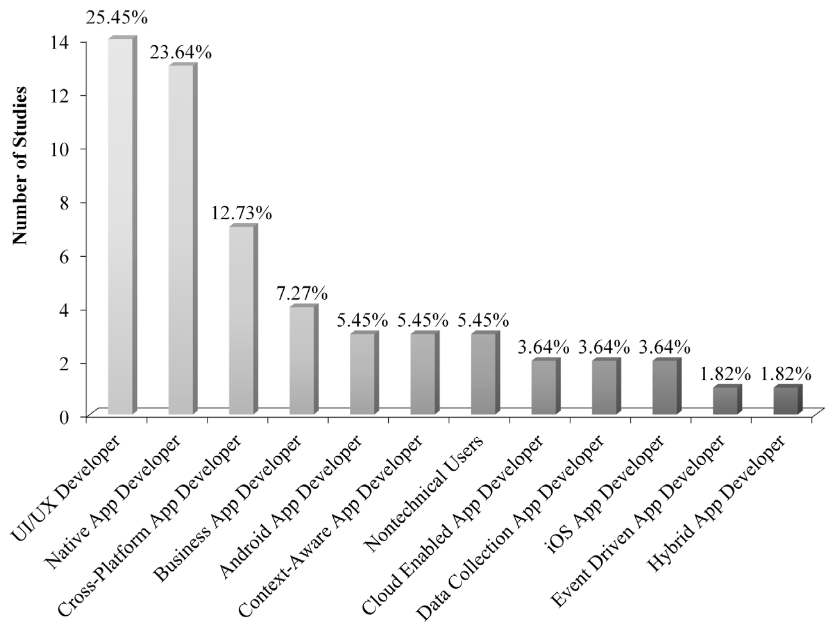
Fig. 7(a) summarizes the primary tool user groups we identified for each selected study. We identified the target tool user by two methods (i) Directly from the paper (23.6%) or (ii) Inferred by evaluations described in the papers (76.4%). We tried to distinguish the tools for different types of developers by (i) Existing facilities such as programming language, frameworks and SDK and (ii) Work procedure and functionalities that are shown in the examples and evaluations in the studies.

We found that the tool design choices of the researchers mostly focus on app development or the GUI aspect of part of the app development. Interestingly only study ED24 focuses solely on supporting professional mobile app developers and they explicitly mention that their tools are not for general users or novice app engineers. Three studies (ED10 ED38 ED49) explicitly target non-technical app developers. In our analysis we were also able to identify the 12 types of users in terms of generated apps. All these types are extracted from the selected studies evaluation and use cases. We could not identify any specific target user of the app development tool for three studies (ED28 ED34 ED50) because there is no example or evaluation.

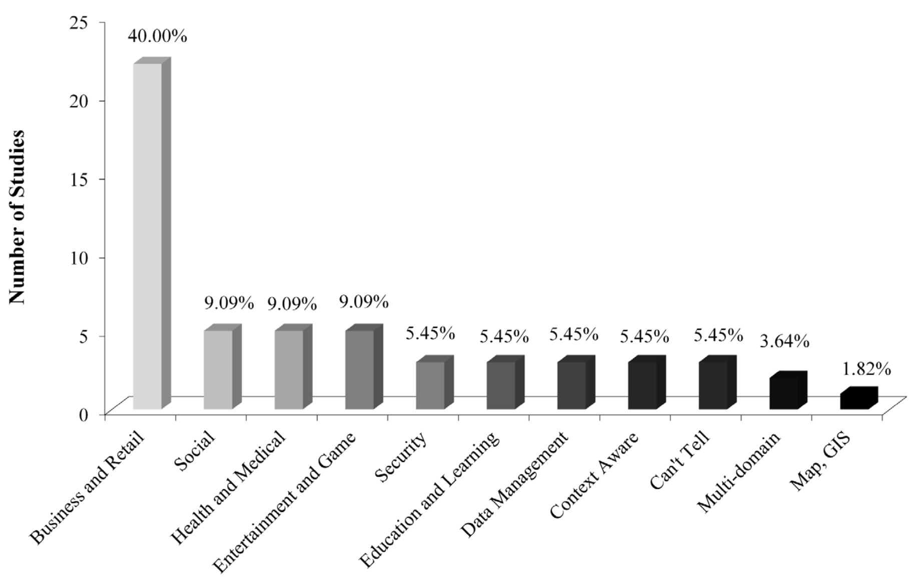
Fig. 7(b) summarizes the primary target domain end users we identified for each selected study. We determined this by reviewing each study's objectives, any case studies they used, and the evaluations described in the studies. While sub-categorizing the target end-user of the generated app we tried to match the evaluation results with the example so that the main goal of the paper remains addressed i.e., not consider the concerns of the developer, but rather focused on the final app user. Most apps produced by MDD tools in the studies focus on some form of business domain e.g. business app generation, e-commerce, human resource management and ERP solutions (ED5 ED11 ED12 ED14 ED22 ED23 ED27 ED29 ED30 ED32 ED35 ED36 ED40 ED41 ED45 ED46 ED47 ED48 ED51 ED52 ED53 ED54). A small number of studies target health and medical(ED10 ED17 ED37 ED38 ED49), security (ED2 ED3 ED20), entertainment and games (ED13 ED16 ED24 ED31 ED55) and social media domains (ED15 ED19 ED26 ED39 ED33). A few studies target mapping and data management (ED42 ED43 ED44). Two studies claim to be suitable for multi-domain (ED7 ED8), one for map and GIS (ED1). We could not tell in which domain the tools aim to produce app for remaining three apps (ED28 ED34 ED50).

4.1.3. RQ_1-$SubRQ_C$ Is the study applied to academic or industrial problems or both?

There is no clear difference in MDD approaches as to being applied in academia vs industry. In industry, approaches are driven by the goal to develop new products and services and improve quality and productivity, rather than solving a problem theoretically. In academia,



(a)



(b)

Fig. 7. Identified (a) Target end-users of the tool for app modeling and code generation (b) Target domain of app usage -for selected primary studies.

Table 7
Distribution of the selected studies as per academia or industry.

Domain	Selected Studies	%
Industry	ED39 ED50	3.6%
Academia	ED1 ED2 ED5 ED7 ED9 ED10 ED13 ED14 ED15 ED16 ED17 ED18 ED19 ED20 ED21 ED22 ED23 ED24 ED25 ED26 ED27 ED28 ED30 ED31 ED32 ED33 ED34 ED35 ED36 ED37 ED38 ED42 ED43 ED44 ED45 ED46 ED47 ED48 ED49 ED51 ED52 ED53 ED54 ED55	80.0%
Industry academia collaboration	ED3 ED4 ED6 ED8 ED11 ED29 ED40 ED41	14.5%
Unresolved	ED12	1.8%

researchers might be more focused on exploring new theories, concepts, models, platforms, techniques and code generation approaches. We tried to distinguish whether each selected study – in terms of the carried out case studies, examples presented, and claimed end users of the tools – was done on problems in academia or in industry.

Table 7 summarizes the results of our findings. Most applications of MDD to date appear to have only been used in academia, with a few to support academic/industry collaborations. We found only two reporting purely industry-based case studies, and for one it was not possible to identify. Further studies are needed to apply MDD-based techniques for mobile app generation to industry-scale problems and determine their strengths and weaknesses for industrial practice. We also note that the use cases, evaluation results, detailed examples and tools are often not available from the industrial and collaboration cases. For the academic domain many are available either in the paper itself or through publicly accessible downloads.

Table 8
Summarized context of MDD for mobile apps.

Study No	Platform	Programming Language	Model and Architecture	Business Logic	UI/GUI/ App Screen	Data	Behavior	Model Completeness	Study No	Platform	Programming Language	Model and Architecture	Business Logic	UI/GUI/ App Screen	Data	Behavior	Model Completeness
ED1	WE	H						F	ED29	MP	H	✓		✓		✓	P
ED2	MP	H					✓	P	ED30	AN	N		✓			✓	F
ED3	MT	H	✓	✓			✓	F	ED31	AN	N	✓	✓	✓			P
ED4	MP	H					✓	F	ED32	MP	H			✓		✓	F
ED5	MP	H	✓		✓			F	ED33	MP	H				✓		P
ED6	MP	H	✓			✓	✓	F	ED34	MT	N			✓			P
ED7	AN	N	✓		✓			F	ED35	WE	N		✓				P
ED8	MT	N	✓		✓			P	ED36	MP	H	✓	✓	✓			P
ED9	CX	H			✓		✓	P	ED37	MT	N		✓	✓			P
ED10	MT	N			✓			F	ED38	MT	N	✓			✓		P
ED11	MT	N	✓	✓	✓			P	ED39	iO	N		✓	✓			F
ED12	MT	N			✓		✓	P	ED40	H	H	✓		✓			P
ED13	MP	H		✓	✓	✓		P	ED41	MP	H		✓				F
ED14	AN	N						P	ED42	MT	N	✓			✓		P
ED15	H	H	✓		✓			F	ED43	MP	H				✓		F
ED16	AN	N	✓				✓	P	ED44	MT	N				✓	✓	P
ED17	AN	N	✓		✓			P	ED45	MT	N		✓	✓			P
ED18	CX	H	✓		✓		✓	P	ED46	MP	H		✓		✓	✓	F
ED19	iO	N			✓			P	ED47	WE	N	✓	✓	✓			P
ED20	MP	H	✓				✓	P	ED48	MT	N	✓		✓		✓	P
ED21	AN	N	✓		✓			P	ED49	MT	N	✓	✓	✓			F
ED22	AN	N		✓				F	ED50	MP	H	✓					P
ED23	AN	N		✓			✓	P	ED51	MP	H				✓	✓	P
ED24	AN	N	✓					F	ED52	MT	N	✓		✓			P
ED25	CX	H	✓				✓	P	ED53	MP	N		✓	✓			F
ED26	MT	N	✓		✓			P	ED54	MP	H	✓	✓	✓			P
ED27	iO	N			✓			P	ED55	AN	N	✓	✓			✓	F
ED28	AN	N	✓		✓	✓		P									

*Note: In this table we consider following Acronym: **AN** as Android platform, **iO** as iOS platform, **WE** as Wed/Cloud platform, **MP** as Multi/Cross-platform, **MT** as Mobile telephony system (non-samrt), **H** as Hybrid (for column two it represent Web and MT), **CX** as Context aware for MT, **N** as Native, **F** and **P** as Full or Partial completeness of the model, respectively.

4.2. RQ2. What model-driven approaches have been applied to date to generate mobile apps?

This research question tries to identify the model-driven techniques used by the selected studies to generate mobile apps, e.g., modeling, code generation, run-time configuration, and model transformations. Overall, we found that the UML, DSLs, and template frameworks are used to describe an app. We also found that most studies use templates, transformation rules, compilers, parsers, synthesizers, and Java resources for code generation. However, for 29.09% studies, could not determine how this was done. Either detailed information on code generation steps were absent or model interpretation to execute the running apps is not illustrated. An overall summary of context of MDD of mobile apps generation is presented in Table 8. More discussion is presented below, answering two related sub-research questions.

4.2.1. RQ_2-$SubRQ_A$ What are the main domain model(s) used by the researchers?

To answer this sub-research question, initially, we identified domain model(s), framework(s) and tool(s) used by the researchers in the selected primary studies to model the app. We then identified the primary aspects of the apps described in the models, which is summarized in Table 9. We found that the app structure/behavior and front-end development are two main aspects, whereas the data modeling is the least. We also tried to identify how complete the model is, and how much of the app can be generated. We found that the entire system is modeled in 34.55% (19 out of 55) studies where a fully working app is generated. In contrast, in the remaining 36 studies, the model is partially complete and can only generate a part of the app. The *Model Completeness* column of Table 8 (9th and 18th columns) presents this result for each individual study. Finally, we tried to find out how are the app requirements modeled/designed; and which studies used executable UML or OMG's MDA approaches.

In the selected studies we found there are mainly three types of common domain models — the UML, textual Domain-Specific Languages, and Domain-Specific Visual Languages. These three approaches were used by the researchers to model an app in 70.90% of our selected studies. The rest of the research works either present a new prototype, framework or tool to model an app. Fig. 8 summarizes the distribution of this finding in the four areas, and details of these categories are presented next.

Table 9
Primary aspects of the apps described in the models in the selected studies.

Aspects of the Apps Described in the Models	Selected Studies	%
Requirement modeling	ED7 ED10 ED24 ED34 ED41 ED44 ED50 ED52	14.54%
Front-end development (UI/GUI/Screen/Resources/ Transition)	ED8 ED11 ED13 ED17 ED19 ED21 ED26 ED27 ED28 ED31 ED36 ED37 ED39 ED45 ED47 ED49 ED53 ED54	32.72%
Back-end development (client/server/cloud components)	ED1 ED14 ED15 ED22 ED35 ED40	10.90%
App structure/behavior	ED2 ED3 ED4 ED5 ED9 ED12 ED16 ED18 ED20 ED23 ED25 ED29 ED30 ED32 ED46 ED48 ED51 ED55	32.72%
Data (modeling/binding)	ED6 ED33 ED38 ED42 ED43	9.09%

A. Use of UML for system modeling and app generation: We identified 13 studies that used UML or a similar standardized modeling languages (UML 2.0, Ecore) for specifying, visualizing, constructing, business modeling and documenting the artifacts of mobile apps. More detail on modeling process of these studies are as follows:

In ED4, the authors identify a subset of UML that fits the need of the mobile app development domain that applies use-cases for requirements gathering, class diagrams for structural modeling, and state machines for behavioral modeling. The authors also develop a tool

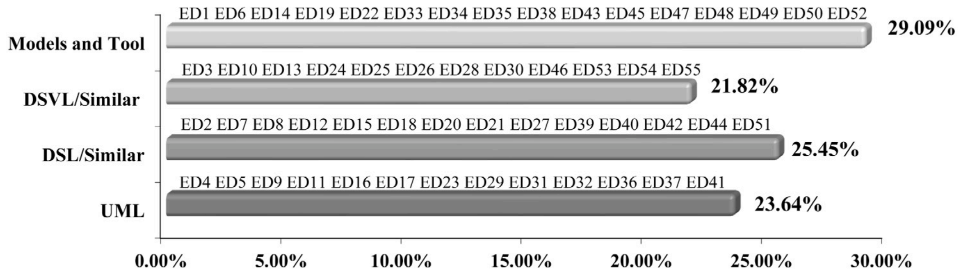


Fig. 8. Distribution of key model types used by researchers for system modeling and app generation.

named Mobile Application Generator (MAG) that takes UML models as input and generates application for the specified target mobile platforms. Therefore, we marked this study as 'F' in the 9th column of Table 8, which means it can model the entire system and can generate a fully working app. Study ED9 uses UML, PSM and composition model for Aspect-Oriented MDD for Context-Aware applications. Here, an app designer begins by modeling the pervasive application in 'Theme UML'. Then, app code is generated from this high-level model. In ED11, the authors propose an approach where mobile business process models are extended with UI models using UML2 class and activity models. The model guides the generation of user interfaces and adds specific platform requirements to the end-user device.

B. Domain-Specific Language (DSL) for system modeling and app generation: Fig. 8 shows that 25.45% (14 out of 55) selected studies use some form of textual domain-specific language (DSL) to describe an app. These studies can be grouped into two subcategories (i) Introducing a new DSL or using a set of new DSLs to model and generate an app (nine studies), and (ii) Extending or use existing DSLs along with tools to model and generate a mobile app (five studies). More details on the modeling process of these studies are discussed below:

In ED2 and ED20, the Agile Model-Driven Approach for developing cross-platform mobile applications is proposed based on AXIOM. AXIOM provides a modeling DSL written in a dynamic language and it defines an app as the platform-independent intent models. In ED7, authors develop a high-level modeling language called MoDroid to ease the development of Android applications. MoDroid implements a meta-Model and its supported tools for the app development e.g., model composition, permission detection, testing and code generation. The authors of MoDroid implement a visitor pattern that traverses the tree structure of an Android model. The pattern takes interface input that declares methods to be executed depending on the node that was localized. In ED8, authors propose a DSL named Menu-Navigation Viewpoint (MNV) for modeling the UI architecture of embedded telephony apps. In MNV, the developers can describe the UI architecture using the fundamental domain concepts.

The first study (ED39) of our second subgroup (extending or using existing DSLs and tools for modeling app) details the criterion that a DSL should have to produce real-world mobile applications. The presented DSL can be used to write the corresponding reference model based on the existing Xtext framework. In ED40, a modeling strategy for cloud-mobile hybrids apps is shown. The DSL shown here is based on the MVC modeling techniques used for web development. The tool is named MobiCloud that produces Android and Blackberry applications as frontends, and Google App Engine and Amazon EC2 applications as back-ends.

C. Domain-Specific Visual Languages (DSVLs) for system modeling and app generation: Twelve out of fifty-five selected studies employ DSVL to describe an app, where the primary app modeling is done through visual platforms. More detail on these modeling techniques are illustrated below:

In ED3, The Event Model (TEM) diagram describes the event causality dependencies for event processing mobile applications. It also illustrates the structure of the logic by TEM Diagrams. Logic concepts are implemented by TEM policy tables, TEM computation tables and TEM event derivation tables. Similar to ED3, authors of ED13 used screen flow diagrams to define screens and describing transitions between screens with events in a mobile application based on an Eclipse plug-in.

In ED10, the authors propose a novel prototype for visually modeling healthcare plans and mobile device code generation using two DSVLs. The first DSVL is named VCPML, which allows healthcare providers to model complex care plans, health activities, performance measurements, sub-care plans, etc. It can also be saved as templates. The second DSVL is named VPAM, which describes a mobile device interface to the user (patients) for the care plan. Both DSVLs are developed using Marama meta tools. The care plan DSVL is incorporated as an Eclipse plug-in. In ED24, the authors develop a tool named RAPPT for generating mobile apps. The tool consists of three major components, the parser, code generator and interface. The interface consists of three screens, that a designer can use to design layout of the app (i) DSVL (ii) Code editor for DSTL (App Modeling Language) and (iii) Code Browser for viewing the generated app.

The authors of ED25 developed a visual tool for visual modeling of contextual rules and contextual information. The name of the tools is CRITiCAL which is designed as an Eclipse IDE plugin. Initially, a model was created by a developer using the tool. Then, Java classes are built from this model and turn it into a Java code. The final product is an Android project integrated with LoCCAM. In ED26, the author presents a GUI modeling language named MIM for mobile applications. Here, The design of the screens were done through MIM Diagram that then checks XML interoperability, and then class diagram is generated.

The authors of ED46 and ED53 use the MAML framework where data, views, business logic, and user interactions are jointly modeled from a process perspective using a graphical DSL. In ED54, the authors develop a tool that allows the use of models and provides a way to support transformations for different target device families. The graphical editor component was built using the JGraph3 library for originating MOF compliant architecture and UI Specification. In ED55, MVC pattern is used to the model business logic which is separated from the UI and the control use class diagrams for Android app development.

D. Uses of frameworks, prototypes and tools for system modeling and app generation: The rest of the sixteen studies either utilize an existing framework or propose a new prototype based on existing tools for generating mobile apps. More detail on domain model for these studies are as follows:

The authors of ED1 use Styled Layer Descriptor (SLD) for the dynamic generation of context-adaptive mobile maps. A tool named ArcMap2SLD-generator is developed. This tool allows designing a ESRI ArcMap and converting it into a valid SLD-file. In ED6, a modeling language and an infrastructure is shown that supports specifying different app variants according to the user roles for MDD of Android apps. Here, users may continuously configure and modify custom content with one app variant, whereas end users are supposed to use provided content in their variant. Models are separated into three sub-models (i) Data model (ii) Process model and (iii) GUI model. Data modeling is supported by the EMF. The language is also designed using EMF and has the possibility to add a textual syntax.

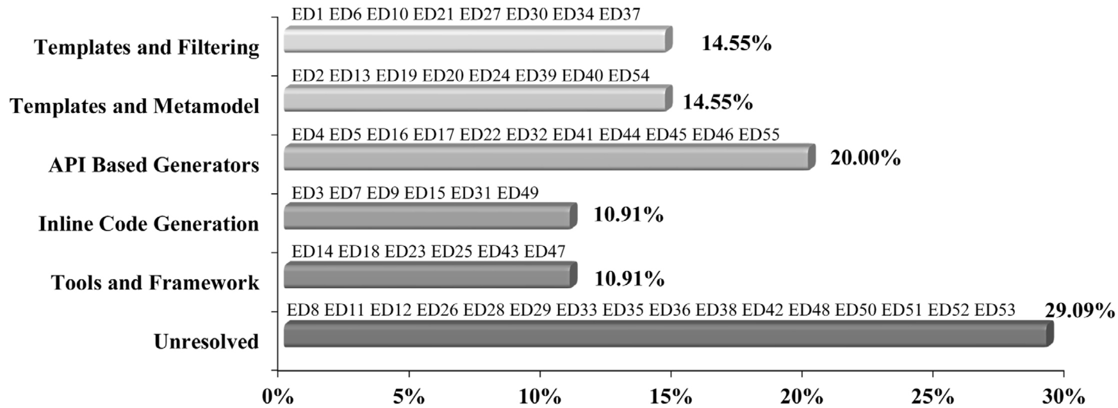


Fig. 9. Distribution of key code generation techniques.

Study ED34 contributes to the Drools rule-based transformation approach to automate the mobile application development process based on the Umple model. Umple is a model-oriented programming language that supports modeling using a textual notation. The presented architecture is divided into three major parts: (i) Parser: Receives an input model, written in Umple language, tokenizes it and passes it to the next component transform; (ii) Transformer: Processes the tokens previously obtained and transforms them into internal representation consistent with target source code model using a predefined set of Drools mapping rules. It also deploys the knowledge base rule engine; and (iii) Code generator: Translates the internal representation into target artifacts; source code as Java, XML and android activity class. Each component is tested independently to ensure that the input is processed correctly and the produced output is valid.

In ED47, the authors introduced the BAMOS platform that supports the specification of a mobile app using XML files to generate XForms screens i.e., it can only partially model the system and can generate a part of an app. Therefore, we marked this study as 'P' in the 18th column of Table 8. ED48 shows R & D efforts to answer (i) How models of mobile software architecture can be built?, (ii) How instrumented emulation code can be generated to run on the target mobile device?, and (iii) How the emulation code can be used to glean important estimates of software power consumption and performance? The authors develop a System Power Optimization Tool (SPOT) for optimizing performance and power consumption of mobile applications at design time based on the Generic Eclipse Modeling System. It allows developers to rapidly model potential app architectures and obtain automatic feedback on the architecture performance and power consumption.

The authors of ED49 propose MobiaModeler tool and framework that enables non-technical people to create their own domain-specific mobile apps. The MobiaModeler tool generates an app by first creating a model and later on transforming the model to platform-specific code through transformation tool component named Mobia processor. Study ED50 uses a tool suite called WebRatio for MDD of web and mobile applications. It supports developers in specifying domain model and user interaction model for the applications according to the extended versions of the OMG standard language called IFML. Finally, study ED52 attempts to review existing usability studies and subsequently proposes a usability model for conducting early usability evaluation for generated mobile apps.

4.2.2. RQ_2-$SubRQ_B$ What are the code generation steps? How is it accomplished?

Model Driven Development (MDD) for mobile apps uses models for an abstract representation of a system, and then uses suitable chains of transformations to refine the models into a final application, or part of it. This code generation step is thus used to generate code from a higher level model to create a working application. Most of the selected studies use forward engineering approaches for code generation. Fig. 9 summarizes our findings and more discussion on it is presented below.

A. Templates and filtering generation patterns: We found eight selected studies (ED1 ED6 ED10 ED21 ED27 ED30 ED34 ED37) create a subset of models from the source model. Then, templates are instantiated based on filtered source model values to generate the code. For example, study ED1 converts a map (ESRI ArcMap) into an XML file (SLD-file) for model validation, where code is generated based on XSL transformation scripts and using ArcGIS-Map to SLD converter tool. In the tool, ArcObjects performs this transformation (of an SLD document), acting as an input base for further modifications according to the user and context models. Study ED6 developed separate code generators for Android and iOS platforms based on the Xtend language. The Android code generation process produces two projects: (i) an Android project containing the Android app and (ii) an Android library project containing the data layer code. The Android library project is created by reusing an existing EMF generator that generates code for the EMF runtime. The EMF generator becomes a sub-generator of the complete code generator and processes an Ecore data model separately. Then, the process and GUI models are translated by sub-generators written in Xtend. The iOS code generator's workflow is nearly the same as for Android, except it creates one project, and it cannot reuse the EMF generator due to inapplicability on the iOS platform. The generated project must also be exported from Eclipse and imported into the XCode IDE. Similarly, ED21 used the Xtend expressions of multi-line models to write the platform-specific code generator.

The Xtext framework and Xtend2 have also been used in ED27 and ED37 for transformations and projections in templates, along with mapping rules to generate the app source codes. In ED30, developers need to model the business classes and their relationships using a graphical modeling tool named JBModel. JBModel transforms the application class diagram into Java classes augmented with annotations provided by the framework named JustBusiness, which generates persistence code, interfaces and Android app resources.

B. Templates and meta-model generation patterns: We found 14.55% of our selected studies (ED2 ED13 ED19 ED20 ED24 ED39 ED40 ED54) parsed source model to create instance of meta model for code generation using templates. The basic difference we found in this category compare to the previous category one is that these studies instantiated the templates using instance values from the original meta model not the filtered source model. For example, studies ED2 and ED20 convert AXIOM source models into code (Java for Android and Objective-C for iOs) based on a set of platform-specific templates. The Android app AXIOM model contains nodes that were mapped to specific items during implementation, such as project files, class files and resource files. It also includes information needed to populate each item to be generated. The task is to serialize the information stored in the abstract model trees (AMTs) into linear text files in the implementation. AXIOM's code generation algorithm accepts an AMT and produces native code. The generator is template-based and templates capture knowledge and information about both the programming language used and the API of the native SDK. Each code template contains a parametric code fragment and an injection point, the location where the code fragment can be inserted. This information, along with the injection descriptors from the implementation model, drives the code generation process. In contrast, ED19 proposed RUMO framework that provides a platform-independent definition of a UI backed by constraints in the form of rules. The final model is transformed into a platform-specific UI code that uses predefined distinct template files for each component to create source code for the desired platform. Similarly ED13, ED39, ED40 and ED54 takes the models, either PSM or Xpand reference implementation as input and uses statically typed templates to translate the model into source code.

C. API based generators: One-fifth of the selected studies (ED4 ED5 ED16 ED17 ED22 ED32 ED41 ED44 ED45 ED46 ED55) uses Grammar-based APIs and client programs to generate the code. In ED4, source code is generated from the AFL-based state diagram as follows: (i) Class diagram to structural Java code generation using the UJECTOR tool and (ii) UML state machine(s) to their equivalent behavioral code generation using the well-known state pattern. In state pattern, each state is transformed into a class inherited from the base state class. The events in the states are included as the methods in the specific classes. These methods are implemented in derived classes with their corresponding actions. All ALF actions are transformed according to the mobile platform-specific language (currently Android and Windows), where the controller design pattern is implemented on top of the generated codes. The controller pattern also allows the integration of the business logic with the user interface. Studies ED5, ED22, ED32 and ED55 all used a similar approach to automatically generate app source code from a class diagram model using Java, JUSE4Android, Eclipse, and Acceleo tools, respectively. The authors ED22 used the DB4O object-oriented database management system to avoid unnecessary transformations into the host language. Study ED45 follows the same approach to include gesture-based interaction in the UI.

Study ED16 extends the GenCode tool to generate Android code based on class and sequence diagrams. ED17 works similarly to ED16, where XMI code for a PIM is generated from an object diagram based on JDOM API. Then, this code is transformed into platform-specific code.

D. Inline code generation: We found six selected studies (ED3 ED7 ED9 ED15 ED31 ED49) that use inline code generation through a precompiler that modifies the program, which is then compiled or interpreted. Study ED3 generate code directly from a PIM using Java compiler. Here, PROTON's runtime engine accesses the input JSON file, loads and parses all the definitions, creates a thread for each input and output adapters, starts listening for events, and generates code for incoming events from the input adapters and forwards the events to output adapters. Similarly, in ED31, code generator was implemented in Java for transforming model instances into executable JS code. The generator makes use of the javax.xml parser package to create an object from the XMI file. The object is then parsed using the org.w3c.dom package. The result of the generation process is a JavaScript application.

E. Synthesizers, tools and framework: We found six studies (ED14 ED18 ED23 ED25 ED43 ED47) that use distinct but similar code generation techniques. For example, study ED14 implements code generator using the openArchitectureWare (oAW) generator framework and workflow file. An Apache Ant script triggers the oAW workflow. After generating the code, it builds and signs the application package. The example applications in ED14 used XML-RPC and WSDL described services. For the realization of the stub for accessing the WSDL described services via SOAP, kSOAP 2 has been used. For invoking XML-RPC operations, the android library has been utilized. ED18 also uses ANTR parser technology to translate their DSL into target app code.

F. Unknown: We could not find descriptions of the code generation techniques for sixteen studies (ED8 ED11 ED12 ED26 ED28 ED29 ED33 ED35 ED36 ED38 ED42 ED48 ED50 ED51 ED52 ED53), either because it was considered their future work or a detailed design and working procedures remain absent from the papers. For example, in ED8, the platform-specific code is generated based on an MNV description transformation. However, how it is achieved is not explained, and hence we categorize this to unresolved/unknown group. In contrast, the authors of ED26 considered supporting tools using their MIM diagram to generate app code as potential future work.

4.3. RQ3 : Which empirical methods are used in the selected studies to evaluate MDD based app development approaches, and what are the results obtained?

We expected that all selected studies would use empirical methods to evaluate their work and provide appropriate explanations. However, in seventeen of the selected studies, we did not find an empirical comparative analysis and related discussion. These studies present some experiential data/results, but how the results are measured/evaluated is not well explained, or is unclear i.e., unclear steps about how the experience results were obtained. Therefore, we grouped these seventeen studies under the 'experience results' subsection. Overall, analyzing the extracted data, we found several strengths and gaps in the selected studies, based on which we recommend future works in this domain to address the emerging trends.

4.3.1. RQ_3-$SubRQ_A$ How were the studies evaluated?

We identified how the main results in each primary study are evaluated. Fig. 10 summarizes the results of the four main themes that we identified during our analysis.

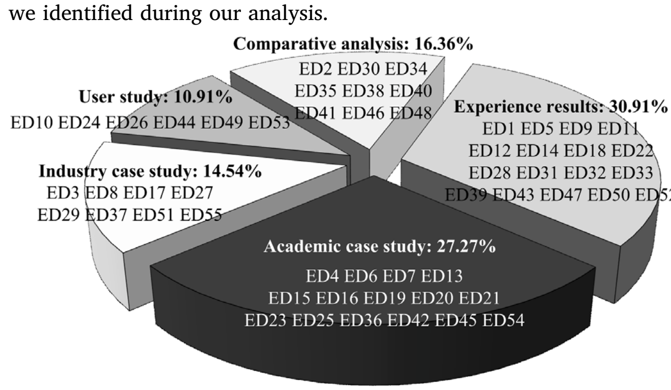


Fig. 10. Distribution of studies in terms of evaluation approaches.

A. Academic case study: We identified 27.27% of the selected primary studies evaluated their work with an academic case study. These studies demonstrate evaluation by developing "toy" apps or partial components of an app, and then explain their results applied to a specific platform, mostly Android. Table 10 presents the summarized evaluation strategy for these studies using three **Ev**aluation **Qu**estions (EvQu): (i) Is the presented work scalable — considering modification possibility, development time, and required knowledge both for user and developer? (ii) Does the presented work generate a complete output, or is a partial output generated? (iii) How is the work evaluated and how is the quality measured or compared? We use these three questions for the discussion of other groups shown in Fig. 10 in the following subsections. Table 10 shows that only one study (ED16) uses a reverse engineering approach to prove the efficiency of their approach. Two studies (ED4 and ED45) validate their work with quality criteria, three studies (ED6 ED25 ED45) include a user-study, and only one study considers the development of industry best-practice code

Table 10
Summarized evaluation results for the fifteen selected studies that use academic case studies.

Study	Eval Question 1 – is it scalable?	Eval Question 2 – complete output app?	Eval Question 3 – how is it evaluated / quality compared?
ED4	Yes: Supports multi-platform, generated output is modifiable.	Complete: Toy app for English words' learning game.	Validates approach, integrating with business logic, use-case is explained, quality of produced output measured through testing.
ED6	Yes: Designed full app variant, generated output is modifiable.	Complete: Toy apps for guiding conference participants and reminding TV show broadcast times.	Comprises a detailed domain analysis, illustrates how the given input model automatically fills data, layout, style, behavior, and service, presents a good user study.
ED7	Partially Yes: Same app is generated with native JAVA and ED7 method, incomplete GUI model.	Complete: Several toy apps are developed targeting game, personalization, education and analysis.	Showed how it reduces the development efforts, requires a smaller number of Lines Of Code (LoC), results are compared with Robolectric, Robotium, and Espresso benchmarks.
ED13	No: Toy game app.	Partial : Developer needs to implement logic and algorithms.	Complete: Use-case comparison with and without tools are shown in terms of development time measurement, animation performance measurement.
ED15	Yes: Generate products for different platform from the same model.	Partial: DSL to Code.	Compares performance with the existing tools in terms of LoC and features description, quality depends on the specified requirement.
ED16	Yes: Generates the same outputs as for existing app.	Complete: Popular snake game.	Proved efficiency, present reverse engineered case study for process and comparison.
ED19	Yes: Able to transform defined UI into a destination UI of any platform.	Partial: Only UI for a personal app.	Test set comprises a defined UI consisting of a view for different components and transformation is shown for iOS from Android.
ED20	Partially Yes: Supports multiplatform, but explanations and references are missing for use-cases.	Complete: Complete set of apps generated.	More than a hundred test apps are generated and evaluated in terms of LoC, produce industry best practice code.
ED21	Yes: Multiplatform extension is possible	Partial: Only GUI generated	Showed GUI of an E_Exam MCQ app code conversion, not compared with existing works.
ED23	No: Only GUI and it is incomplete.	Partial: Android scaffolding for producing flexible GUIs	Compare performance for GUI views, analyzed prototype production from GUI requirement.
ED25	Yes: Model transforms to Android app code and is modifiable.	Complete: Toy app for reading brightness and color changing based on sensor data.	Comparative result in terms of LoC, Number Of Attributes (NOA), Complexity, Weighted Methods for Class (WMC), Depth of Inheritance Tree (DIT), Lack of Cohesion of Methods (LCOM), coupling etc., include user study results.
ED36	No: Only considers UI and non-convertible.	Partial: Only UI for restaurant delivery application	Evaluate efficiency of network operations handling in the app UI.
ED42	Yes: Runtime modification is possible	Complete: Data collection tool for a field trip of biology students	Analyze development tool processing considering presentation models and dialog models.
ED45	No: Output is not modifiable	Partial: Only Gesture UI generated	Validated the method and supporting tools in two different types of applications.
ED54	Yes: Translatse abstract models into implementation artifacts for web, hybrid and desktop.	Complete: Field Force Automation toy app	Analyzed Skeleton is independent of any platform domain, having its central core based on model transformations.

(ED20) for performance measurement, evaluation, and analysis. The table also summarizes evaluation processes carried out in these studies. For example, we marked study ED4 as scalable since it produces a complete output (a working app) that can be modified, and the presented tool produces a quality output and the use cases for development stages are explained in detail. In addition, the 'Scramble' toy app it uses is an easily understandable and accessible example for the reader. The author also validates their approach by integrating this app with existing Business logic and applications. The scalability assumption is also valid for other studies shown in this table. Similarly, we consider the study ED20 as partially scalable because the presented framework can transform native code for cross-platform development. However, analysis and explanation remain absent for several use cases mentioned in it. In contrast, we marked study ED30 as unscalable because the UI is non-convertible and comparisons with other approaches are not shown.

B. Industry case study: We found that eight out of the fifty-five selected studies use an industry case study to evaluate their work. These studies demonstrate evaluation by developing a real-world app through a pilot project and analyzing the performance of the tool-set using different day to day development scenarios. Table 11 presents the summarized evaluation strategy for these studies using the same three questions used for Academic case study evaluation. Table 11 shows that only one study (ED3) validates their used model service, one study (ED8) compares the approach's performance with existing industry practices, and one study (ED17) evaluates the transformation process from GUI model to code for a real-world app. The remaining five studies assess their method and showed its scalability and reliability via various example uses cases.

In addition, a real-world industry project for essential functions of mobile telephony applications, including messages and top screens, are considered as an industry case-study (in ED8), which is easy to follow and understand. Study ED17 we considered as partially scalable since it only offers a graphical way to design GUIs in UML but the generated code is not modifiable. The study ED51 enables desktop and mobile environments integration in graphical modeling along with the architecture and necessary tools. Study ED51 also analyzed dynamic

Table 11
Summarized evaluation results for the eight selected studies that use industry case studies.

Study	Eval Question 1 – is it scalable?	Eval Question 2 – complete output app?	Eval Question 3 – how is it evaluated / quality compared?
ED3	Yes: Suitable for business users, supports top-down goal-oriented modifications.	Complete: Pilot app deployed in industry for mobile network fraud user detection.	Validate Used model services, analyses pilot project, quality is measured with produced outputs.
ED8	Yes: Stakeholders track requirements throughout the project life cycle.	Partial: UI architecture of a pilot industry project for mobile telephony app, MNV based layout.	Compares with UML state-charts and analyzes backtracking support.
ED17	Partially Yes: Offers an easy graphical GUI design.	Partial: GUI of a real-world app for mental health.	Not much detail, only evaluates transformation and case study.
ED27	Partially Yes: Abstract low-level boilerplate code generation for iOS app.	Partial: Registration page of an e-commerce app that contains a collection of elements, but incomplete.	Analyze iOS project model consistency, shows effectiveness over UML based modeling approaches.
ED29	Yes: Existing code integration, reference architecture for MD2.	Partial: Architecture of a commercial product app that takes geo-data and map-based technologies.	Showed applicability in data-driven app development, equivalence analysis for reference architectures and meta models.
ED37	Yes: Supports multiple platforms, generated output is modifiable.	Partial: GUI of a health app that used for posting queries anytime and qualified doctors answer it.	Analyzed platform independency.
ED51	Yes: Supports integration of desktop and mobile graphical modeling environments.	Complete: Parallel development of a factory app to control machinery, uses token for operation	Analyzes dynamic settings in architecture, evaluates real-world scenarios, assesses token-based collaboration.
ED55	Yes: Memory efficient for multiplatform development.	Partial: UI and functional core of a real-world Sudoku game app.	Analyzes implementation decisions, evaluates performance on real-world problems.

settings in the architecture and present several real-world scenarios for their tool evaluation. ED55 in contrast only evaluates performance problems and analyzes implementation decisions.

C. Evaluation based on user studies: We found that only 10.91% of the selected studies evaluate their work through user studies, i.e., students or practitioner usage, survey, interview and comments. Table 12 presents the summarized evaluation strategy for these six studies using the same three questions as above. From Table 12, we found that one study (ED10) carries out a Cognitive dimensions analysis, and one study (ED26) validates the reliability of the evaluation results. In this table, we mark all the studies as scalable except for ED26, because there is no supporting tool available for the used modeling construct. Like ED26, study ED49 also produces a partial output, but we marked this one as scalable since it enables non-technical developers to create their app. Interestingly, study ED53 performs a user study on 23 students. Although the participants have little experience in app development, at the end, more than half of the assigned tasks were completed in time bound assignments that showed the usability of their approach.

D. Comparative analysis for evaluation: We identified nine selected studies that evaluate their works through comparative analysis. These studies demonstrate evaluation by developing apps or components of an app and then compare the performance of the approach proposed. The comparison shown in these studies is similar to that of the academic case studies, however they differ in discussing the analysis results in their evaluation. Table 13 presents the summarized evaluation strategies for these nine studies. From Table 13, we see that two studies (ED2 and ED38) used a controlled user study mainly for performance analysis and comparison among technologies, hence we grouped here rather than in the user study section. We also found that two studies, ED38 and ED46, enable non-technical end-users to develop a real-world app. Study ED34 uses a set of eighteen apps for performance analysis, and ED48 discusses detailed SPOT evaluation results.

E. Experience results: We were unable to find evaluation techniques or analysis results for seventeen of the selected studies. These studies claimed that they use various use cases, but no proof is provided and some share only the author experiences. For example, in ED5, evaluation is missing; rather, it analyzes the code generation in different stages. However, how it is evaluated/analyzed is not shown, and hence we grouped it in the ‘experience results’ cluster. In contrast, the authors of ED39 and ED50 shared only their own experiences using their approach, but no comparison or evaluation is carried out.

4.3.2. RQ_3-$SubRQ_B$ *What are the strengths and limitations of the selected studies?*

This SLR tries to reveal strengths and limitations for the mobile apps development approaches that exclusively utilize Model-Driven Development (MDD) which are presented below.

Primary strengths of the selected studies: We found several advantages claimed in the selected studies, which we group into the following six categories, shown in Fig. 11(a): (A) Increase the abstraction level and enable model manipulation; (B) Productivity gains; (C) Raise Flexibility; (D) Support multi-platform, or different versions of the same platform development; (E) Increase automation in code generation; and (F) Contribute to efficiency increases. Moreover, we found some studies have multiple strengths and hence we present a Venn diagram in Fig. 12 to show their overlaps. We did not found any additional strength for the 30.91% studies (ED8 ED9 ED11 ED12 ED15 ED18 ED21 ED23 ED26 ED27 ED34 ED43 ED44 ED47 ED50 ED52 ED55) except their primary strength.

However, many of these claimed strengths are the authors’ opinions based on incomplete evidence rather than proof that can be measured based on application to real-world scenarios, since 80% of the studies are evaluated in academic environments and not tested in industrial cases. Further studies of the techniques on industrial-scale mobile app development problems would need to be carried out to substantiate these claimed strengths translate to real-world scenarios. A more detailed discussion is presented below.

A. Increase abstraction level: We identified six selected studies (ED2 ED3 ED8 ED34 ED46 ED51) that claimed to increase the abstraction level for target users with various methodologies and models. This abstraction aims to make the app development easier and faster. For

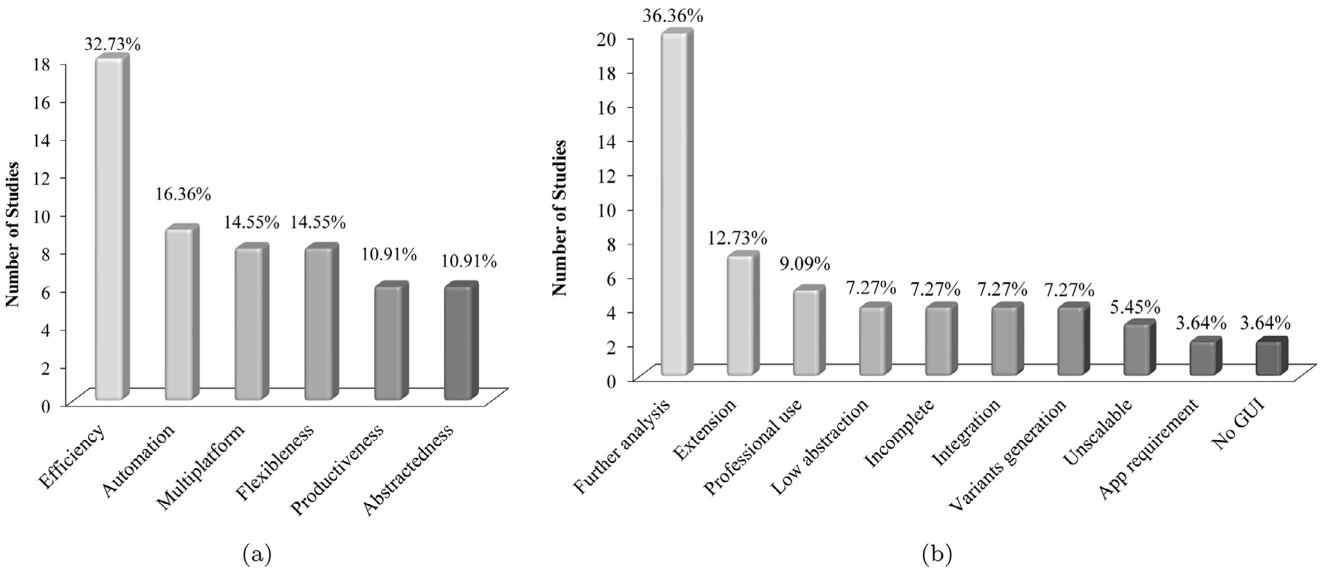


Fig. 11. Identified areas of (a) Primary strength and (b) Common limitations in the primary selected studies.

Table 12
Summarized evaluation results for the six selected studies that use user study.

Study	Eval Question 1 – is it scalable?	Eval Question 2 – complete output app?	Eval Question 3 – how is it evaluated / quality compared?
ED10	Yes: Visual modeling and synthesis in code generation.	Complete: Health app for obesity treatment plan and interfaces.	Cognitive dimensions analysis to assess languages and environment, user evaluation of entire approach.
ED24	Yes: Scaffolding mobile app, generated output is modifiable.	Partial: Commercial Prompa app and a data-driven app that displays movie data from API. Some generated code modification	Collect feedback from developers about use-cases, carried out a feasibility study.
ED26	No: No supporting tool for its modeling constructs.	Partial: Only GUI of the Gmail app generated.	Validated reliability, analyzed survey results, prove effectiveness through T-Test.
ED44	Yes: Output UIs are device and platform-independent.	Complete: Interactive survey questionnaires for the public transportation network.	User study for preferences, analyzed different data collection forms common in performing questionnaire surveys.
ED49	Yes: Enables non-technical people to create their own domain-specific app.	Partial: UIs and configurable components of the app.	Qualitative user study for performance evaluation, but verification or validation is missing.
ED53	Yes: Supports various stakeholders, Inter-operable apps from a common PIM.	Complete: General app for smartphones and smartwatches.	Training-based user study among IT students, analyzed usability score based on the System Usability Scale.

example, the authors of ED2 provide a DSL to express the model of the applications independently of platforms. Based on AXIOM and agile methods, they retain key elements of UML state-charts to represent the app behavior. The use of Groovy in ED2 for the modeling language facilitates the transformation of AXIOM PIM into PSM. These models are claimed by the authors to be fast to develop and easy to verify, making them compatible with agile. Their intent was to provide a sufficient cross-section of functionality in terms of the modeling notation and transformation tools to increase abstraction with good code quality. A similar abstraction is achieved in ED3 and ED8. The transformation of the event model to an event-driven app is considered in ED3, and ED8 manages complex functional requirement combinations in embedded software and improves app functionality.

B. Increase productivity: We found that 10.91% of the selected studies (ED9 ED10 ED29 ED39 ED40 ED48) contribute to productivity gains through supporting crosscutting behavior and minimizing development time. For example, authors of ED40 introduce a new approach to utilizing cloud resources for mobile users. Here heavyweight components of CMH app are developed on cloud-side, whereas lightweight or native code is developed in devices for execution. CMH applications execution does not need profiling, partitioning, and offloading processes, hence producing the least computation overhead on the target mobile devices. Study ED48 reduces production costs and time by enabling the app developers to understand the consequences of architectural decisions long before implementation. Similarly, the paradigm provided in ED9 modularizes crosscutting behavior by integrating aspect-oriented development techniques with MDD.

C. Increase flexibility: We identified eight out of fifty-five selected studies (ED12 ED14 ED18 ED25 ED35 ED38 ED43 ED44) that claim to make the app development process more flexible, or target different domain stakeholders, especially to manage model, data, and services. For example, ED12 presents a meta-model for defining context-aware applications which is different from the other selected studies in the sense that (i) It does not only model Web service descriptions, but also considers the faults in these descriptions, and (ii) It considers the inefficiencies of different Web services development techniques in generating data-type descriptions.

Study ED38 applies end-user programming techniques to ease the process of modeling data collection instruments. It offers an intuitive

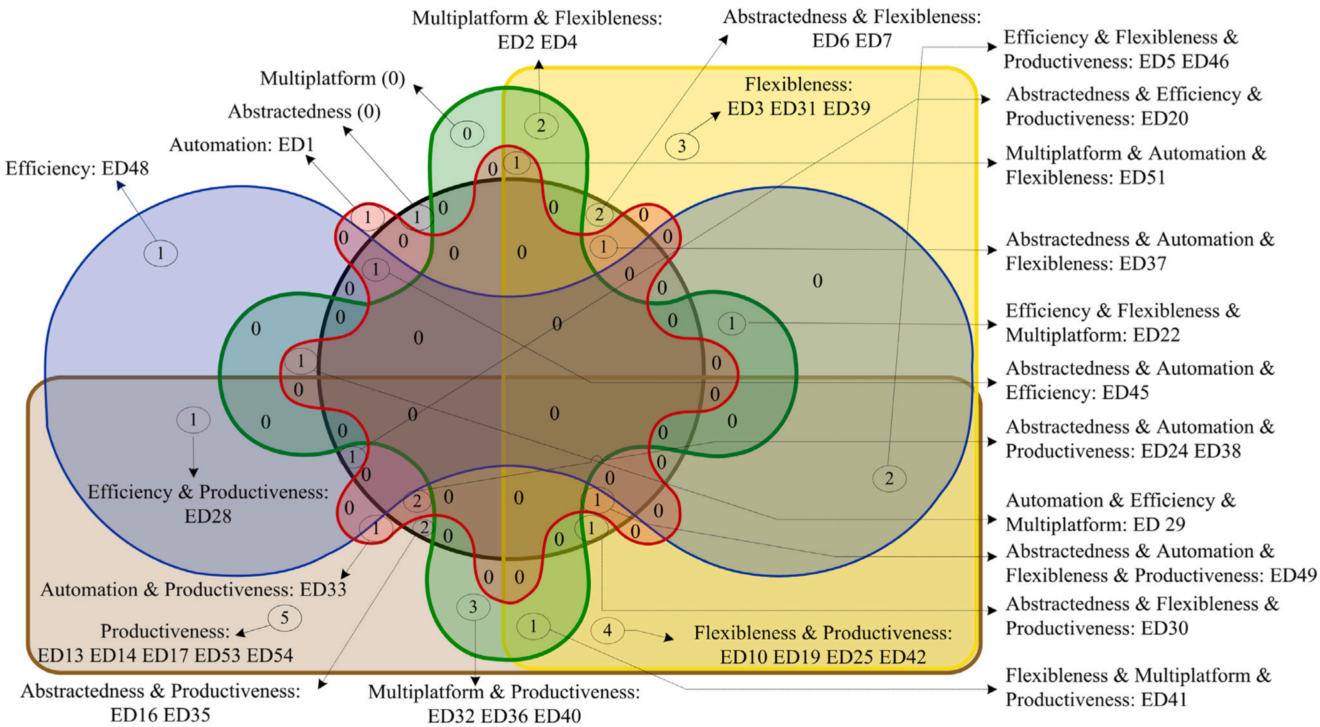


Fig. 12. Venn diagram representing additional strengths of the selected studies with corresponding overlap.

Table 13
Summarized evaluation results for the nine selected studies that use comparative analysis.

Study	Eval Question 1 – is it scalable?	Eval Question 2 – complete output app?	Eval Question 3 – how is it evaluated / quality compared?
ED2	Yes: Customizable model transformation, code is consistent with industry best-practice.	Complete: Toy app for broader security component that supports and manages associations of users to roles.	Feasibility study, proof-of-concept app using performance analysis for a set of apps, productivity measurement i.e., lines-of-code per person-hour.
ED30	Partially Yes: Product completion needs developer inputs.	Complete: HR management app for employee's overtime request and approval system.	Performance analysis between traditional and proposed approaches in terms of time, LoC, Number of Files (NoF), etc.
ED34	Yes: More consistent and transformable model.	Partial: Model to code for a set eighteen apps, mostly data-driven.	Object-oriented metrics based comparative study comprise measures for size and complexity, coupling, cohesion, and inheritance.
ED35	Yes: Supports business model transformation, rapid development by cloud service integration.	Partial: Business scenarios from a sales management system to avoid conflict for service usages and follow constraints.	Comparison between Data-, Role- and Process-driven patterns, performance analysis with other SaaS methods, feasibility study considering usability and adaptability.
ED38	Yes: Enables healthcare professionals to create data collection and sensing apps.	Partial: Process modeling using the framework.	Controlled study for performance comparison, inferential tests, evaluate associations between the performance measures and subjective complexity, correlations analysis.
ED40	Partially Yes: Hybrid, abstractions over cloud and mobile features.	Complete: Jobs tracking, Fetching and displaying time values, Contact list extraction apps.	Comparative analysis based on code metrics e.g., LoC and MVC.
ED41	Yes: Supports cross-platform development.	Complete: Real-world industry apps for insurance tariff calculation and library management.	Comparative analysis in terms of LoC, performance analysis in real-world scenarios.
ED46	Yes: Enables non-technical users to develop multi-platform apps.	Complete: Personalized app for research paper and profile update.	Comparative analysis in SUS ratings, controlled qualitative study.
ED48	Yes: Reduce power consumption, increase performance.	Complete: Car accident detection app.	SPOT performance analysis and discussion.

configurator component that allows researchers to create their instruments in a flexible, graphical manner. Similarly, ED25 includes a DSL for modeling contextual information and adaptation rules, transforming them into executable code. The middleware (DSL) provides the code required to deal with sensors and react and execute functionality according to contextual rules to improve the user experience. In ED35, the authors combined MDA ideas to promote SaaS application development based on semantic reasoning mechanisms.

D. Support cross-platform, multi-platform or multi-version app development: We identified 14.55% (eight out of fifty-five) selected studies (ED6 ED13 ED19 ED21 ED27 ED37 ED47 ED53) that aim to support either cross-platform, multi-platform or different versions of the same mobile app platform development. Most of these studies focused on reusable code component development to reduce product development time and cost. For example, ED13 describes MVC patterns to assist the agile development of a multi-platform mobile app. It uses a set of rules for each target platform to transform the UI to meet customer requirements quickly and adequately. Similarly, in ED19, UI is generated using the RUMO framework. ED6 also considers the generation of mobile app variants.

Study ED21 designs UIs for mobile and web applications based on components, for which a DSL is defined to generate native code for several platforms based on a textual model for the generation of graphic interfaces based on components. ED53 also considers similar cross-platform development, whereas ED27, ED37 and ED47 consider UI/GUI development and code generation. In ED27, the authors defined a platform-independent language as the base for generating native code for iOS apps. ED37 showed that the presented DSL defines GUI independently of the target mobile platforms and allows developers to generate native code to these several platforms automatically.

E. More automation for the app development: We identified nine out of fifty-five primary selected studies (ED5 ED17 ED20 ED22 ED23 ED28 ED31 ED32 ED36) aim to raise the use of automation in the app development process. For example, ED17 generates GUI source code in three steps: (i) The system GUI is modeled in class and object diagrams, (ii) The models are transformed into platform-independent XMI files using JDOM API4, and (iii) They adopt an MDA approach to transform the models into platform specific GUI code. Here, transformation rules are defined using ATL and the MDA based approach. Similarly, studies ED5, ED23 and ED30 automatically generate codes from UML class diagrams.

ED20 produces an app by transforming PIM to an implementation model, and finally to code. It uses AXIOM Abstract Model Tree for model representation to be the basis for all model transformations and code generation. ED31 intends to facilitate the development of the "Virtual Worlds" app, such as games, by visualizing scenes and characters construction. It directly uses Java to create DSL tools. Code is directly generated from this high-level language with a sound and maintainable architecture alike ED22, ED28 and ED32. The method of ED28 allows android application prototyping from WND, where certain portions of the GUI code are generated without providing a mechanism to exploit native capabilities of a smartphone.

F. Increase efficiency in app development: The remaining 32.73% studies claim to increase app development efficiency by proposing a new method, framework, tool or languages. For example, in ED1, the authors introduce a publicly available tool named ArcMap2SLDGenerator for efficient map designing for Web Map-Servers and mobile apps. Authors of ED7 develop MoDroid, a high-level modeling language that implements the Meta-Model and its supported tools for Android app development. In ED11, requirements for adaptable and user-centric mobile business processes have been studied. The advantage of this method is that the code generator does not generate just any generic code but instead generates code based on the user's specifications.

Model transformations of ED30 help developers focus on the app design rather than implementation issues. ED16 and ED49 present two similar modeling tools for app development using UML and UML 2.0 models. Study ED45 proposes gestUI for multi-stroke gesture-based UI development that defines multi-strokes gestures. The gestUI can create a gesture catalog model and support model transformations to obtain the source code, including gesture-based interaction. ED26 presents a method for modeling mobile interfaces based on MIM, as part of the future mobile development project. ED42 generates data collection applications using CAMELEON reference framework, where task models typically describe constraints on task execution as temporal relations between sub-tasks and decentralizes the development process. ED33 provides a modeling facility that generates code in JSON format on the IFML model. The approach of ED50 supports developers in domain model specification and the user interaction model for apps according to IFML discussed in ED33.

Limitations of the Selected Studies: We identified ten common kinds of limitations in the selected studies, summarized in Fig. 11(b).

A. No GUI development: Two studies ED4 and ED7 fall in this category. The proposed tool of ED4 (MAG) allows the developer to automatically generate the business logic code of the app from the app models, but the GUI of the app is developed separately. In ED7, input interfaces are declared using the methods for execution that depend on the localized nodes, but the GUI-based model is not implemented.

B. Unsupported requirements: We consider two studies ED34 and ED37 fall into this group. For example, study ED34 allows automatic model manipulation, but how the tool gets the target app requirements is not specified, whereas, in the ED37 approach, behavioral requirements are unsupported.

C. Large scale development: Three studies ED13, ED19, and ED42 face obvious scalability problems. For example, the templates and rules used in ED13 and ED19 for a platform are not extendable to other platforms. The data collection apps of ED42 produced by using the CAMELEON reference framework cannot be enhanced by developers.

D. Variant generation: We consider four studies ED5, ED6, ED18 and ED48 fall into this group. ED5 generates structural codes for a mobile application, but it restricts the mobile app generation for a specific platform (Android). Moreover, it does not support variant generation during modeling, i.e., domain-specific hardware and software features transformation are not substantiated and not reusable. Similarly, variants considered in studies ED6 and Ed18 have deficiencies in modeling both platform-specific and platform-independent features. Study ED6 also excludes the mechanisms for exploiting smartphone native capabilities, such as cameras and embedded sensors. The developed tool in ED48 (SPOT) automates power consumption emulation code generation, but does not process the proposed templates as intermediate representations of the platform variants. In addition, power consumption due to network access is too low-level consideration. These studies can easily specify automatic variants processing by formalizing and solving the integrated constraint sets to derive valid platforms.

E. Integration and interoperability: Four of the selected studies (ED11 ED30 ED40 ED47) have deficiencies in supporting physical elements integration or require manual interpretation. For example, ED11 does not provide mechanisms for integrating physical elements in the model or the following stages. Similarly, tools used in ED47 are not yet integrated, tailored towards J2ME and do not exploit the interoperability benefits of Web Services. Study ED30 needs a method code to be inserted manually. Cloud components of ED40 are not transferable due to their underlying heterogeneity. Moreover, isolating the development of mobile and cloud components like ED40 creates further versioning and integration challenges.

F. Development process completeness: We identified that the app development process in four selected studies (ED3 ED24 ED28 ED52) is partially complete. For example, ED24 uses rules to build models of native GUI code. These rules are stored in the system beforehand, but the rules database is not extendable. Hence, the approach of ED24 will fail when some models that have not been inside the system occur. The code generator also does not generate a ready-to-ship application as its primary features is to generate the database and then some higher-level code. This means that the user still needs to piece the code together to make it work as a full application. However, once the developer modifies the generated code, it cannot return back to the system/tool. Similar problems exist in ED28 and ED51 for the GUI code and app model, respectively. In ED3, the work is generally a model-driven development based on a spreadsheet. However, it does not generate analysis logic and app codes.

G. Low abstraction: We found four selected studies (ED9 ED21 ED27 ED43) need abstraction increases through proper use of MDD techniques. For example, the paradigm in ED9 provides a set of techniques to modularize aspect-oriented development techniques. The main disadvantage of this approach is the lack of support for high-level abstraction elements to express conceptual characteristics. Similarly, the patterns modeling languages of ED21, ED27 and refined mobile-specific interactions of ED43 do not provide much abstraction for tool users.

G. Professional development: We found 9.09% of the primary selected studies (ED10 ED16 ED20 ED44 ED46) are not suitable for use in the professional level. For example, pattern modeling language considered in ED20 is unsuitable for software developers as the input model is built using Groovy. Similarly, the MAML framework shown in ED46 is not suited for use by a professional mobile app developer because it does not contains much customization support for addressing complex issues during modeling. Experts must implement text corrections or validations during code generation in ED44, and the generated code in ED10 is not modifiable at all. ED16 does not generate optimized code, nor take into account good practices, potentially creating performance or security issues in the generated code.

I. Extension: Seven of the selected studies (ED2 ED12 ED33 ED35 ED38 ED45 ED53) have not examined how their approaches can be extended to other domains. In ED2, the authors claim that there is no practical limit to the overarching AXIOM approach, but they also say that extension to web apps needs to be examined in the future. In ED33, creating a database of already created UI elements to ensure their reuse was not considered. ED35 does not consider the rule extension, same as ED45 for the gesture, ED38 for recommendation criteria in the tool, and ED53 for supporting devices other than touchscreens.

J. Require further explanation, empirical tests, and evaluations: We believe that many selected studies (ED1 ED8 ED14 ED15 ED17 ED22 ED23 ED25 ED26 ED29 ED31 ED32 ED36 ED39 ED41 ED49 ED50 ED51 ED54 ED55) need to analyze their work more thoroughly and better explain the development processes proposed. For example, In ED1, the authors show that it is possible to adapt **G**eographic **I**nformation (GI) services dynamically to context and user properties in general. How to achieve optimal results requires further empirical tests, evaluations, and theoretical work. For example, what parameters to choose, how to weigh them, and what types of adaptation to realize to get the expected outcome still need more experiments. This is also true for nineteen other works grouped in this category. For example, ED25 tries to improve user experiences. This approach was tested only using the Android Dalvik VM runtime environment.

In ED26, MIM is used to specify characteristics of the final UI of an app. However, there is no standard procedure for confirming the completeness of GUI specified by MIM. Two gaps we identified in ED29 are: (i) It focuses on object structure and behavior, and not on interaction; (ii) The employed top-down approach does not adequately consider platform-specific features. ED31 directly uses Java to create DSL tools. Compared to the plain programming approach, this can be more productive and easier to adopt changes. However, more test results should be presented, especially for the end-users. Proposal of ED32 presents comparatives with at least two output platforms, but in some cases, the approaches were described in only one platform, similar to ED36, where more detail about code generator with feasibility study should be presented. In ED39, it is not clear whether the presented DSL has a static type system or not. All the works categorized in this group also lack consideration of real-world scenarios in their evaluation, adoption in the current environment, and the current progress in the domain.

4.3.3. *RQ_3-SubRQ_C What are our recommendations for future work in this area?*

We suggest several recommendations for future research into mobile app development based on model driven development techniques. These recommendations are based on the common identified gaps and proposed future work found in the fifty-five selected studies of this SLR.

A. App requirements modeling: The majority of the studies have applied model driven development to the design and implementation phases of mobile app development, but we found that the requirements phase is missing in 20% of the selected studies. This finding is interesting in the sense that the requirements engineering community has been modeling requirements for many years, but we found no mention of app requirement capture or modeling in these eleven selected studies. In addition, only two studies explicitly considered modeling of **N**on-**F**unctional **R**equirements (NFR), and 23.63% studies partially apply MDD in all development phases. Although it is possible to develop a fully functional mobile application only with functional requirements, considering NFRs may better increase the reliability, performance, scalability, useability, and security of the target apps. Therefore, the researchers, development community, and stakeholders need to pay significant attention to apply MDD in the entire app development life-cycle and address NFRs in-app generations.

B. Logic and presentation: The custom MDD based solutions dedicated to mobile app development are mostly concerned with the following two aspects: (i) UI/UX, e.g., how end-users interact with the application and what they see?; and (ii) Business logic, e.g., information parsing, data manipulation, API calls, etc. However, aspects related to the content and data layer need more attention as decisions related to these layers have a significant impact on the overall app performance. For example, issues such as where should the data source reside? how will communication takes place between the app and data source? how does the app deal with hardware issues? These are generally not handled by most MDD approaches proposed and hence further investigation is required.

C. Use of artificial intelligence and machine learning: None of the selected studies considered integrating **A**rtificial **I**ntelligence (AI) and **M**achine **L**earning (ML) techniques with MDD based solutions for app development. This is a promising area, especially for data processing, decision-making, and use cases. For example, a health app model powered with AI/ML, might be able to analyze its data more appropriately. Hence, relevant up-to-date recommendations from AI and ML components for the solutions to be used in the app could be suggested. This is also true for apps related to e-commerce, retail, game, entertainment etc. MDD approaches supporting AI and ML aspects of mobile apps could include modeling these intelligent aspects of apps, reusing existing AI and ML algorithms in generated apps, and using AI/ML techniques as part of the app development e.g. evolutionary techniques to generate parts of apps.

D. Code quality and target tool users: The majority of the studies have proposed new MDD based tools/languages to provide most of the code generation and theoretically offer higher productivity gains. However, these tools often have a poor reputation among developers due to their limited flexibility and sometimes poor code quality output. Rarely we were able to find a complete and good quality output was a focus of the development and evaluation. We found only one tool (ED24) that was dedicated for experienced mobile app developers, and only two methods (ED38 and ED49) were exclusively targeted for use by non-technical developers.

E. Human centric issues: None of the studies considered issues of end users of the apps with widely differing ages, languages, culture, accessibility issues etc. Different end user groups may need different approaches in their apps for interaction, explanation, and presentation of information to address these human issues. **H**uman-**C**entric

Issues (HCIs) appear to have had little attention to date by researchers into MDD based approaches for mobile app development. For example,two studies (ED10 and ED38) explicitly designed for health care applications generation, targeting diverse end-users. However, these approaches did not provide any mechanism to address (model) the need for diverse users, e.g., elderly users, users with physical and mental challenges, and users with different languages, cultures, and socioeconomic backgrounds. Failure to incorporate such HCIs into app modeling can generate an app that is unsuitable for whom it is designed.

F. Native, cross-platform, web and hybrid apps: Most attention has been paid to producing native mobile apps development for a single platform. Almost every approach relies on its own DSL, either defined from scratch or by enhancing an existing one. However, no specific standard has been devised for the mobile app modeling domain, and hence, no advantages offered by standardization is leveraged. Only just over 20% of the examined approaches target the development of cross-platform apps or target at least two different app platforms. In this case, a distinct code generator was used for each distinct platforms. This itself introduces serious maintainability issues for the approach as platforms evolve. We also found very few approaches investigating hybrid and web-based techniques in this domain.

G. Reliability and scalability of studies: Relatively little work has been done to prove the reliability and scalability of the proposed approaches. More than half of the studies do not report any validation method to ensure the appropriateness of their solution. Overall, we found that the details for feasibility studies and proof of scalability of the solution are missing for most of the selected studies.

5. Threats to validity

This SLR is subject to standard search and selection bias threats. We counter this threat by searching the most commonly used databases in the SE and IT context. We modified our search strings several times during the automatic search to maximize the number of relevant articles that match the SLR concepts defined in Table 2. We also kept our search string generic to search through the titles, abstracts, keywords and full text of an article to cover the maximum number of relevant papers. We have also conducted a manual search on Google Scholar to complement the automatic search using a snowballing strategy. All these together covered more than a thousand relevant publications and resulted in a broad set of original papers. We did not extensively search on relevant journals, proceedings of relevant conferences, and books related to MDD as we believe the search in the electronic database covered will this. However, we did include the option to search for book chapters while performing an automatic search. This process allowed us to find some book chapters, but only one of them got into our final selected paper set that is leveraged for data extraction after applying inclusion, exclusion and quality criteria filtering. Although we found more than a thousand potentially relevant articles during our automatic and manual searches, only 5% of these papers met our paper selection criteria. To mitigate the paper remotion risk, we cross-checked and discussed several times among the authors before excluding a paper from the final list. Moreover, predefined review protocols with detailed inclusion and exclusion criteria helped us reducing bias in selecting primary studies.

The results of this SLR paper are based on the data extracted and synthesized from the selected MDD-based mobile app development studies. We applied several quality criteria (shown in Sections 3.2.6 and 3.2.7) to estimate the quality of the selected primary studies. Even though the proposed criteria are not too strict, applying them indeed caused several initially selected papers to be excluded. To mitigate the risk of missing important data from the primary studies, we put back the excluded papers closely related to the primary studies. Eventually, we re-selected two papers to be included in the final set of primary selected studies for data extraction and analysis. Moreover, the paper list was gathered in early 2020, and there may be papers published after our search, which are not included here. Moreover, we still have a risk of producing biased results addressing only expert needs as the people involved in these processes have extensive experience in this domain. We provided detailed documentation on the searching, study filtration, and result analysis process to counter this issue. The results and recommendations for this SLR were prepared to help the reader identifying the scope and opportunities in MDD based mobile app development. To this end, we ignored those focusing only on testing and test case generation for mobile apps. We also ignored making recommendations in the area of mobile app development that exclude MDD approaches.

6. Conclusion

To better understand the research done to advance MDD-based approaches for mobile app development, we conducted a Systematic Literature Review (SLR). To this end, extracting data, analyzing them based on our three main research questions and corresponding eight sub-research questions are defined in the SLR protocol. We also identified the popularity of different applied MDD techniques, supporting tools, artifacts and evaluation techniques. This review study found that the existing MDD techniques for mobile application development are helpful in general. The primary strengths of the selected studies are categorized into six areas – Support for abstraction, Productivity, Flexibility, Multi-platform development, Process automation and Greater efficiency. We also found ten common limitation groups – No GUI development, Unsupported requirements, Lack of scalability, Problem in the variant generation, Limited integration and interoperability, Development process incompleteness, Need for further abstractions, Not suitable for professional development, Lack of domain extension, and Lack of empirical tests and theories. These identified gaps helped us to recommended seven high priority potential future research areas in this domain — including Better app requirement modeling, Greater logic and presentation Support, Support for artificial intelligence and machine learning in apps and app development, Better quality and more comprehensive code generation, Flexibility and output mapping, Better support for human-centric issues for diverse end-users, Integrate cross-platform, web and hybrid app development, and Increase support for reliability and scalability.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

Md. Shamsujjoha is supported by Monash International Tuition Scholarship (MITS), RTP Stipend and CSIRO Data61 Top-up Scholarship for his Ph.D. study at Monash University, Melbourne, Australia.

This work was also supported by the Australian Research Council (ARC) under a Laureate Fellowship project FL190100035, a Discovery Early Career Researcher Award (DECRA) project DE200100016, and a Discovery project DP200100020.

Appendix A. List of selected studies

[ED1] A. Zipf, Using styled layer descriptor (SLD) for the dynamic generation of user- and context-adaptive mobile maps a technical framework, in: Proceedings of the 5th International Conference on Web and Wireless Geographical Information Systems, 2005, p. 183–193. doi:10.1007/11599289_16.

[ED2] X. Jia, C. Jones, Cross-platform application development using AXIOM as an agile model-driven approach, in: Proceedings of the 7th International Conference Software and Data Technologies, Vol. 411, 2013, p. 36–51. doi:10.1007/978-3-642-45404-2_3.

[ED3] O. Etzion, F. Fournier, I. Skarbovsky, B. von Halle, A model driven approach for event processing applications, in: Proceedings of the 10th ACM International Conference on Distributed and Event-Based Systems, 2016, p. 8192. doi:10.1145/2933267.2933268.

[ED4] M. Usman, M. Z. Iqbal, M. U. Khan, A model-driven approach to generate mobile applications for multiple platforms, in: Proceedings of the 21st Asia-Pacific Software Engineering Conference - Volume 01, 2014, p. 111–118. doi:10.1109/APSEC.2014.26.

[ED5] H. Benouda, M. Azizi, R. Esbai, M. Moussaoui, MDA approach to automate code generation for mobile applications, in: Mobile and Wireless Technologies, 2016, p. 241–250. doi:10.1007/978-981-10-1409-3_27.

[ED6] S. Vaupel, G. Taentzer, R. Gerlach, M. Guckert, Model-driven development of mobile applications for Android and iOS supporting role-based app variability, Softw. Syst. Model. 17 (1) (2018) 35–63. doi:10.1007/s10270-016-0559-4.

[ED7] M. Jaber, Y. Falcone, K. Dak-Al-Bab, J. Abou-Jaoudeh, M. El-Katerji, A high-level modeling language for the efficient design, implementation, and testing of Android applications, International Journal on Software Tools for Technology Transfer 20 (1) (2018) 1–18. doi:10.1007/s10009-016-0441-2.

[ED8] J. S. Lee, H. S. Chae, Domain-specific language approach to modelling UI architecture of mobile telephony systems, IEE Proceedings - Software 153 (6) (2006) 231–240. doi:10.1049/ip-sen:20060022.

[ED9] A. Carton, S. Clarke, A. Senart, V. Cahill, Aspect-oriented model-driven development for mobile context-aware computing, in: First International Workshop on Software Engineering for Pervasive Computing Applications, Systems, and Environments, 2007, p. 5–5. doi:10.1109/SEPCASE.2007.3.

[ED10] A. Khambati, J. Grundy, J. Warren, J. Hosking, Model-driven development of mobile personal health care applications, in: Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering, 2008, p. 467–470. doi:10.1109/ASE.2008.75.

[ED11] A. Ruokonen, L. Pajunen, T. Systa, On model-driven development of mobile business processes, in: Proceedings of the Sixth International Conference on Software Engineering Research, Management and Applications, 2008, p. 59–66. doi:10.1109/SERA.2008.30.

[ED12] C. Taconet, Z. Kazi-Aoul, Context-awareness and model driven engineering: Illustration by an e-commerce application scenario, in: 2008 Third International Conference on Digital Information Management, 2008, p. 864–869. doi:10.1109/ICDIM.2008.4746829.

[ED13] Y. Choi, J.-S. Yang, J. Jeong, Application framework for multi platform mobile application software development, in: 11th International Conference on Advanced Communication Technology, Vol. 01, 2009, p. 208–213. https://ieeexplore.ieee.org/document/4809935

[ED14] M. Feldmann, V. Oleniuk, L. Globa, A. Schill, Overview of a model-to-code transformation approach for generating service-based interactive applications for google Android, in: 19th International Crimean Conference Microwave Telecommunication Technology, 2009, p. 362–364. URL https://ieeexplore.ieee.org/document/5293090

[ED15] A. Manjunatha, A. Ranabahu, A. Sheth, K. Thirunarayan, Power of clouds in your pocket: An efficient approach for cloud mobile hybrid application development, in: IEEE Second International Conference on Cloud Computing Technology and Science, 2010, p. 496–503. URL https://ieeexplore.ieee.org/document/5708492

[ED16] A. G. Parada, L. B. De Brisolara, A model driven approach for Android applications development, in: Brazilian Symposium on Computing System Engineering, 2012, p. 192–197. doi:10.1109/SBESC.2012.44.

[ED17] A. Sabraoui, M. E. Koutbi, I. Khriss, GUI code generation for Android applications using a MDA approach, in: IEEE International Conference on Complex Systems, 2012, p. 1–6. doi:10.1109/ICoCS.2012.6458567.

[ED18] J. Schafer, D. Klein, Implementing situation awareness for car-to-x applications using domain specific languages, in: IEEE 77th Vehicular Technology Conference, 2013, p. 1–5. doi:10.1109/VTCSpring.2013.6692589.

[ED19] A. Schuler, B. Franz, Rule-based generation of mobile user interfaces, in: 10th International Conference on Information Technology: New Generations, 2013, p. 267–272. doi:10.1109/ITNG.2013.43.

[ED20] C. Jones, X. Jia, The AXIOM model framework: Transforming requirements to native code for cross-platform mobile applications, in: 9th International Conference on Evaluation of Novel Approaches to Software Engineering, 2014, p. 1–12. doi:10.5220/0004882100260037.

[ED21] M. Lachgar, A. Abdali, Generating Android graphical user interfaces using an mda approach, in: 3rd IEEE International Colloquium in Information Science and Technology, 2014, p. 80–85. doi:10.1109/CIST.2014.7016598.

[ED22] L. P. d. Silva, F. Brito e Abreu, A MDE generative approach for mobile business apps, in: 9th International Conference on the Quality of Information and Communications Technology, 2014, p. 312–317. doi:10.1109/QUATIC.2014.50.

[ED23] L. P. d. Silva, F. Brito e Abreu, Model-driven GUI generation and navigation for Android BIS apps, in: Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development, 2014, p. 400–407. doi:10.5220/0004715504000407.

[ED24] S. Barnett, R. Vasa, J. Grundy, Bootstrapping mobile app development, in: IEEE/ACM 37th IEEE International Conference on Software Engineering, Vol. 2, 2015, p. 657–660. doi:10.1109/ICSE.2015.216.

[ED25] P. A. de Sousa Duarte, F. M. Barreto, F. A. de Almada Gomes, W. V. de Carvalho, F. A. M. Trinta, Critical: A configuration tool for context aware and mobile applications, in: IEEE 39th Annual Computer Software and Applications Conference, Vol. 2, 2015, p. 159–168. doi:10.1109/COMPSAC.2015.91.

[ED26] S. Geiger-Prat, B. Marín, S. España, G. Giachetti, A GUI modeling language for mobile applications, in: IEEE 9th International Conference on Research Challenges in Information Science, 2015, p. 76–87. doi:10.1109/RCIS.2015.7128866.

[ED27] M. Lachgar, A. Abdali, DSL and code generator for accelerating iOS apps development, in: 3rd World Conference on Complex Systems, 2015, p. 1–8. doi:10.1109/ICoCS.2015.7483269.

[ED28] T. Channonthawat, Y. Limpiyakorn, Model driven development of Android application prototypes from windows navigation diagrams, in: International Conference on Software Networking, 2016, p. 1–4. doi:10.1109/ICSN.2016.7501929.

[ED29] S. Evers, J. Ernsting, T. A. Majchrzak, Towards a reference architecture for model-driven business apps, in: 49th Hawaii International Conference on System Sciences, 2016, p. 5731–5740. doi:10.1109/HICSS.2016.708.

[ED30] F. Freitas, P. H. M. Maia, Justmodeling: An MDE approach to develop Android business applications, in: VI Brazilian Symposium on Computing Systems Engineering, 2016, p. 48–55. doi:10.1109/SBESC.2016.016.

[ED31] M. Stürner, P. Brune, Virtual worlds on demand? Model-driven development of JavaScript-based virtual world UI components for mobile apps, in: 4th International Conference on Model-Driven Engineering and Software Development, 2016, p. 648–655. URL https://ieeexplore.ieee.org/document/7954416

[ED32] H. Benouda, M. Azizi, M. Moussaoui, R. Esbai, Automatic code generation within MDA approach for cross-platform mobiles apps, in: First International Conference on Embedded Distributed Systems, 2017, p. 1–5. doi:10.1109/EDIS.2017.8284045.

[ED33] C. Bernaschina, S. Comai, P. Fraternali, Online model editing, simulation and code generation for web and mobile applications, in: IEEE/ACM 9th International Workshop on Modelling in Software Engineering, 2017, p. 33–39. doi:10.1109/MiSE.2017.1.

[ED34] E. E. Thu, N. Nwe, Model driven development of mobile applications using Drools knowledge-based rule, in: IEEE 15th International Conference on Software Engineering Research, Management and Applications, 2017, p. 179–185. doi:10.1109/SERA.2017.7965726.

[ED35] H. Cai, Y. Gu, A. V. Vasilakos, B. Xu, J. Zhou, Model-driven development patterns for mobile services in cloud of things, IEEE Transactions on Cloud Computing 6 (3) (2018) 771–784. doi:10.1109/TCC.2016.2526007.

[ED36] S. Rehman, R. M. K. Ullah, S. Tanvir, F. Azam, Development of user interface for multi-platform applications using the model driven software engineering techniques, in: IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference, 2018, p. 1152–1158. doi:10.1109/IEMCON.2018.8615013.

[ED37] A. Sabraoui, A. Abouzahra, K. Afdel, M. Machkour, MDD approach for mobile applications based on DSL, in: International Conference of Computer Science and Renewable Energies, 2019, p. 1–6. doi:10.1109/ICCSRE.2019.8807572.

[ED38] J. Schobel, T. Probst, M. Reichert, M. Schickler, R. Pryss, Enabling sophisticated lifecycle support for mobile healthcare data collection applications, IEEE Access, 7 (2019) 61204–61217. doi:10.1109/ACCESS.2019.2916142.

[ED39] H. Behrens, MDSD for the iPhone: Developing a domain-specific language and IDE tooling to produce real world applications for mobile devices, in: Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion, 2010, p. 123–128. doi:10.1145/1869542.1869562.

[ED40] A. H. Ranabahu, E. M. Maximilien, A. P. Sheth, K. Thirunarayan, A domain specific language for enterprise grade cloud-mobile hybrid applications, in: Proceedings of the Compilation of the Co-Located Workshops on DSM'11, TMC'11, AGERE! 2011, AOOPES'11, NEAT'11, & VMIL'11, SPLASH '11 Workshops, 2011, p. 77–84. doi:10.1145/2095050.2095064.

[ED41] H. Heitkötter, T. A. Majchrzak, H. Kuchen, Cross-platform model-driven development of mobile applications with MD^2, in: Proceedings of the 28th Annual ACM Symposium on Applied Computing, 2013, p. 526–533. doi:10.1145/2480362.2480464.

[ED42] A. Dittmar, M. Kühn, P. Forbrig, A domain-specific model-based design approach for end-user developers, in: Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems, 2014, p. 161–166. doi:10.1145/2607023.2610275.

[ED43] M. Franzago, H. Muccini, I. Malavolta, Towards a collaborative framework for the design and development of data-intensive mobile applications, in: Proceedings of the First International Conference on Mobile Software Engineering and Systems, 2014, p. 58–61. doi:10.1145/2593902.2593917.

[ED44] M. Kühn, P. Forbrig, Mobile data collection forms based on DSLs with different levels of abstraction, in: Proceedings of the International Conference on Multimedia, Interaction, Design and Innovation, 2014, p. 1–8. doi:10.1145/2643572.2643588.

[ED45] O. P. González, S. España, O. Pastor, Including multi-stroke gesture-based interaction in user interfaces using a model-driven method, in: Proceedings of the XVI International Conference on Human Computer Interaction, 2015, p. 1–8. doi:10.1145/2829875.2829931.

[ED46] C. Rieger, Business apps with MAML: A model-driven approach to process-oriented mobile app development, in: Proceedings of the Symposium on Applied Computing, 2017, p. 1599–1606. doi:10.1145/3019612.3019746.

[ED47] J. Dunkel, R. Bruns, Model-driven architecture for mobile applications, in: Proceedings of the 10th International Conference on Business Information Systems, 2007, p. 464–477.

[ED48] C. Thompson, J. White, B. Dougherty, D. C. Schmidt, Optimizing mobile application performance with model–driven engineering, in: IFIP International Workshop on Software Technolgies for Embedded and Ubiquitous Systems, 2009, p. 36–46. doi:10.1007/978-3-642-10265-3_4.

[ED49] F. Balagtas-Fernandez, M. Tafelmayer, H. Hussmann, Mobia modeler: Easing the creation process of mobile applications for non-technical users, in: Proceedings of the 15th International Conference on Intelligent User Interfaces, 2010, p. 269–272. doi:10.1145/1719970.1720008.

[ED50] R. Acerbis, A. Bongio, M. Brambilla, S. Butti, Model-driven development based on OMG's IFML with webratio web and mobile platform, in: Proceedings of the 15th International Conference on Engineering the Web in the Big Data Era - Volume 9114, 2015, p. 605–608. doi:10.1007/978-3-319-19890-3_39.

[ED51] D. Vaquero-Melchor, A. Garmendia, E. Guerra, J. de Lara, Domain-specific modelling using mobile devices, in: International Conference on Software Technologies, 2017, p. 221–238. doi:10.1007/978-3-319-62569-0_11.

[ED52] L. B. Ammar, A usability model for mobile applications generated with a model-driven approach, International Journal of Advanced Computer Science and Applications 10 (2) (2019) 140–146 . doi:10.14569/IJACSA.2019.0100218.

[ED53] C. Rieger, H. Kuchen, A model-driven cross-platform app development process for heterogeneous device classes, in: Proceedings of the 52th Hawaii International Conference on System Sciences, 2019, p. 1–10. doi:10.24251/HICSS.2019.894.

[ED54] A. Nestor Ribeiro, C. Rogério Araújo, An automated model based approach to mobile UI specification and development, in: Proceedings, Part I, of the 18th International Conference on Human-Computer Interaction. Theory, Design, Development and Practice - Volume 9731, 2016, p. 523–534. doi:10.1007/978-3-319-39510-4_48.

[ED55] Y. Cheon, A. Barua, Model driven development for Android apps, in: Proceedings of the International Conference on Software Engineering Research and Practice, The Steering Committee of The World Congress in Computer Science, Computer . . . , 2018, p. 17–22. URL https://csce.ucmss.com/cr/books/2018/LFS/CSREA2018/SER3296.pdf

Appendix B. Individual quality assessment scores for selected studies

No	Q1	Q2	Q3	Q4	Q5	Q6	No	Q1	Q2	Q3	Q4	Q5	Q6
ED1	1	2	2	3	3	2	ED29	3	2	3	2	2	3
ED2	5	5	4	4	4	2	ED30	5	5	5	4	3	3
ED3	3	5	4	5	3	4	ED31	3	3	3	3	2	3
ED4	5	5	5	5	3	3	ED32	3	2	2	2	2	1
ED5	5	3	3	3	2	2	ED33	4	3	2	2	1	3
ED6	5	5	5	5	4	5	ED34	4	4	3	4	2	3
ED7	5	5	5	4	3	4	ED35	4	5	5	5	3	5
ED8	2	4	4	3	2	4	ED36	4	2	2	2	1	1
ED9	5	4	2	2	2	1	ED37	3	3	3	2	1	1
ED10	5	4	5	4	4	4	ED38	5	4	5	4	3	5
ED11	2	3	2	3	2	3	ED39	4	3	2	2	2	3
ED12	3	3	3	3	3	2	ED40	4	3	2	2	1	3
ED13	4	4	4	4	2	3	ED41	5	5	5	5	3	5
ED14	3	3	3	2	2	2	ED42	3	3	5	4	2	4
ED15	2	4	3	3	3	3	ED43	3	3	3	2	1	3
ED16	5	3	3	3	3	2	ED44	2	3	3	2	2	2
ED17	4	4	4	4	2	1	ED45	2	5	4	4	2	4
ED18	1	3	3	3	1	4	ED46	4	3	3	4	2	3
ED19	3	4	4	4	2	1	ED47	2	3	3	3	1	2
ED20	5	5	2	5	3	4	ED48	1	4	3	3	1	3
ED21	4	3	3	3	2	2	ED49	4	3	3	2	3	3
ED22	3	2	2	2	3	2	ED50	4	1	1	2	1	3
ED23	3	2	2	2	3	3	ED51	4	4	3	3	1	4
ED24	5	4	4	5	4	5	ED52	2	2	2	2	2	2
ED25	5	5	5	4	2	3	ED53	4	4	3	4	2	3
ED26	3	5	5	4	3	3	ED54	4	3	2	3	2	2
ED27	3	3	3	3	2	1	ED55	4	3	2	3	2	4
ED28	5	4	4	3	1	1							

References

[1] M. Shamsujjoha, J. Grundy, L. Li, H. Khalajzadeh, Q. Lu, Human-centric issues in ehealth app development and usage: A preliminary assessment, in: 28th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER, IEEE, 2021, pp. 506–510, http://dx.doi.org/10.1109/SANER50967.2021.00055.

[2] A. Holst, Smartphone Users Worldwide 2016-2021, Tech. rep., Statista, 2019, Available at https://www.statista.com/statistics/330695, Accessed: Dec-2020.

[3] S. Felgoise, App Economy Stats You Should Know, Tech. rep., IronSour-rce, 2019, Available at https://www.ironsrc.com/blog/app-economy-stats-you-should-know, Accessed: Dec-2020.

[4] M. Shamsujjoha, J. Grundy, L. Li, H. Khalajzadeh, Q. Lu, Checking app behavior against app descriptions: What if there are no app descriptions?, in: 29th IEEE/ACM International Conference on Program Comprehension, ICPC 2021, IEEE, 2021, pp. 422–432, http://dx.doi.org/10.1109/ICPC52881.2021.00050.

[5] J. Clement, Total Global Mobile App Revenues 2014-2023, Tech. rep., Statista, 2019, Available at https://www.statista.com/statistics/269025/worldwide-mobile-app-revenue-forecast, Accessed: Dec-2020.

[6] J. Grundy, M. Abdelrazek, M.K. Curumsing, Vision: Improved development of mobile ehealth applications, in: Int. Conf. on Mobile Software Engineering and Systems, MOBILESoft, 2018, pp. 219–223, http://dx.doi.org/10.1145/3197231.3197263.

[7] B. Selic, What will it take? A view on adoption of model-based methods in practice, Softw. Syst. Model. 11 (4) (2012) 513–526, http://dx.doi.org/10.1007/s10270-012-0261-0.

[8] A.W. Brown, Model driven architecture: Principles and practice, Softw. Syst. Model. 3 (4) (2004) 314–327, http://dx.doi.org/10.1007/s10270-004-0061-2.

[9] S. Barnett, I. Avazpour, R. Vasa, J. Grundy, A multi-view framework for generating mobile apps, in: IEEE Symposium on Visual Languages and Human-Centric Computing, 2015, pp. 305–306, http://dx.doi.org/10.1109/VLHCC.2015.7357239.

[10] S. Barnett, I. Avazpour, R. Vasa, J. Grundy, Supporting multi-view development for mobile applications, J. Comput. Lang. 51 (2019) 88–96, http://dx.doi.org/10.1016/j.cola.2019.02.001.

[11] B. Kitchenham, S. Charters, Guidelines for Performing Systematic Literature Reviews in Software Engineering, Technical Report EBSE 2007-001, Keele University and Durham University Joint Report, 2007.

[12] C. Rahmani, M. Zand, H. Siy, S. Srinivasan, A survey on model driven software development, in: 18th International Conference on Software Engineering and Data Engineering 2009, SEDE 2009, 2009, pp. 105–110.

[13] S.W. Liddle, Model-driven software development, in: Handbook of Conceptual Modeling, Springer, 2011, pp. 17–54, http://dx.doi.org/10.1007/978-3-642-15865-0_2.

[14] T. Stahl, M. Voelter, K. Czarnecki, Model-Driven Software Development: Technology, Engineering, Management, John Wiley &; Sons, Inc., Hoboken, NJ, USA, 2006.

[15] M. Brambilla, J. Cabot, M. Wimmer, Model-Driven Software Engineering in Practice: Second Edition, second ed., Morgan &; Claypool Publishers, 2017, http://dx.doi.org/10.2200/S00751ED2V01Y201701SWE004.

[16] M. Staron, Adopting model driven software development in industry–a case study at two companies, in: International Conference on Model Driven Engineering Languages and Systems, Springer, 2006, pp. 57–72, http://dx.doi.org/10.1007/11880240_5.

[17] H. Heitkötter, T.A. Majchrzak, H. Kuchen, Cross-platform model-driven development of mobile applications with md^2, in: Proceedings of the 28th Annual ACM Symposium on Applied Computing, 2013, pp. 526–533, http://dx.doi.org/10.1145/2480362.2480464.

[18] M. Willocx, J. Vossaert, V. Naessens, Comparing performance parameters of mobile app development strategies, in: Proceedings of the International Conference on Mobile Software Engineering and Systems, 2016, pp. 38–47, http://dx.doi.org/10.1145/2897073.2897092.

[19] A.I. Wasserman, Software engineering issues for mobile application development, in: Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research, 2010, pp. 397–400, http://dx.doi.org/10.1145/1882362.1882443.

[20] M.E. Joorabchi, A. Mesbah, P. Kruchten, Real challenges in mobile app development, in: 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, 2013, pp. 15–24, http://dx.doi.org/10.1109/ESEM.2013.9.

[21] M. Fowler, Domain Specific Languages, first ed., Addison-Wesley Professional, 2010.

[22] A. Hudli, S. Hudli, R. Hudli, An evaluation framework for selection of mobile app development platform, in: Proceedings of the 3rd International Workshop on Mobile Development Lifecycle, MobileDeLi 2015, ACM, 2015, pp. 13–16, http://dx.doi.org/10.1145/2846661.2846678.

[23] A. Sahay, A. Indamutsa, D. Di Ruscio, A. Pierantonio, Supporting the understanding and comparison of low-code development platforms, in: 46th Euromicro Conference on Software Engineering and Advanced Applications, SEAA, IEEE, 2020, pp. 171–178, http://dx.doi.org/10.1109/SEAA51224.2020.00036.

[24] E. Umuhoza, M. Brambilla, Model driven development approaches for mobile applications: A survey, in: International Conference on Mobile Web and Information Systems, 2016, pp. 93–107, http://dx.doi.org/10.1007/978-3-319-44215-0_8.

[25] H. Tufail, F. Azam, M.W. Anwar, I. Qasim, Model-driven development of mobile applications: A systematic literature review, in: 2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference, 2018, pp. 1165–1171, http://dx.doi.org/10.1109/IEMCON.2018.8614821.

[26] P. Kong, L. Li, J. Gao, K. Liu, T.F. Bissyandé, J. Klein, Automated testing of android apps: A systematic literature review, IEEE Trans. Reliab. 68 (1) (2019) 45–66, http://dx.doi.org/10.1109/TR.2018.2865733.

[27] L. Li, T.F. Bissyandé, M. Papadakis, S. Rasthofer, A. Bartel, D. Octeau, J. Klein, L. Traon, Static analysis of android apps: A systematic literature review, Inf. Softw. Technol. 88 (2017) 67–95, http://dx.doi.org/10.1016/j.infsof.2017.04.001.

[28] D. Hidellaarachchi, J. Grundy, R. Hoda, K. Madampe, The effects of human aspects on the requirements engineering process: A systematic literature review, IEEE Trans. Softw. Eng. (2021) 1, http://dx.doi.org/10.1109/TSE.2021.3051898.

[29] S.K. Lo, Q. Lu, C. Wang, H.-Y. Paik, L. Zhu, A systematic literature review on federated machine learning: From a software engineering perspective, ACM Comput. Surv. 54 (5) (2021) http://dx.doi.org/10.1145/3450288.

[30] M. Petticrew, H. Roberts, Systematic Reviews in the Social Sciences: A Practical Guide, John Wiley & Sons, 2008.

[31] K. Lano, L. Alwakeel, S.K. Rahimi, H. Haughton, Synthesis of mobile applications using agileuml, in: 14th Innovations in Software Engineering Conference (Formerly Known As India Software Engineering Conference), ISEC 2021, 2021, pp. 1–10, http://dx.doi.org/10.1145/3452383.3452409.

[32] M. Derakhshandi, S.K. Rahimi, J. Troya, K. Lano, A model-driven framework for developing android-based classic multiplayer 2D board games, Autom. Softw. Eng. J. (2021) (Accepted in March 2021), Available at https://mdse.ui.ac.ir/a-model-driven-framework-for-developing-android-based-classic-multiplayer-2d-board-games/.

[33] D. Wolber, App inventor and real-world motivation, in: Proceedings of the 42nd ACM Technical Symposium on Computer Science Education, 2011, pp. 601–606, http://dx.doi.org/10.1145/1953163.1953329.