Highlights

Accessibility of Low-Code Approaches: a Systematic Literature Review

Hourieh Khalajzadeh,John Grundy

- We report an analysis of the literature on the accessibility of low-code approaches
- We investigate the accessibility considerations in the existing literature
- We categorise the model-driven methods adopted in the existing literature
- We inform other researchers of the future directions in making low-code approaches accessible

Accessibility of Low-Code Approaches: a Systematic Literature Review

Hourieh Khalajzadeh[a], John Grundy[b]

[a]*School of Information Technology, Deakin University, 221 Burwood Highway, Burwood, 3125, Victoria, Australia*
[b]*Faculty of Information Technology, Monash University, Wellington Road, Clayton, 3800, Victoria, Australia*

ARTICLE INFO

Keywords:
Systematic literature review
Low-code
Visual languages
Block-based programming
Accessibility

ABSTRACT

Context: Model-driven approaches are increasingly used in different domains, such as education, finance and app development, in order to involve non-developers in the software development process. Such tools are hugely dependent on visual elements and thus might not be accessible for users with specific challenges, e.g., visual impairments.
Objectives: To locate and analyse existing literature on the accessibility of low-code approaches, their strengths and weaknesses and key directions for future research.
Methods: We carried out a systematic literature review and searched through five leading databases for primary studies. We used both quantitative and qualitative methods for data synthesis.
Results: After reviewing and filtering 918 located studies, and conducting both backward and forward snowballing, we identified 38 primary studies that were included in our analysis. We found most papers focusing on accessibility of visual languages and block-based programming.
Conclusion: Limited work has been done on improving low code programming environment accessibility. The findings of this systematic literature review will assist researchers and developers in understanding the accessibility issues in low-code approaches and what has been done so far to develop accessible approaches.

1. Introduction

Software development requires collaboration among stakeholders with diverse skill sets, technical and non-technical expertise, and various backgrounds, fostering communication and interaction within multidisciplinary teams (Khalajzadeh et al., 2020). According to the research conducted by Costello and Rimhol (2021), an average of 41% of non-Information Technology (IT) professionals engage in customising or creating data and technology solutions. The rise of these “citizen software developers", where individuals without specialised IT knowledge create software, highlights the need to make software development accessible to users without technical backgrounds (Adrian et al., 2020).

Low-code development platforms are software development environments designed to construct software applications using graphical user interfaces and models that represent the underlying code. These platforms facilitate rapid software application development in a format that is more approachable for non-experts (Grillo et al., 2012). The adoption of low-code approaches can significantly boost software development productivity and empower project stakeholders lacking IT expertise to contribute their insights based on their domain knowledge (Bock and Frank, 2021; Zhuang et al., 2022). Visual-based programming languages, like Scratch, which function as low-code platforms, have shown great promise in teaching programming skills, particularly to children (Zubair et al., 2023).

However, most software systems, including Low-code approaches, often prioritise performance and cost-effectiveness while neglecting many diverse human factors among end-users, such as their accessibility for those with diverse ages, preferences, characteristics, experiences, and impairments (Grundy et al., 2020). The software engineering research community has concentrated on the accessibility of software engineering products but has paid less attention to the accessibility of software engineering processes and development tools themselves (El-Attar, 2023). Since low-code heavily relies on graphical user interfaces, it can often exclude many users with visual impairments, including colour blindness and low vision.

According to the World Health Organization (2023b) report, approximately 1.3 billion individuals, which accounts for 16% of the world’s population, are facing substantial disabilities. This figure is on the rise due to the increasing prevalence of non-communicable diseases and the extended lifespan of people. World Health Organization (2023a) also reports a minimum of 2.2 billion individuals worldwide who experience near or distance vision impairments. Visually and cognitively impaired individuals deserve accessible means to actively participate in software development tasks and reap the benefits of low-code abstraction. Consequently, low-code approaches should be inclusive and provide accessibility not only to those without a technical background but also to other citizen developers, including individuals with limited socio-economic resources and those with visual impairments. As an example, students with visual impairments encounter a lot of barriers to learning programming in existing programming environments and curricula (Mountapmbeme and Ludi, 2021). Educators working with visually impaired students often find them either overly intricate or insufficient for accommodating students with visual impairments (Mountapmbeme and Ludi, 2021). Moreover, Luque et al. (2014) claim that graphical notations are considered inaccessible since they impose significant usage barriers

hkhalajzadeh@deakin.edu.au (H. Khalajzadeh); john.grundy@monash.edu (J. Grundy)
ORCID(s): 0000-0001-9958-0102 (H. Khalajzadeh); 0000-0003-4928-7076 (J. Grundy)

on blind people. UML, being the predominant graphical notation in software system development, is extensively embraced within the industry, with numerous educators incorporating UML diagrams into computer education programs to instruct students on object-oriented concepts.

Several literature reviews have been published on investigating the accessibility of programming environments for visually impaired users (Mountapmbeme et al., 2022), accessibility of UML (Seifermann and Groenda, 2016), and accessibility of diagrams in general (Brown et al., 2004; Torres and Barwaldt, 2019). In comparison to these review studies, this novel study identifies and analyses studies discussing the ***accessibility of low-code approaches using model-driven engineering and visual languages*** for a diverse user base through a systematic literature review. The objective of this SLR is to identify the existing approaches taken to make low-code accessible for diverse users. The key contributions of this work are as follows:

- We report on work published to date on the accessibility of low-code approaches;
- We investigate key low-code tool accessibility considerations in the existing literature;
- We categorise key model-driven methods adopted in the existing low-code literature; and
- We suggest to other researchers some key future research directions for making low-code approaches more accessible.

The rest of this paper is organised as follows. Section 2 presents a background on the topic. Section 3 discusses the research methodology, our research questions and the SLR steps. Section 4 presents the results, according to the research questions. Section 6 discusses the threats to the validity, while Section 7 concludes the paper.

2. Background and Motivation

2.1. Low-code

As model-driven development (MDD) becomes more prevalent in the engineering of software systems, there is a concurrent rise in the quantity of low-code applications (Kirchhof et al., 2023). Low-code approaches enable simplified one-step deployment by employing declarative, high-level programming abstractions like model-driven and metadata-based programming languages. They expedite the development, deployment, execution, and management of applications, encompassing user interfaces, business processes, and data services (Vincent et al., 2019). Low-code methods have garnered substantial attention from both academic and industrial circles in recent years, with 40% of businesses adopting these techniques (Hale, 2022). Nonetheless, research regarding the accessibility of low-code approaches for a diverse range of end-users, including those with visual and cognitive impairments, remains notably limited. Domain-specific visual languages (DSVL) are visual modelling languages used in low-code approaches, where the models and notations are customised for a particular domain (Bottoni et al., 2004). On the other hand, general-purpose modelling languages, such as UML, are not specific to a domain. Several design principles are presented to guide through notational design and evaluation, e.g., Cognitive Dimensions (CD) of notations (Green and Petre, 1996) and Physics of Notations (PoN) (Moody, 2009).

2.2. Accessibility

While everyone has the right to access and take advantage of the opportunities presented by the Information Society, certain demographics, such as the elderly and individuals with disabilities, may encounter challenges in utilising these new technologies and services at times (Kavcic, 2005). "Accessibility" is a broad term used to convey the ease with which individuals can access, utilise, and comprehend various elements (Kavcic, 2005). The objective of software accessibility is to ensure the availability and usability of software applications by as many users as possible. This entails enabling all users to perceive and understand on-screen content and operate controls. Consequently, software application design must accommodate variations in users' abilities to see, hear, input, read text, and process information, recognising that these abilities may differ among users, change over time, and be influenced by the context of use (Kavcic, 2005).

Different guidelines have been proposed to ensure software products are accessible to diverse users. Efforts such as the World Wide Web Consortium (W3C) Initiative (Chisholm et al., 1999), known for publishing the inaugural Web Content Accessibility Guidelines in 1995, have established it as the longest-standing standards body for web accessibility guidelines. W3C is, however, designed to ensure web accessibility.

2.3. Prior Surveys and Reviews

Sarioğlu et al. (2023) conducted a systematic literature review that explores the current research on accessibility in conceptual modelling. Mountapmbeme et al. (2022) conducted a literature review on 70 studies investigating the accessibility of programming environments for visually impaired users. This study does not investigate low-code programming environments. The study conducted by Seifermann and Groenda (2016) evaluated the existing status of 31 textual notations for UML, which can serve as alternatives to, or work in conjunction with, graphical notations. This assessment was performed through an SLR. They study the textual UML notations given UML has become the common language in software description languages. UML's textual notations are available to individuals with visual impairments, offering a more developer-centric and concise way of presenting information. This study only focuses on UML accessibility. Brown et al. (2004) conducted a literature review to explore the challenges associated with making graph-based diagrams accessible to individuals with visual impairments through nonvisual presentation methods. Torres and Barwaldt (2019) conducted an SLR on 26 studies

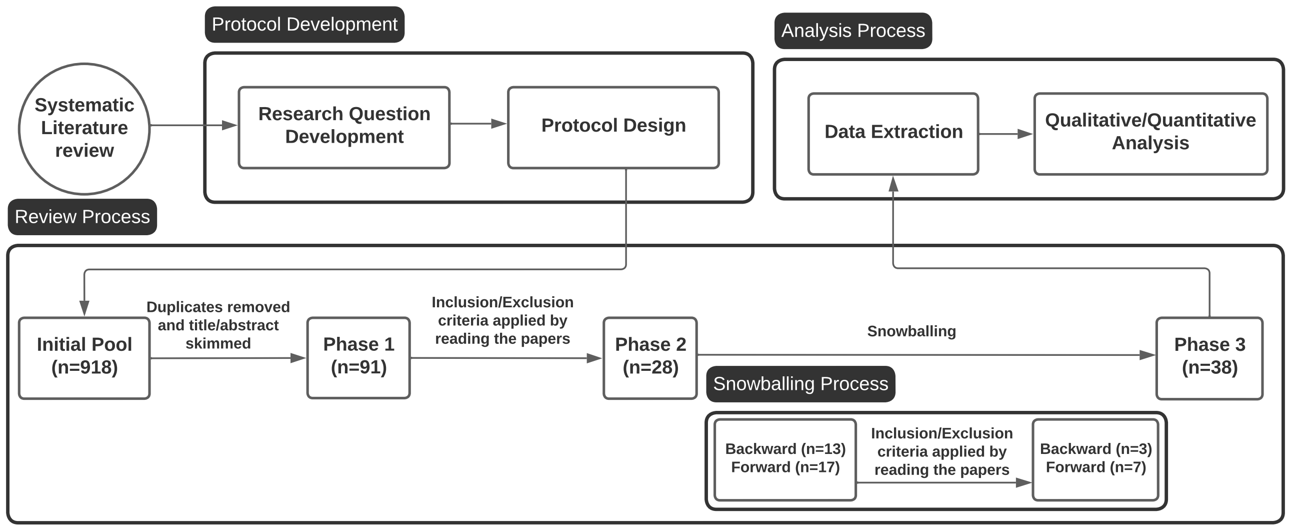
Protocol Development
Analysis Process
Systematic Literature review
Research Question Development
Protocol Design
Data Extraction
Qualitative/Quantitative Analysis
Review Process
Initial Pool (n=918)
Duplicates removed and title/abstract skimmed
Phase 1 (n=91)
Inclusion/Exclusion criteria applied by reading the papers
Phase 2 (n=28)
Snowballing
Phase 3 (n=38)
Snowballing Process
Backward (n=13) Forward (n=17)
Inclusion/Exclusion criteria applied by reading the papers
Backward (n=3) Forward (n=7)

Figure 1: Stages of SLR process

proposing solutions to make diagrams accessible for blind people. The authors found that the keyboard was the main input interface for the interactions. These studies focus on diagrams in general and not specifically diagrams used in model-driven approaches.

3. Research Methodology

We adhered to the SLR guidelines and procedures outlined in Kitchenham et al. (2022) and drew upon the methodology introduced by Watson et al. (2022) to ensure the robustness of our analysis and to establish a process that can be replicated. The primary author formulated the review protocol, which underwent a thorough review by the second author to mitigate potential biases. Our protocol outlines the core objectives of the review, the essential contextual information, research questions (RQs), inclusion and exclusion criteria, the search strategy, data extraction, and the subsequent data analysis. The entire process is illustrated in Figure 1, which delineates the three principal phases of our review: planning the review, selecting studies and extracting information. These phases will be discussed in more detail in the following subsections.

3.1. Research Questions

We aimed to formulate research questions (RQs) that naturally guide the establishment of a taxonomy for the examined research and tackle the obstacles encountered in the design of low-code approaches. The RQs of this study are as follows:

RQ$_1$: *What research to date has focused on developing accessible low-code IDEs?* This RQ investigates primary studies found on Low-code IDEs that are accessible, in terms of the authors' affiliations, publication venues and years, users and applications.

- **RQ$_{1a}$**: *What is the publication venue distribution of the primary studies?* RQ$_{1a}$ analyses the authors' affiliations, the venues papers related to the accessibility of low-code IDEs are published in, and the publication years.

- **RQ$_{1b}$**: *What applications are they used for? industry/academic examples?* RQ$_{1b}$ investigates the applications domains.

RQ$_2$: *What accessibility considerations are taken into account in the low-code IDEs designed to date?* In this RQ, we aim to investigate the accessibility considerations that low-code IDE developers have taken into account in our selected primary studies. Therefore, the following sub-RQs have been defined.

- **RQ$_{2a}$**: *Are any of them accessible for users with special needs, including visually impaired users?* RQ$_{2a}$ analyses whether the selected primary studies have developed a fully accessible solution, a partially accessible one, or have just proposed an accessible solution without actually implementing it.

- **RQ$_{2b}$**: *What guidelines they have used?* RQ$_{2b}$ investigates whether they have used any accessibility guidelines to implement their accessible solution.

- **RQ$_{2c}$**: *How do they ensure accessibility?* RQ$_{2c}$ analyses and classifies the methods they have used to implement their accessible solution.

RQ$_3$: *What MDE approaches have been employed?* RQ$_3$ investigates the MDE approaches being used by accessible low-code approaches. It studies the guidelines they used, whether they use code generation or transformation methods, the solution and study types and how the approaches are being evaluated.

- **RQ$_{3a}$**: *What design guidelines are employed? (Physics of Notations, Cognitive Dimensions, etc)* RQ$_{3a}$ investigates whether any of the approaches use design guidelines proposed for low-code IDE development.
- **RQ$_{3b}$**: *What is generated using MDE? What are the key MDE transformation methods used?* RQ$_{3b}$ analyses whether they generate code or apply model transformations.
- **RQ$_{3c}$**: *What are the study types and what solutions are proposed?* RQ$_{3c}$ analyses and classifies the type of the studies.
- **RQ$_{3d}$**: *Do they describe a tool prototype? What technology stack does the tool use? How does this impact accessibility provision?* RQ$_{3d}$ analyses whether the primary study describes a tool prototype, the technology it was built on, and impact of this technology stack on low-code accessibility provision in the tool.
- **RQ$_{3e}$**: *How are the low-code approaches evaluated? Is the generated code verified?* RQ$_{3e}$ analyses and classifies the evaluation methods, and discussed the evaluation results.

RQ$_4$: *What are key future research directions for more accessible low-code approaches?* This research question analyses the limitations discussed in the selected primary studies and their future directions. This aims to help other researchers further build on top of the current research on the accessibility of low-code IDEs.

- **RQ$_{4a}$**: *What are the strengths discussed in the selected primary studies?* RQ$_{4a}$ discusses the strengths discussed in the selected primary studies.
- **RQ$_{4b}$**: *What are the limitations discussed in the selected primary studies?* RQ$_{4b}$ discusses the limitations discussed in the selected primary studies.
- **RQ$_{4c}$**: *What are the future works?* RQ$_{4c}$ discusses the future directions proposed by the authors of the selected primary studies.

3.2. Search strategy

Our search strategy was devised to locate and gather all relevant literature that aligns with the specific inclusion and exclusion criteria outlined in Section 3.3.

3.2.1. Data sources

We evaluated and assessed the search engines employed in prior software engineering literature reviews, as documented in Maplesden et al. (2015) and Shahin et al. (2014). The final selection of electronic databases that we opted to search included: ACM Digital Library, IEEExplore, SpringerLink, ScienceDirect, and Scopus. We excluded Wiley, Compendex, and Inspec due to their substantial overlap with other search engines. Notably, Compendex and Inspec exhibited a high degree of overlap with Scopus, as indicated in Maplesden et al. (2015). Furthermore, Wiley is already indexed by Scopus, as corroborated by Wiley (2022).

3.2.2. Search terms

As shown in Table 1, we used PICOC criteria (Keele, 2007) to determine the search terms. We included all the terms related to model-driven development, e.g. domain-specific visual languages, block-based programming, low-code/no-code, etc in the search string since they are sometimes used interchangeably in the literature.

3.3. Inclusion and exclusion criteria

The criteria for inclusion and exclusion, as outlined in Table 2, were employed to assess all the studies retrieved from the databases and target venues. The criteria that start with an "I" are the inclusion criteria, and those that start with an "E" are the exclusion criteria.

3.4. Study Selection

Figure 1 illustrates the SLR process and the quantity of studies gathered during each phase. The selection of primary research adhered to predefined inclusion and exclusion criteria, detailed in Section 3.3. Throughout the progression from the initial stage to the ultimate screening, essential records of the chosen primary studies were meticulously maintained within Excel spreadsheets. An overview of the 38 studies that met the inclusion criteria, along with demographic analysis, can be found in Table 9 in Appendix A. The selection process is structured into four distinct phases:

Initial Pool: We executed the search query across the five digital libraries, resulting in the retrieval of 918 papers.

Phase 1: We then removed duplicates and assessed the publications found during the initial search based on their title and abstracts. We retained papers for thorough screening when it was challenging to decide based solely on their titles and abstracts. By the conclusion of this stage, 91 papers were chosen.

Phase 2: Publications that passed the initial screening in Phase 1 proceeded to undergo a comprehensive full-text assessment. Out of these, 28 papers were identified as the pertinent subset and were subject to review by the first author. In addition, four of the selected primary studies were independently evaluated by an external reviewer. Any disagreements that arose were promptly resolved through discussions. Subsequently, we employed both backward and forward snowballing techniques as described in (Wohlin, 2014) and identified a total of 30 potentially relevant papers based on their titles and abstracts.

Phase 3: In this stage, we conducted a review of the papers obtained through snowballing. The entire snowballing process is depicted in Figure 1. We utilised Google Scholar for both *forward snowballing*, which involved identifying additional papers that cited any of the included studies, and *backward snowballing*, which entailed examining the references of the selected papers from Phase 2. We deemed this manual search phase as necessary to minimise the risk of losing potentially relevant literature that might be overlooked in automated searches. Following the application of inclusion and exclusion criteria, removal of duplicates, and full-text screening, we retained 10 papers from the

Table 1
Key search terms aligned with the PICOC criteria

Concepts	Major search terms
Population	("Domain Specific Visual Language" OR "Domain-Specific Visual Language" OR "DSVL" OR "Visual domain-specific language" OR "Visual domain specific language" OR "VDSL" OR "Unified Model*ing Language" OR "UML" OR "istar" OR "Model-Driven Engineering" OR "Model-Driven Development" OR "Model Driven Engineering" OR "Model Driven Development" OR "Platform Independent Model" OR "Computation Independent Model" OR "Platform Specific Model" OR "Model-Driven Architecture" OR "Meta Model" OR "low-code" OR "lowcode" OR "no-code" OR "nocode" OR "MDE" OR "MBE" OR "MDA" OR "MDD" OR "block programming" OR "block-based programming")
Intervention	("Abstract": "accessibility" OR "colo*r blind*" OR "low vision" OR "vision impair*" OR "visual* impair*")

Criteria ID	**Criterion**
I01	Full-text papers published as a journal, or conference proceeding, including workshops co-located with the conferences, with a focus on the accessibility of low-code approaches including domain-specific modelling languages, and block-based programming approaches.
I02	The complete papers are authored in English and cite academic literature references.
I03	The study must be available in full text and published in one of the digital libraries we considered in this review.
E01	Gray literature, posters, books, work-in-progress proposals, keynotes, editorial, secondary or review articles.
E02	Short papers with fewer than four pages and studies that are irrelevant or of low quality, lacking a substantial amount of extractable information.
E03	An extended or up-to-date journal edition authored by the same individuals is accessible for the same research.
E04	Papers that discuss the accessibility of low-code approaches but do not propose a solution.
E05	Papers that use low-code approaches to generate accessible solutions.

Table 2
Inclusion and Exclusion Criteria

snowballing process. This expanded the total number of primary studies in our analysis to 38.

3.5. Data extraction strategy

To address RQ$_1$–RQ$_4$ and streamline our data extraction procedure, we employed a Google data extraction form to gather essential data from the studies included in our analysis. The extraction form is divided into four sections. The first section captures the paper's title, authors, affiliation, country, year, venue, type and ranking of the venue, number of pages, and study aim. The second section addresses accessibility, noting if the approach is accessible, the special needs considered, the accessibility guidelines and methods used, and the provided solutions. The third section examines the application domain, specifying if the study is academic or industrial, the MDE approach, design guidelines, technologies, MDE transformation method, and evaluations. The final section outlines the limitations, strengths, and future work. The data extraction form can be accessed through Khalajzadeh and Grundy (2024).

3.6. Data Synthesis, and Taxonomy Derivation

We compiled all essential information within a data extraction sheet, encompassing two key components: **a)** Demographic data, such as title, authors, venue, affiliation, and publication type. **b)** Responses to each RQ, comprising both qualitative and quantitative data.

Quantitative analysis: We conducted both *univariate and multivariate frequency distribution analyses*. The *univariate* frequency distribution served to provide a summarised count of occurrences within specific variables. Meanwhile, *multivariate* frequency analysis was applied to aggregate the distribution of two or more variables, thereby elucidating their interrelationships.

Qualitative analysis: To build our taxonomy, we used open coding technique (Glaser and Strauss, 2017).

4. Results

4.1. RQ$_1$: What research has been focusing on developing accessible low-code IDEs?

4.1.1. RQ$_{1a}$: What is the distribution of the primary studies found?

We did a comprehensive analysis to find the active researchers working on the accessibility of low-code approaches and the venues where most of these works are published. We found six of the selected primary studies being conducted at the University of São Paulo in Brazil, three at Toyama Prefectural University in Japan, two at NOVA University in Lisbon, and two at the University of North Texas in the USA. There was only one paper from each of the other universities. A list of the countries and the first authors are affiliated with is shown in Fig 2. As shown in this image, Brazil and USA are the frontrunners in publishing papers in this field.

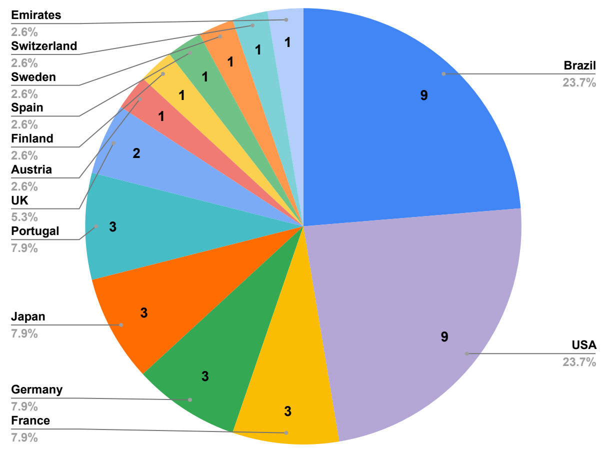

Figure 2: Countries where the first author is affiliated with

We did not limit the year of the studies, however, the primary studies we found through the systematic method were published since 2004, with most of them being published in 2015 and 2021, as shown in Fig 3.

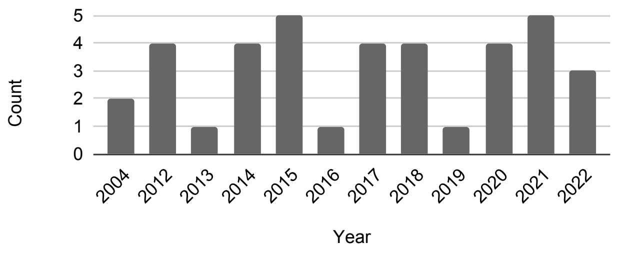

Figure 3: Publication years

In terms of the venues, most of the selected primary studies are published in conferences (n=26), followed by journals (n=7), workshops co-located with top conferences such as MODELS (n=4), and finally only one published as a book chapter. The conferences are mostly not ranked (n=23) according to the Computing Research and Education Association of Australasia (CORE) ranking[1]. using Acientific Journal Rankings (SJR)[2], our selected primary studies are published in two A*-ranked, three A, two B, and three C-ranked conferences, and one Q1 and four Q2 journals. A complete list of the venues, their counts, types and rankings are shown in Table 3.

4.1.2. *RQ$_{1c}$: What applications are they used for?*

Almost all of the selected primary studies were conducted in academia, with only one having an author affiliated with The National Council for the Blind [S29]. Approaches are developed and discussed for various applications, but mainly for education purposes (n=13), and general applications (n=11). Two of the selected primary studies focus on industrial projects [S19], [S33]. Two focus on education for robot programming, [S2] and [S14], and one on robot programming [S5]. The rest of the selected primary studies focus on Education through E-Learning [S6], Education and Industry [S17], General - Education and Automation [S22], Computational Physics Research [S27], Creating and Modifying UML Class Diagrams on Mobile Devices [S1], Interactive e-learning Activities [S30], Interactive Whiteboards [S24], Security in requirements engineering [S25], and Web Applications Development [S28].

[1]http://portal.core.edu.au/conf-ranks/
[2]https://www.scimagojr.com/

RQ1 Summary. Most of the selected primary studies were conducted in Brazil (23.7%) and the USA (23.7%). All were published since 2004, and mostly from 2012 afterwards. The studies are mostly published in conferences (68.4%) and small number in journals (18.42%). Most studies are conducted in academia (97.3%). The methods are developed for a variety of different applications, mainly education (34.2%), and general applications (28.9%).

4.2. RQ$_2$: What accessibility considerations are taken into account in the low-code IDEs designed to date?

4.2.1. *RQ$_{2a}$: Are any of them accessible for users with special needs, including visually impaired users?*

Among the 38 studies, 32 of them have developed an accessible solution for various special needs, one presents a partially accessible solution, while the remaining five studies have discussed the accessibility issue, but have not developed a solution. The solutions proposed or discussed in different papers are proposed to make low-code approaches accessible. Some of them consider general accessibility, while others consider accessibility for a special group of users. The special needs considered in these 38 studies are listed in Table 4.

4.2.2. *RQ$_{2b}$: What guidelines have they used?*

Surprisingly, most of the selected primary studies do not consider any accessibility guidelines. However, those that do consider accessibility guidelines are mainly based on Web Content Accessibility Guidelines (WCAG) international standard (n=3) [S31], [S32], and [S36]. WCAG is a collaborative effort of the World Wide Web Consortium (W3C) involving individuals and organisations worldwide. Its primary objective is to establish a global standard for web content accessibility that caters to the requirements of individuals, organisations, and governments on an international scale. WCAG offers documentation that outlines methods for enhancing the accessibility of web content, encompassing textual information (such as text, images, and audio) as well as code and markup (comprising structure and presentation) to better serve individuals with disabilities, as stated in Caldwell et al. (2008). [S37] follow Web Accessibility Initiative and Accessible Rich Internet Application (WAI-ARIA) (Craig et al., 2009) attributes. Finally, [S35] mentions

Table 3
List of the venues

Venue	Count	Venue type	Ranking
International Conference on Computers Helping People with Special Needs (ICCHP)	3	Conference	C
Conference on Human Factors in Computing Systems (CHI)	2	Conference	A*
IEEE Frontiers in Education Conference (FIE)	2	Conference	Not Ranked
International Workshop on Human Factors in Modeling / Modeling of Human Factors (HuFaMo) at MODELS	2	Workshop	Not Ranked
Software and Systems Modeling	2	Journal	Q2
Universal Access in Human-Computer Interaction (UAHCI)	2	Conference	Not Ranked
ACM SIGACCESS Conference on Computers and Accessibility (ASSETS)	1	Conference	A
Annual Computers, Software, and Applications Conference (COMPSAC)	1	Conference	B
Brazilian Conference of Software	1	Conference	Not ranked
Brazilian Journal of Computers in Education	1	Journal	Not ranked
Computer Physics Communications	1	Journal	Q1
Conference on Technologies and Applications of Artificial Intelligence	1	Conference	Not Ranked
Conferencia Iberoamericana en Software Engineering (CIbSE)	1	Conference	Not Ranked
Graphical Modeling Language Development (GMLD) workshop at ECMFA	1	Workshop	Not Ranked
IEEE Blocks and Beyond Workshop	1	Workshop	Not Ranked
IEEE Symposium on Robotic Intelligence in Informationally Structured Space (RiiSS)	1	Conference	Not Ranked
Inclusive Designing	1	Conference	Not Ranked
Innovation and Technology in Computer Science Education (ITiCSE)	1	Conference	A
Interaction Design and Children Conference (IDC)	1	Conference	Not Ranked
International Conference on Auditory Display	1	Conference	Not Ranked
International Conference on Automated Software Engineering Workshops (ASEW)	1	Conference	Not Ranked
International Conference on Computers for Handicapped Persons	1	Conference	Not Ranked
International Conference on Human-Computer Interaction	1	Conference	B
International Conference on Model Driven Engineering Languages and Systems (MODELS)	1	Conference	A
International Model-Driven Requirements Engineering (MoDRE)	1	Conference	Not Ranked
Journal of Visual Language and Sentient Systems (VLSS)	1	Journal	Q2
Journal of Visual Languages & Computing	1	Journal	Q2
Mensch und Computer (MuC)	1	Conference	Not Ranked
Procedia Computer Science	1	Journal	Not Ranked
Springer Nature Switzerland	1	Book Chapter	Not Ranked
World Automation Congress (WAC)	1	Conference	Not Ranked

Table 4
The special needs considered in different studies

The special need	# of studies
Visual Impairment	24
Visual Impairment and Age (Younger Learners)	5
Visual Impairment and Technical Background (Beginners)	1
Technical Background and Location	1
Diversity	1
Technical Background	1
Physically disability	1
Socio-economic Status	1
General and Disabilities	1
General	2

that ARIA accessibility guidelines can be added to their approach but have not implemented it in their paper.

4.2.3. *RQ$_{2c}$: How do they ensure accessibility?*

The selected primary studies follow three approaches for developing accessible solutions: 1) Developing new accessible tools (n=24), 2) Using existing accessibility options to make the existing solutions accessible (n=8), and 3) Developing algorithms to provide accessible solutions (n=1). The rest of the papers have only studied the existing solutions (n=5).

Developing Accessible Solutions: Most of the selected primary studies propose a variety of accessible tools. [S2] developed P-CUBE, which is a new programming education tool. With P-CUBE, users construct programs by arranging surface gap blocks on a program mat per the algorithm's structure. This tool is renowned for its robustness owing to its straightforward design and is anticipated to serve as an educational toy for students. [S3] created P-CUBE2, a tangible block-based tool for programming education. P-CUBE2 was designed to incorporate the concept of subroutines into the original P-CUBE. Notably, this version introduces an audio output feature, enabling visually impaired individuals to discern the outcomes of program execution through voice feedback. [S5] made enhancements to P-CUBE with the specific aim of making it more accessible to individuals with visual impairments. These improvements include programming blocks that offer tactile cues regarding their types to users, and a mobile robot that emits distinct sounds when turning right or left, effectively conveying data on the robot's operational status. In the study conducted by [S37], the authors undertook a redesign of Blockly's user interface, introducing supplementary features aimed at enhancing accessibility for visually impaired individuals and those who cannot utilise a mouse. Their approach encompassed the integration of a keyboard interface, compatibility with screen readers, options for customising the appearance, and other related features, all geared toward expanding access to systems built on Blockly.

The goal of [S8] was to establish an environment that could be utilised independently and found engaging by individuals who are sighted, have low vision, or are blind. They strived to implement modifications that could be readily integrated into existing block-based environments. Furthermore, they aimed to preserve the features of block-based environments that render them suitable for young children. [S9] introduced Accessible Web Modeler (AWMo), a prototype web tool designed to enable users to model software artifacts using UML class diagrams. AWMo offers two distinct views: a graphical view for sighted software developers to create diagrams in a conventional manner by visually dragging elements and connecting them, and a textual view that allows software developers, regardless of visual impairment, to generate identical diagrams using a text-based language they developed. Similarly, [S16] proposes to provide users with both textual and graphical views. [S10] developed Tangible Blocks Go Online (TaBGO), a tool comprising enhanced tangible Scratch's blocks and associated optical recognition software. [S11] presents a prototype design that uses lego and plays music for visually impaired users.

[S18] propose StoryBlocks, a tangible block-based game that creates audio stories to help enable blind programmers to learn basic programming concepts. [S19] presents a method that maintains synchronisation between models and text by skillfully merging advanced natural language processing technology with OCL model querying. [S21] developed a novel module built on an entirely new platform. [S22] constructed a modelling workbench platform that, similar to modelling workbenches focused on diagrammatic languages, enables a software language engineer to design a domain-specific language. This platform also facilitates the creation of a voice/audio editor, allowing domain end-users to interact with and navigate diagrams using speech recognition and voice synthesis tools. [S27] developed a tool designed to streamline code development in global collaborations, catering to developers with varying levels of expertise and coding skills. This tool enhances backward traceability, supports the iterative and incremental development of intricate algorithms, and improves code readability and transparency. [S28] made improvements to the WebML language. The system designed by [S29] employs various interfaces and representation techniques to address the hurdles of rendering UML diagrams accessible to individuals with visual impairments.

[S30] developed a web-based prototype called Inclusive UML with different views for blind and sighted users. [S31] created a tool that offers alternative textual representations of models within a web-based environment, promoting effective collaboration. In the work by [S32], they created a tool that incorporates a screen-reader compatible interface tailored for blind users, alongside a graphical interface designed for sighted users. [S33] developed a tool that has a graphical view, an Auditory Design, and an Audio-Haptic Design. [S34] developed a tool that has a voice recognition component. [S35] developed a tool with diagrams that are readable by screen readers. In their study, [S15] put forward a model-based approach for the bidirectional generation of textual and graphical representations of UML diagrams. The UML4ALL syntax aligns with the workflow of screen reader users, facilitating content consumption through text-to-speech or Braille reading, as it reads content line by line. They also provide an accessible syntax structure for blind people to help understanding the semantics of UML elements. [S36] introduces a prototype for an accessible block-based programming library named "Accessible Blockly." This library empowers users to both generate and traverse block-based code with the assistance of a screen reader and keyboard. Accessible Blockly facilitates access to the abundant visual data found in block-based programming environments and offers an alternative interaction method to the conventional mouse-based approach.

Use of existing accessibility tools: Several selected primary studies propose accessibility tools to make modelling diagrams more accessible for users with special needs. [S1] harnesses the accessibility tools already available on contemporary mobile devices and enhance these devices with an economical physical guide. This integration seamlessly complements the existing features of iOS VoiceOver. [S7] employs abstract sounds and appropriate metaphors for an intuitive mapping. This approach enables the transformation of actions performed by others into perceptible cues for individuals with visual impairments through an auditory interface and sonification of crucial elements related to workspace awareness. As a result, blind users gain the ability to access collaborative resources and stay informed about workspace updates. [S13] creates 3D-printable haptic UML models from an existing UML modelling tool. [S12] employed an accessible UML editor designed for collaborative work on UML models in an inclusive manner. Accessibility was ensured through the use of a textual syntax and the incorporation of flexible assistive reading and editing functionalities. They also implemented model synchronisation mechanisms to enable collaborative work. The introduced accessibility features encompassed change histories and audio notifications, providing support for individuals with visual impairments. [S20] uses Google's latest Keyboard Navigation to enhance the environment. [S23] utilises tactile representations and haptic interaction as a means to potentially empower individuals with visual impairments to work with UML diagrams using a widely adopted industry-standard editor. They have developed both tactile representations and a verbalised version. [S24] introduces a voice interaction mode within software design environments. [S38] enables UML modelling with voice recognition.

Using New Algorithms: One of the selected primary studies applies a new algorithm to make diagrams accessible. [S4] employes transfer learning and data augmentation techniques to classify UML diagrams. The authors used a dataset custom-made for this research, encompassing six distinct types of UML diagrams. The project served as an exploratory endeavor to implement Convolutional Neural

Networks (CNN) in the task of UML diagram classification. In addition to CNN, they harnessed transfer learning and data augmentation to leverage the benefits of pre-trained models.

Studying the Existing Solutions: [S14] engaged educators and visually impaired children in their design activities, adhering to established best practices in the literature. In their first study, the authors conducted a retrospective analysis of established solutions, extracting insights into the strengths and limitations of such approaches. In their second study, they took a prospective approach to evaluate the performance of a tailored accessible environment and derived insights that could guide the development of future environments. [S17] does not propose an accessible solution but instead focuses on assessing existing solutions. [S25] carried out an empirical study involving 84 IT professionals to evaluate the cognitive effectiveness of the RGD version of the PoN-enabled misuse case notation, as introduced by (Saleh and El-Attar, 2015). [S26] contrasted block-based programming lessons designed for two different participant groups: 29 high school interns with prior programming experience but no formal pedagogical training, and 86 teachers, with no background in programming but formal pedagogical training. [S6] assessed the perceptions of learners, which included both visually impaired and sighted individuals, during e-learning lectures focused on software requirements specification utilising UML/SysML use case models. To facilitate communication and coordination of lecture activities, participants utilised a tool known as Model2gether for collaborative real-time interaction with use case models.

RQ2 Summary. Most of the selected primary studies developed an accessible solution for users with visual impairments (63.1%). The rest of the special needs include age, technical background, diversity, physical disability, and socio-economic status. Most of the studies do not consider any accessibility guidelines (86.8%). The studies rarely consider or discuss WCAG (7.8%), and WAI-ARIA (5.2%). We categorise the methods they used to ensure accessibility into *Developing Accessible Solutions* (63.1%), *Use of existing accessibility tools* (21%), *Using New Algorithms* (2.6%), and just *Studying the Existing Solutions* (13.1%).

4.3. RQ$_3$: What MDE approaches have been employed?

The grouping of selected primary studies by approach is shown in Table 5. Over 40% of the selected primary studies build on top of UML, aiming to make UML accessible for users with special needs (n=16). Nearly 30% of the selected primary studies use block-based programming (n=11). Two primary studies do not consider any specific approaches, and two are based on Scratch. The rest of the selected primary studies are based on variations of UML, such as jsUML2, OctoUML, PlantUML, both UML and PlantUML, and both UML and SysML. One of the studies is based on WebML, and one based on Workspace Sketch.

Table 5
MDE Approaches used in different studies

MDE approach	Studies
UML	[S1,S4,S9,S12,S13,S15,S16,S17,S19, S23,S25,S29,S30,S32,S33,S38]
Block Programming	[S2,S3,S5,S8,S14,S18,S20,S21, S26,S36,S37]
General	[S22,S34]
Scratch	[S10,S11]
jsUML2	[S31]
OctoUML	[S24]
PlantUML	[S27]
UML and PlantUML	[S35]
UML and SysML	[S6]
WebML	[S28]
Workspace Sketch	[S7]

4.3.1. *RQ$_{3a}$: What design guidelines are employed?*

Most of our selected primary studies do not consider any specific design guidelines. The exceptions are [S8] and [S10] which have considered their own design guidelines for developing model-driven approaches. [S30] established 11 requirements for the accessibility of UML for blind users in an online mode. These requirements were derived from user stories provided by sighted participants, blind participants, and general participants. The analysis drew upon their collective experience in teaching object-oriented analysis with UML, which highlighted five common activities that occur during the process. [S25] assess their method which is based on the Physics of Notations (PoN) framework and [S28] use both Cognitive Dimensions (CDs) and PON to develop their solution.

4.3.2. *RQ$_{3b}$: What is generated using MDE? What are the MDE transformation methods used?*

Most of the approaches described in the selected primary studies do not generate any arefacts using the MDE approach, except for [S27], [S31], and [S34]. [S27] created the Flowgen tool, which produces a collection of interconnected high-level UML activity diagrams. Specifically, it generates one diagram for each annotated function or method found in the C++ source code. [S31] converted a textual model into a format that is compatible with the jsUML2 JavaScript library on the application view. [S34] used the Epsilon Transformation Language (ETL) and Epsilon Generation Language (EGL) to save the diagram in XMI.

4.3.3. *RQ$_{3c}$: What are the study types and what solutions are proposed?*

The solutions proposed by our selected primary studies are summarised in Table 6. Almost three quarters of the selected primary studies develop a prototype (n=28), three conduct empirical studies, two propose a proof of concept, and the rest propose a framework, set of guidelines, method,

Table 6
Proposed solutions in different studies

Solution		Studies
Prototype	Providing graphical and textual views	[S6,S7,S9,S32,S16,S30,S31]
	Extending block-based tools	[S2,S3,S5,S10,S18,S21,S36,S37]
	Using assistive technologies	[S8,S11,S20,S22,S23,S24,S34]
	Converting to accessible versions	[S29,S19,S35,S38]
	User-centred design	[S1,S33]
Empirical Studies		[S17,S25,S26]
Proof of concept		[S13,S27]
Framework		[S12]
Guidelines		[S14]
Method		[S4]
New Graphical Symbols		[S28]
UML notation		[S15]

new graphical symbols, and UML notations. In this section, we summarise all these methods.

Prototype: Most of the selected primary studies propose an accessible low-code prototype. We have divided the prototypes into five different categories. Category 1 presents prototypes where they provide both graphical and textual views, to make them accessible for users with visual impairments. Category 2 prototypes build on top of block-based tools and make them accessible. Category 3 prototypes use assistive technologies, such as voice recognition for those who are not able to use graphical interfaces. Category 4 prototypes convert the general user interface to accessible versions. Finally, Category 5 prototypes are built based on user-centred design approaches, i.e. through co-design sessions or workshops with real users.

Category 1: Providing graphical and textual views. [S32] and [S6] introduced Model2gether, an online tool designed to facilitate collaborative modelling and ensure the participation of individuals with visual impairments. The prototype offers two distinct interfaces: one designed to be compatible with screen readers, catering to blind users, and a graphical interface tailored for sighted users. Both interfaces incorporate mechanisms to facilitate conversation, coordination, and awareness during collaborative modelling activities. [S7] designed an auditory interface and then implemented a web-based prototype for diagrams' collaborative modelling in order to evaluate the auditory interface. [S31] introduced several model representations utilising both graphical and textual notations. They also put forth the Accessible Web Modeler (AWMo), a web-based tool designed to enhance collaboration with visually impaired users through multiple model presentations. With the utilisation of a screen reader, a blind or visually impaired software developer can engage with the textual model, while other developers can continue using the conventional graphical model that they are accustomed to when working on the same model. Any changes made in one view are reflected in the other, ensuring synchronisation between the two representations. [S9] and [S16] further used AWMo, where sighted and visually impaired users can work in two different modes, visual and textual, which allows collaborative work in both modes.[S30] developed a web-based tool prototype with an accessible version of the diagrams. This accessibility was achieved by implementing a textual Domain-Specific Language (DSL), built upon the yUML DSL, and introducing keyboard shortcuts for interaction. Consequently, both sighted and visually impaired individuals can access distinct views of the same diagram.

Category 2: Extending block-based tools. [S2] presented a prototype called PCUBE with which the users can create two types of programs: sequential programs and line trace programs. [S3] developed P-CUBE2, a tangible block-type programming education tool. [S5] developed a further extension of P-CUBE. [S10] developed Tangible Blocks Go Online (TaBGO) prototype, to allow Scratch programming in classrooms for visually impaired people. The tool uses tangible blocks and associated optical recognition software. [S18] developed StoryBlocks prototype, a tangible block-based game that helps blind programmers to learn basic concepts of programming by creating audio stories. [S21] designed MUzECS, which is intended to be a more affordable alternative to Lego Mindstorms, all while maintaining support for the same educational curriculum objectives. It relies on a custom block-based programming approach, providing a more economical alternative for low-income schools to deliver Computer Science courses. [S37] extended blockly by adding screen reader support. They used three attributes from WAI-ARIA guidelines to make Blockly accessible. They also included changes such as the option to change the workspace text size and colour, an accessible customised help guide, and a window to display comments. [S36] created a prototype for an inclusive block-based programming library named "Accessible Blockly," which enables users to generate and navigate block-based code using a combination of a screen reader and a keyboard.

Category 3: Using assistive technologies. In [S8] the authors developed Blocks4All, a prototype environment in which they integrated diverse methods to overcome accessibility challenges using a touchscreen tablet computer. [S20] created a self-voicing solution that does not rely on a screen reader, instead utilising synthetic speech and interactive text-to-speech methods. [S22] developed a prototype platform that employs voice, sound, and gestures as interactive elements for manipulating software models. [S23] designed a screen explorer for UML that retains the layout information and textual labels of diagrams. This tool enables both blind and sighted individuals to read UML diagrams within a software system, comprising a standard UML editor and assistive technology while maintaining consistency. [S24] developed an extended version of their previous work, OctoUML, which is voice-recognition-enabled (OctoUML-V). [S34] designed a prototype known as ModelByVoice, which seeks to leverage contemporary voice recognition and speech synthesis technologies for the purpose of editing models in various modeling languages. [S11] created an innovative instructional platform referred to as the Tangible Programming Tool (TPT) environment. This prototype utilises LEGO bricks and incorporates musical elements to cater to visually impaired users.

Category 4: Converting to accessible versions. [S29] developed a fully automated system that allows users to receive UML diagrams and access their content without the need for a sighted mediator. This is achieved by taking UML design tool output in XMI format and converting and representing the diagram content in a format suitable for blind users. [S38] introduced an approach to creating speech interfaces for UML tools. They manually derived SpokenUML, a spoken imperative command language for the UML domain, from the UML metamodel by following language derivation guidelines. To assess the potential of speech-controlled user interfaces in software engineering, they developed the prototype software VoCoTo2 (VoiceControlTool). [S35] developed software called Latitude (Light and Accessible Tags Into plain Text using Universal DEsign). It automatically generates documents in HTML format and graphics and diagrams in SVG format from plain text enriched with lightweight and nonintrusive tags. These two XML-structured formats are particularly screen reader-friendly. [S19] proposed a Literate Modeling algorithm to validate element relationships in UML models using a set of validation constraints triggered by specific sentence characteristics. The synchronisation algorithm takes a UML model and an annotated piece of text as input, returning a set of problem markers. Each marker contains information about a specific inconsistency, including the location within the text, the error message, and additional guidance on error resolution. This approach has been implemented in a prototype editor for Literate Modeling called LiMonE.

Category 5: User-centred design. [S1] implemented a prototype using SwiftUI for the user interface. The prototype has gone through three iterations with different subjects in order to improve it based on the feedback received from the test subjects. [S33] developed a cross-modal collaborative tool based on the requirements collected through a workshop. The tool facilitates self-directed non-visual diagram editing and enables real-time collaborative work.

Empirical Studies: In [S17], empirical studies were conducted building upon the research of the HyperBraille project, which involved the development of a novel touch-sensitive tactile display. By expanding screen reader technology, they created a modular audio-tactile screen explorer, referred to as HyperReader, to provide blind individuals with access to on-screen information. They further extended this technology to develop a screen explorer for UML diagrams and evaluated the availability and accessibility of solutions for involving blind individuals in activities related to UML diagram creation and editing. [S25] developed a PoN-enabled version of the misuse case notation previously introduced by (Saleh and El-Attar, 2015) and conducted an accessibility evaluation in this paper. [S26] conducted empirical studies to compare block-based programming lessons made by high school interns and teachers.

Table 7
MDE Approaches used in different studies

Technology Stack	Studies
HTML	[S6, S7, S9, S16, S27, S30, S32, S35]
Tangile blocks	[S2, S3, S10, S11, S18]
Windows UI	[S24, S29, S33, S38]
Blocks + Sound	[S20, S36, S37]
Eclipse	[S22, S34]
SwiftUI	[S1, S8]
Visio 2007	[S23]
Tangible 3D	[S13]
Lego mindstorm	[S21]

Proof of concept: [S13] designed PRISCA, a proof of concept toolchain that automates the generation of a 3D-printable haptic UML model from an existing UML modelling tool. PRISCA also produces a 3D Braille representation of the textual annotations present in the graphical models. [S27] introduce the Flowgen tool, which produces flowcharts derived from annotated C++ source code.

Framework: [S12] outline their intentions to implement an accessible editing environment for UML models. The proposed plan involves utilising a textual concrete syntax integrated with a state-of-the-art editor to enable accessible interaction with UML models. Additionally, they provide an overview of their envisioned support for collaborative editing of UML models within teams.

Guidelines: [S14] formulated a set of criteria that programming environments should encompass to be inclusive for children with varying visual abilities.

Method: [S4] utilised Data Augmentation and introduced two distinct scenarios to assess the effects of employing the dataset with and without implementing data augmentation.

New Graphical Symbols: [S28] identified a set of possible improvements and new graphical symbols for WebML.

UML notation: [S15] created the UML4ALL syntax to accommodate the step-by-step working method of blind individuals who rely on screen reader technology.

4.3.4. *RQ_{3d}: Do they describe a tool prototype? What technology stack does the tool use? How does this impact accessibility provision?*

Table 7 summarises the technology platforms, as far as we could work out if not explicitly described in the primary studies. HTML or web-based interfaces are the most common technology platforms. These have the advantage that most web browsers provide some (limited) forms of accessibility support for some content e.g. magnification, screen readers, etc. For example, study S6 describes Model2gether, a web-based textual DSL with a screen reader enabling blind users to create UML models. S7, S9 and S16 all provide similar support tools for blind users to construct low-code models. However, these do not usually work on explaining graphical figures in detail nor interacting with graphical figures.

Several works provide tangible block-based programming tools, mostly targeted to vision-impaired users. For example, S2 and S3 describe PCUBE and its enhancements, and S10 describes TaBGO for creating Scratch programs. Several works describe tangible block-based interfaces with audio enhancements, such as S20 and S36. These combine a tangible block-based programming model with various forms of auditory feedback to vision-impaired users.

Several low-code programming tools use a Windows-style interface e.g. S24 describes OctoUML, as a CASE tool with a variety of accessibility enhancements. Several tools use Eclipse plug-ins to provide accessibility e.g. S34 describes ModelByVoice with voice control of Eclipse diagram editing. A Visio 2007 enhancement, S23, provides various accessibility enhancements to browse UML diagrams. A challenge with these 'desktop'-type interfaces is sometimes limited ability to provide a range of accessibility support due to the technology capabilities of the platform. While the web-based tools above often use browser plug-ins or third-party plug-ins to support accessibility, most desktop-based tools use custom augmentations.

One tool uses a 3D printed tangile interface to support sight challenged user accessibility, S13. One provides programming support on top of the Lego Mindstorm robot platforms, S21. Two provide iPad/iPhone interfaces using the Swift GUI toolkit. Like HTML-based tools, Swift provides developers with a range of accessibility-supporting programming APIs and supports natively.

4.3.5. *RQ_{3e}: How are the low-code approaches are evaluated? Is the generated code verified in some way?*

Our selected primary studies evaluated their approaches in several ways. Over half (n=19) conducted some form of user studies with both sighted and visually impaired users making use of the tool/approach proposed. Several studies conducted ongoing studies, while others collected feedback through online questionnaires or interview-based studies. Several studies used their approaches in the classrooms and some conducted experiments. Several papers only proposed evaluations without implementing them, Finally, [S30] and [S31] did not evaluate their approaches. Table 8 presents different evaluation types and the participants' types.

4.3.6. *User studies*

Nineteen of the studies evaluated their approach through user studies. [S1] compared their prototype with a textual modelling language (TML) called PlantUML. Their findings indicate that their solution is often more user-friendly for newcomers, significantly reducing the learning curve when compared to TMLs. It also offers advantages to visually impaired users who are already familiar with a TML, particularly when it comes to reading and navigating diagrams. Additionally, during the test cases, all visually impaired users preferred their prototype for both reading class and relationship information. Even visually impaired users without prior knowledge favoured their solution for tasks like creating classes, generating class information, and exporting images.

[S2] assessed and validated the efficacy of P-CUBE system through a series of tests: 1) They conducted a discrimination test to evaluate the effectiveness of the tactile information provided on the blocks. 2) They examined the process of creating a sequential program and assessed the ease of using the P-CUBE system. 3) They verified the procedures for loop and conditional branch programming, where a mobile robot follows a black line. To evaluate the P-CUBE system, they compared the creation of a sequential program using the P-CUBE system and PC software (Beauto Rover). In Prototype 1, the accuracy rate was 95 percent. Based on these outcomes, they developed Prototype 2, where all subjects were able to create a proper sequential program. In a separate study, [S3] organised a programming workshop for visually impaired children. They compared the process of transferring using P-CUBE2 with conventional systems, analysing it in terms of Therblig factors and transfer time. The results indicated that young children could effectively perform transfers using Pro-Tan, leading to the conclusion that visually impaired children can easily manage program transfers.

To assess the ease of operating P-CUBE, [S5] initiated the evaluation process by examining how sighted individuals create a sequential program. They then proceeded to compare the ease of operating P-CUBE with PC programming software. All participants completed the task of creating a sequential program using both programming tools. The questionnaire results indicated that individuals with no prior programming or flowcharting experience found it easier to create a sequential program using P-CUBE. Furthermore, the workshop results demonstrated that users with impaired vision could also monitor the operating status of the mobile robot using their remaining sight. In a separate evaluation, [S7] conducted a user testing session to assess the interface's performance using the success rate per task method. The results suggest that it is feasible for visually impaired users to have awareness of the actions and activities of others in a shared workspace. During the usability test, participants were able to discern who was involved in specific tasks and where those activities were taking place.

A usability study was conducted by [S11], to assess a preliminary version of their prototype. Their findings indicated that it is possible to enhance the accessibility of visual programming languages for users with visual impairment and low vision by incorporating the following features: 1)

Table 8
Evaluation types in different studies

Evaluation	Participants #	Participants' category	Studies
User studies	4	Completely blind to partially blind	[S1]
	16 (prototype 1) + 10 (prototype 2)	Sighted users who wore eye masks	[S2]
	13	High school students. Seven used braille and six used extended character points	[S3]
	10 (test 1) + 4 (test 2)	test 1: Sighted participants. test 2: High school students. Three with very poor vision, one totally blind.	[S5]
	5	Sighted participants	[S7]
	9	Participants with visual impairments and low vision	[S11]
	7 (study 1) + 2 (study 2)	Study 1: Visually impaired children with special needs, Study 2: Same group as Study 1 + focus group with (six from Study 1 + one new special needs educator and an IT instructor)	[S14]
	2	Blind participants	[S16]
	3	One blind and two sighted	[S17]
	16	Six design sessions with five blind and low-vision students (middle and high school), eight teachers, and three staff members, including two Braillists (Braille transcriptionists) and one preschool staff member	[S18]
	4	Varying vision needs: low vision to total blindness.	[S20]
	10	Computer science students or graduated computer engineers	[S22]
	7	Blind participants	[S23]
	30	Two user studies with two post-docs, 13 PhD and 15 M.Sc. students)	[S24]
	34	Blind and partially-sighted users	[S29]
	5	Four blind and onr sighted participants	[S32]
	5	Two blind and three sighted participants	[S34]
	12	Blind programmers	[S36]
	5	Children aged 5-10 with visual impairments	[S8]
Ongoing studies	3	Professional pairs; employees in the head office of a Children and Families Department an international charity, and a private business.	[S33]
	5	The authors of the paper and three other users	[S38]
Online Questionnaire	14	Seven sighted (six blind and one severely visually impaired) and seven visually impaired participants	[S15]
	84	Red—green colourblind IT professionals	[S25]
	45	Software engineers (30 students and 15 experts in Software Engineering)	[S28]
	20	Companies	[S17]
Interview-based studies	1	Teacher of the visually impaired	[S8]
	2	Visually impaired users	[S9]
Class-based studies	5 (series 1) + 5 (series 2)	One blind and 4 sighted learners in each of the series	[S6]
	115	Two participant groups new to creating K-12 computing curricula (Group 1: 29 high school interns with prior programming experience but not formal pedagogical training. Group 2: 86 teachers with formal pedagogical training but no background in programming.)	[S26]
	Not mentioned	Their students for three years	[S35]
Experiments	N/A	N/A	[S4]
	N/A	N/A	[S13]
	N/A	N/A	[S19]
	N/A	N/A	[S27]
	1	Member of the team who was colour-blind + a colour-blindness simulator	[S37]
Evaluation proposals	N/A	Young visually impaired and sighted students	[S10]
	N/A	Industrial software engineering team that includes visually impaired engineers	[S12]
	N/A	High school students and teachers	[S21]

Implementing focus navigation to facilitate movement between different sections of the interface. 2) Incorporating audio effects to signal valid connections. 3) Ensuring compatibility of screen readers with block labels and symbols, such as directional arrows. During their evaluation, participants encountered certain challenges when using Scratch, including difficulties in navigating between interface sections and receiving feedback regarding valid block connection points. [S14] conducted a workshop organised through two studies.

In the first study, educators emphasised the significance of incorporating a robot with its physical and socio-emotional attributes. They also suggested that providing a confined space for organising instructions would enhance their programming activities. Furthermore, educators appreciated the use of a map, such as one with DOC (Dynamic Obstacle Course), which allowed children to explore spatial boundaries and foster orientation skills. While the solutions presented in the first study were found to be inaccessible, they exhibited qualities that could be adapted for an accessible programming environment. The second study's findings indicate that, from a physical access perspective, visually impaired children could engage in spatial training activities similar to their sighted peers.

[S16]' user tests results indicate that the AWMo approach is feasible for enhancing collaboration and communication between sighted and visually impaired users during software modeling activities. [S18] conducted six design sessions where the researcher conducted interviews with the group members regarding their usage of StoryBlocks and discovered that the holistic approach of integrating tangible programming with audio narratives can serve as an engaging method for instructing foundational programming concepts inclusively and collaboratively. [S20] evaluated their solution and found some issues listed in detail in the limitations section. [S22] conducted a pilot experiment and found that their novel interaction model expedited user engagement and enhanced the overall user experience on the platform. Nonetheless, certain aspects of the interaction may require further enhancement in the future. [S23] conducted evaluations and compared their newly developed representation with the common method blind people utilise sequence diagrams, that is non-visually through verbalisation. The findings indicated that computer scientists with visual impairments can navigate tactile UML sequence diagrams on a flat display, but they identified issues that arose during navigation.

[S24] performed two user studies to assess and compare the usability and efficiency of two versions of OctoUML: one with voice recognition enabled (OctoUML-V) and the other without (OctoUML-Lite). Their findings indicated that (i) users had a positive perception of usability, (ii) voice interaction was primarily preferred for text input, and (iii) the integration of voice interaction in software design environments improved the efficiency of the software design process. OctoUML-V received greater user appreciation and reduced the time required for naming software design diagram elements. [S29] evaluated their system and discovered that the system was well-received, with participants successfully completing UML tasks. In a user study conducted by [S32], all blind participants expressed enthusiasm about the tool and provided positive feedback. One participant even used the tool a week later to collaborate with colleagues in creating a use case model for a course. [S34] carried out a preliminary assessment session, which, despite the limited number of participants, yielded promising and confident results, providing valuable feedback for future improvements. [S36] conducted a study demonstrating that Accessible Blockly effectively assists users in reading and understanding block-based code. Participants found it user-friendly and less frustrating when navigating block-based programs. They also expressed enthusiasm for using the keyboard and screen reader to navigate block-based code, highlighting the accessibility of block-based programming. Lastly, [S17] conducted a user study and survey. The results indicated that not all UML diagrams are accessible, posing a challenge to the inclusion of blind developers in the industry. Except for TeDUB joystick, none of the proposed software solutions were available for download.

4.3.7. Ongoing studies

Two papers conducted ongoing studies. [S33] delved into cross-modal collaboration between visually impaired and sighted coworkers. Their primary objective was to explore the dynamics of cross-modal collaboration in a workplace setting and assess the extent to which the tool they developed could effectively support this collaboration in real-world scenarios. This study unearthed a range of challenges associated with cross-modal technology's impact on collaborative work. These challenges encompassed aspects such as representation coherence, collaborative strategies, and the facilitation of awareness across different sensory modalities. Leveraging their observations, the authors delineated an initial set of preliminary design recommendations, aimed at providing guidance and enhancing the design of support systems for cross-modal collaboration. [S38] conducted preliminary tests over several months, with users who gradually became proficient in utilising the VoCoTo speech interface in conjunction with Rational Rose. These initial tests paved the way for more formal, small-scale user evaluations to gauge the practicality of their approach. Their findings led to the conclusion that their approach was indeed viable. UML proved to be an ideal candidate for speech recognition, and the maturity of speech recognition technology made it a feasible option for enhancing the utilisation of UML tools. Of particular significance from the preliminary tests were the language improvements identified. Initially, some novice users encountered difficulties with certain aspects of the speech interface, such as configuring complex multiplicities for associations. To overcome this problem, they added synonyms.

4.3.8. Online Questionnaire

Four studies designed surveys to collect users' feedback. [S17] utilised a combination of user studies and surveys, as previously described. To assess the usability of the UML4ALL syntax, [S15] conducted an online questionnaire-based empirical study. These findings strongly indicate that the UML4ALL syntax demonstrates comparable usability for both sighted and visually impaired individuals. [S25] conducted an empirical study aimed at evaluating the cognitive effectiveness of the RGD version of the PoN-enabled misuse case notation in comparison to the original misuse case notation. This study involved a comparison of speed and accuracy, with results confirming the continued cognitive effectiveness superiority of the RGD version of PoN-enabled misuse case notation over the original notation. Furthermore, the study demonstrated that the PoN principles' positive influence on cognitive effectiveness remained intact when PoN-enabled diagrams were viewed by individuals with red-green colour blindness. [S28] performed empirical validation, employing a questionnaire to gather feedback on alternative graphical symbols. The questionnaire responses indicated a preference for the new symbols among the majority of study participants.

4.3.9. Interview-based studies

Two of the papers used interview-based studies. In the study conducted by [S9], they employed a combination of interviews as their initial data collection method and observation of the AWMo tool while participants completed a predefined set of tasks as their secondary data collection technique. Each subject participated in two interview sessions, conducted both before and after using the tool. The study's findings indicated that their approach does not result in additional time spent on the activity. Notably, once a textual language description is created, the graphical representation is readily available for sighted users. This suggests that AWMo is a feasible option for enhancing collaboration between sighted and visually impaired individuals in software modelling activities. [S8] conducted a semi-structured interview with a teacher who specialises in teaching visually impaired children. The teacher was questioned about their experiences teaching children how to use technology and various assistive tools, including screen readers, zoom, and connectable braille displays, on touchscreen devices. Additionally, feedback was solicited regarding different tool designs. Following this, five children with visual impairments were recruited to participate in the study. Based on the children's experiences and feedback, certain tool modifications were made, particularly in response to difficulties some of the children encountered.

4.3.10. Classroom-based studies

Three papers conducted studies in their classroom. In the study by [S6], a pre-survey and interviews were conducted before the lectures in the classroom setting. Following the lectures, learning assessment involved both group work and an exam. Participants' perceptions were gathered through a post-survey and interviews. For the first series of activities, the diagrams created during group work were error-free and suitable for the specified system description. Participants appreciated the coordination and visibility mechanisms of Model2gether, which contributed to the success of the lecture. Notably, participants who had previously attended similar face-to-face courses did not observe any significant differences in classroom dynamics due to the participation of visually impaired students. In the second series, after the lectures, all participants exhibited improved grades. The results suggest that the tools used enabled visually impaired students to learn and collaborate effectively with sighted peers without disrupting the typical dynamics of a distance education scenario. In the study conducted by [S26], the research compared block-based programming lessons related to subject areas, focusing on two participant groups new to developing K-12 computing curricula. They assessed a total of 113 lessons for their inclusion of scaffolding, teacher accessibility, equity, computing and subject area content, and assessment-focused components. Comparing lessons created by teachers (who had less coding experience but more pedagogical expertise) to those created by a group of interns, the results revealed that both groups faced challenges in incorporating equitable practices and opportunities for culturally responsive and identity-affirming activities. However, they excelled in including assessment-focused items. Additionally, teachers and interns demonstrated the ability to provide substantial support for coding and content knowledge. In conclusion, both groups of curriculum creators successfully developed computing-infused lessons. [S35] utilised their software within their classroom for three years, and it proved to be highly successful with their students. The majority of students indicated a desire to continue using the tool in the future. They also encouraged other teachers in their department to adopt the same tool, emphasising its convenience. Furthermore, they appreciated the description file, which allowed for a clear establishment of the meaning associated with classes and associations. Notably, as a blind teacher, one of the authors gained autonomy and could present live diagrams to students, effectively illustrating problems and responding to questions. Consequently, the inclusion of this software was deemed to be proven and highly valuable.

4.3.11. Experiments

Five of the studies conducted experiments. In the study by [S4], experiments were conducted using various algorithms, and the results were measured in terms of accuracy and standard deviation between folds. The findings indicated that the utilisation of transfer learning contributed to achieving satisfactory results even when working with limited data. Nevertheless, there is still room for improvement in successfully classifying the UML diagrams addressed in this research. [S13]'s toolchain was validated using sample models created with Visual Paradigm UML, including models developed for projects in collaboration with industrial partners. However, the specific results of this validation are not included in the paper. [S19] analysed the runtime complexity of their methods, revealing the feasibility of incorporating the proposed solution into contemporary CASE (Computer-Aided Software Engineering) tools. [S27] tested their initial implementation of Flowgen on a variety of source files, encompassing code with nested if statements, loops, function and class method calls, annotations with different zoom levels, and links to be followed during browsing. They contend that their tool, offering two views—a high-level semantic view and another focused on code-level implementation details such as branching and important variables and method names related to annotated activities—proves beneficial to users. It serves as a common ground for specialists from various backgrounds to collaborate more efficiently, which is an attractive feature in computational physics research. [S37] conducted testing of their colours using a colour-blindness simulator and through evaluation by a team member with colour blindness to ensure the distinguishability of the blocks from each other and the background. However, specific results of this testing were not reported in the paper.

4.3.12. Evaluation proposals

Finally, three papers proposed evaluations, but have not implemented them. [S10] outlined a forthcoming user study aimed at assessing the usability and ease of use of their solution among young students with visual impairments as well as sighted students. The study intends to gather user feedback concerning aspects of usability, satisfaction, and cognitive load. [S12] put forward a strategy for an industrial software engineering team that incorporates engineers with visual impairments. The strategy involves the continuous evaluation of their outcomes. As for [S21], their approach is presently in use across various schools. They have devised a plan to gather statistics on block usage from high school students and teachers who are already utilising MUzECS. This data will be collected through the use of surveys.

RQ3 Summary. Most of the selected primary studies build on top of UML (42.1%) or block-based programming (28.9%). They mostly do not consider any design guidelines (86.8%) and have not used transformation methods or code generation (92.1%). The study types are divided into *Prototype* (73.6%), *Empirical studies* (7.8%), *Proof of concept* (5.2%), *Framework* (2.6%), *Guidelines* (2.6%), *Method* (2.6%), *New graphical symbols* (2.6%), and *UML notation* (2.6%). The studies that propose a new prototype, are categorised into those *Providing graphical and textual views*, *Extending block-based tools*, *Using assistive technologies*, *onverting to accessible versions*, and *User-centred design*. The studies are evaluated in different ways. The evaluation types are categorised into *User studies* (50%), *Ongoing studies* (5.2%), *Online Questionnaire* (10.5%), *Interview-based studies* (5.2%), *Class-based studies* (7.8%), *Experiments* (13.1%), and *Evaluation proposals* (7.8%).

4.4. RQ$_4$: What are key future research directions for accessible low-code tools?

4.4.1. RQ$_{4a}$: What are the key strengths identified in the selected primary studies?

Our selected primary studies provide multiple strengths, as reported by the authors. Here, we just provide interesting examples of the strengths that can motivate future researchers to uptake the current solutions. The main strengths discussed are making MDE accessible and enabling collaboration among sighted and visually impaired users.

[S1] introduces an innovative approach that provides a **novel method for visually impaired users to navigate and interact with UML** class diagrams. By combining this method with VoiceOver, even those who are unfamiliar with these diagrams can read and create content. This approach significantly **enhances the accessibility of UML class diagrams for newcomers** and **facilitates more comfortable collaboration between sighted and visually impaired users** by presenting all the content in a unified view that links the various diagram elements. According to [S2], the P-CUBE system offers three notable advantages. First, it utilises blocks with tactile information, such as surface gaps, to ensure the system **can be utilised by visually impaired and younger individuals**, specifically those aged 5 to 10 years. Second, the system's interface, based on RFID-tagged blocks, is **robust and resistant to damage**, which is especially advantageous for the younger and visually impaired target audience. Furthermore, the P-CUBE system is **cost-effective**. Lastly, P-CUBE eases the burden on teachers when preparing programming lessons.

[S5] believe that their approach is **easy for beginners to use** since It enables users to focus on designing algorithm structures without the need for tedious PC operations. [S6] suggest that the tools employed enabled visually impaired participants to learn and collaborate with their sighted peers **without the latter realising there was a visually impaired member** in the group. [S7] noted that the participants could effectively discern the actions performed by individuals. Furthermore, it was feasible to identify the awareness categories that required further refinement. [S8] established design principles for creating block-based environments and touchscreen applications catering to children with visual impairments, derived from interviews, formative testing with children and educators, and the Blocks4All application. Furthermore, they have **made both the source code and the application readily accessible** to the public. The AWMo tool by [S9] exhibits a **highly adaptable architecture**, allowing forthcoming contributors to create and explore novel methods for visualising and modifying models without disrupting the current ones. [S10] introduced a **straightforward approach to boost students' computing skills**, irrespective of whether they have visual impairments or not. [S11] introduced a prototype **accessible to users with visual impairments and low vision**. [S12] argue that synchronisation between textual and graphical editors enables the **freedom to choose tools** based on each team member's specific requirements.

[S13]'s PRISCA tool places a strong emphasis on the **reusability** of diagram features (such as similar shapes and line types), **extensibility** to various diagram types, and **adaptability** to a wide range of modelling tools and 3D printers. [S14]'s approach encourages the emergence of different behaviours and opportunities, **fostering discussion** with educators. [S15] demonstrate a **more inclusive design** of work processes, eliminating the need for manual translation of visual documents into accessible formats during subsequent stages. [S19]'s approach is seen as a valuable contribution to ensuring **consistency in documents** within Literate Modeling. [S20]'s spatially organised, yet accessible, block-based coding platform simplifies **collaboration and learning between sighted and non-sighted** peers. [S21] have successfully developed **cost-effective** hardware, software, curriculum, and resources as an alternative for the sixth module of the Exploring Computer Science curriculum. [S22]'s innovative solution enables modelling in a specific language **solely through voice commands**, offering a **flexible and customisable** interaction model while enhancing and expediting interaction with haptic features.

[S23]'s findings suggest that a combined representation of UML diagrams, incorporating both graphical and linear elements, is valuable, particularly when there is a need to **quickly access information** for specific tasks. [S24]'s utilisation of the voice interaction modality in software design environments results in **increased efficiency in the software design process** by reducing the time required to name classes and packages in UML class diagrams. [S25]'s adherence to the comprehensive set of principles in the PoN enables the new notation to maintain its cognitive effectiveness superiority over the original notation, even when colour perception is limited. The main advantage is to **read the diagrams more easily**. [S26]'s strength is **gaining insight into the additional knowledge and training** required to assist content creators in developing improved and more inclusive computing lessons for non-computer science subjects. [S27] delivers a straightforward and visual summary of intricate numerical algorithm implementations. Flowgen creates a shared platform for experts from various disciplines to **collaborate with increased efficiency**. [S29] reported that the users were comfortable with the tool and it was **tested with a large group of blind users**. [S30]'s main benefit is to provide a **collaborative web-based tool** for both blind and sighted users.

[S31] anticipate the full elimination of the requirement for a secondary person to interpret diagrams, thereby **enabling collaboration** in an MDE context. [S32]'s tool has two interfaces for sighted and blind users, which allows **collaboration among the two groups**. [S33] collected requirements from real end-users and developed a tool that provides **options for collaborative usage**. [S34] **makes model-driven development accessible** for blind users. [S35] enables **collaboration between blind and sighted participants**. The approach has been used by students for several years. [S36] introduced their initial endeavour to create a block-based programming library that is **accessible**, employing keyboard and screen reader interaction. [S37]'s strength is to **provide accessibility features** for a well-known block-based tool, Blockly. [S38]'s approach can be **applied to comparable domains**, including other visual modelling or programming languages. It is also usable with the existing UML CASE tools.

4.4.2. RQ_{4b}: What are the limitations discussed in the selected primary studies?

In terms of the limitations, some of the selected primary studies have not discussed any limitations. Among those who discussed the limitations, they mostly reported *Limitations in the approach*, followed by the *Limitations in the evaluation*. Other categories are: *Limited data analysis*, *Complexity of the approach*, *Small sample size*, *Limited data*, *Requirement gathering*, and *Lack of user study*.

Limitations in the approach: Seventeen studies discuss limitations related to their approach. [S7] reported limitations in the auditory interface and lack of features for the prototype, such as the option to replay the most recent auditory cue and browse through the various items. [S8] examined only a limited set of design options. They were also constrained by only making modifications that could be embraced by all creators of block-based environments. Due to the limitations in the approach developed by [S11], participants faced some barriers during the evaluation. [S13] reported several limitations in handling the textual annotations of UML, such as translating the text directly to Braille, limitations in the current graphics library/utilities, and the technique's portability. [S18] pointed out that tangible blocks have constraints in displaying labels, as the blocks may lack the space for Braille or the users might have difficulty reading Braille. A significant issue raised was the potential challenge in situations where the visual aspects of the system outperform the tactile elements, potentially causing difficulties in collaborative work between blind and sighted participants, as observed during some study sessions. Another drawback is that StoryBlocks does not facilitate practices like reusing, remixing, or abstracting and modularising.

In the current state of the prototype developed by [S19], only sentences containing references to model elements are parsed. Pronouns referring to model elements are not considered in the current model elements, and they need to be referred to by their specific name in the text. [S20] has identified limitations related to navigation issues, such as the challenge of determining the start of a new line of code, as identified during usability testing. Additionally, there is no functionality to move blocks across the workspace. The approach proposed by [S21] faces limitations as it cannot install Arduino software, leading the authors to develop a web-based solution through a Chrome browser web page and web app to overcome platform restrictions. Further evaluation of this solution is planned. During the empirical experience organised by [S22], users encountered limitations such as speech recognition not being entirely reliable and issues with the comprehensibility of the synthesiser's voice at times. It is important to note that [S23]'s approach does not provide the capability to edit the diagrams.

[S24] highlighted that several factors, including microphone distance, ambient white noise, and variations in human pronunciation, can influence the effectiveness and accuracy of the voice recognition system. These factors may consequently impact the improvement in the efficiency of the software design process. [S25] discussed the issues related to the approach that participants reported in their evaluation. According to [S28], they noted that many of the enhancements suggested by their analysis should have been implemented during the initial stages of WebML development. However, as the product is now well-established and widely distributed, most of these improvements are considered financially unfeasible. [S33] identified limitations in their approach and came up with recommendations. They plan to conduct further studies. The main limitation reported by the [S34]'s participants in the preliminary experiment, was linked to technical failure, specifically the error rate of the underlying speech recognition technology during the execution of ModelByVoice. [S38] reported limitations of their approach in their evaluation results and future work. [S5] found that in the workshop with visually impaired users, participants found it challenging to differentiate between the IF and LOOP blocks, and they expressed the desire for independent execution of data transfer tasks. After the evaluation, these features were revised to address the concerns.

Limitations in the evaluation: Eight of the studies discuss limitations in their evaluations. [S1] reported that the users were aware of what tool the researchers had developed given they were familiar with the tool they used for comparisons. This might impact the results in their favour. The study conducted by [S14] concentrated exclusively on a cohort of visually impaired children, without involving their sighted counterparts. Additionally, it did not evaluate computational thinking (CT) concepts like loops or conditionals. The discussions with educators only briefly touched upon these forthcoming challenges. The research was confined to a single brief session, featuring a limited range of spatial activities and CT concepts, which might have been influenced by a novelty effect on children's interest. The questionnaire designed by [S15] can solely provide a general usability trend. It does not allow for the interpretation of individual items within the questionnaire. Furthermore, specific usability issues related to the UML4ALL syntax could not be deduced from the outcomes. [S23] reported several limitations in the user study. [S26] did not include their interview or self-report data. In addition to exit interviews, they conducted daily feedback surveys with teachers and collected motivation essays from interns in both groups. For the upcoming phase of their evaluation, [S29] plan to incorporate larger diagrams. The diagrams used in the initial phase were intentionally designed to be small enough for users to 'chunk' or hold in memory entirely, and this adjustment is expected to yield different strategies and outcomes. [S37] did not study their approach's features by their participants. According to [S36], it is important to note that each navigation scheme was assigned to a specific set of programs for the entire study. This approach could have potentially impacted the quantitative results and their statistical significance.

Limited data analysis: Two of the studies report limitations in their data analysis. [S1] reported that it can sometimes be challenging to obtain precise measurements for data points related to questions, doubts, and errors. [S16]'s analysis focused on assessing the navigation strategies employed by the participants within the application and its textual editor. The researchers suggest that the potential for non-sequential navigation warrants further investigation.

Complexity of the approach: One study discusses the complexity of the approach as its limitation. According to [S16], the most significant challenge of the AWMo approach for modelling by blind users is anticipated to be textual navigation. While the approach presents an innovative way for blind and visually impaired users to edit models, it becomes more challenging as systems and models grow in complexity and size. Therefore, strategies to alleviate the memory burden on users may be necessary in such cases.

Small sample size: Five of the studies report small sample size as their limitation. A small number of tested subjects is the second limitation reported by [S1]. [S8] noted that the challenge of recruiting participants from their target population led to a relatively small sample size. This resulted in participants spanning a wide range of ages and abilities, which made it challenging to draw meaningful comparisons among them. [S26] recognised that the use of small or imbalanced sample sizes might have had an impact on the analyses conducted. [S34] reported the limited number of blind users and the limited number of modelling languages used in their experiments. [S36] noted that the study involved a limited number of participants.

Limited data: One study reports limited data as their limitation. [S4] reported that they utilised a set of only 200 images per diagram type, as there was a shortage of available images for certain categories. To address this limitation, they employed data augmentation methods to increase the count to 1000 images per category.

Requirement gathering: Requirement gathering is reported in one study as the limitation. One challenge encountered by [S31] when engaging with computer programmers with disabilities was the fact that some of them never had the opportunity to learn UML class diagrams due to their unique requirements and the absence of accessible tools. This presented a difficulty in collecting the initial requirements for AWMo.

Lack of user study: Lack of user study is reported as the limitation by [S35]. The tool is not being tested for the purpose of the study.

4.4.3. *RQ_{4c}: What are the future works?*

The main future works discussed by different authors are: Extending the tool, Conducting further experimentation, Conducting further research, Expanding the solution to other types of UML diagrams, Conducting usability tests with users from the industry, Making the tool open-source, Developing collaborative modelling, and Enabling the editing of diagrams.

Extending the tool: Twenty-one studies propose an extension of their tool as their future work, mostly based on their evaluation results. [S2] aims to enhance the program transfer process, streamlining it for visually impaired individuals to transfer program data to the mobile robot without undue complexity. They also intend to introduce a feedback function with auditory cues to keep the user informed about the mobile robot's movements. [S4] emphasised that further studies are needed to address the complexity of UML diagram classification, particularly concerning class diagrams. They also recognised the necessity of enhancing the data augmentation transformations for improved results. [S7] envisions adding an auditory interface and new features to their prototype, such as the capability to replay the last sound cue and navigate among different artifacts, to enhance the user experience and achieve better outcomes. [S8] has plans to delve into unresolved questions, such as navigating more intricate hierarchies of nested code, accommodating multiple "threads" of program code, and integrating speech-based commands or other gestures effectively. [S11] intends to implement the participants' recommendations, which include incorporating the music environment, to enhance their project based on valuable user feedback.

[S12] is working on implementing synchronisation mechanisms between graphical and textual editors. [S13] focus on investigating improved methods for providing textual descriptions that assist visually impaired users and foster collaboration by incorporating interactive feedback and exploring emerging technologies. [S14] dedicates to developing or enhancing current solutions and moving towards inclusive classroom activities with children of varying abilities. [S18] plan to expand the existing StoryBlocks system, further supporting creative endeavors and problem-solving while extending the application of this approach to new domains. [S19] investigates the relationship between UML elements and their connection to parts of speech to enhance the parser's accuracy. This includes the potential training of the parser on an appropriate corpus and integration with a more mature framework. [S20] aim to introduce features that allow users to control speech speed and verbosity to align with their audio comprehension needs. The plan also includes making various workspace elements accessible and providing audio feedback on hierarchical toolbox information. Additionally, they are working on adding screen-reader support to blocks for experienced screenreader users and implementing a zoom feature for users with low vision.

[S21] intend to introduce additional features to enhance user creativity and expand the curriculum further. [S22] explore new interaction paradigms, possibly domain-specific, and laying the foundation for understanding the semiotics of sound in model-driven solutions. [S26] prepare for more in-person field testing of lessons created by novices to assess their broader impact on student learning and engagement. [S38] aspire to build a more comprehensive toolset, including a wider range of target tools, to evaluate the speech interface in real software engineering projects beyond diagram drawing tasks. Their plans also encompass tool enhancements and applications in other visual languages. [S29] aim to further developing their system based on their findings to provide a valuable UML tool for blind individuals working in software development. [S24] aim to leverage more sophisticated recognisers to mitigate factors that might compromise the recognition process, reducing the voice recognition failure rate.

[S32] plan to introduce new models, such as the UML class model and entity-relationship model, while also adding new features like support for tactile devices, customisations for domain-specific languages (DSL), and coordination mechanisms. [S36] intend to enhance the screen reader output by refining verbal descriptions and increasing the quantity and variety of information available through the screen reader. [S34] are working on providing users with the option to select their preferred input method, whether it is voice recognition, keyboard input, or a combination of both for modeling activities. They are also looking to introduce an intermediate layer that supports various speech recognition APIs and develop an editor capable of converting XMI diagrams into graphical formats. [S28] plan to conduct a comprehensive analysis of cognitive effectiveness, building upon the key findings and improvements identified in WebML. They aim to explore potential visual dialects that could enhance WebML's user-friendliness for diverse user groups.

Conducting further experimentation: This is discussed in sixteen papers as their key future work. [S1] intend to further explore and experiment with their prototype, as the current paper primarily describes the initial prototype. [S3] plan to validate the usefulness of their tool for visually impaired individuals and continue assessing its impact on learning. [S5] aim to conduct a more extensive evaluation of the learning effects of P-CUBE and compare them to conventional PC software. [S8] is preparing for a formal evaluation of Blocks4All to thoroughly assess its effectiveness. [S9] intend to expand their case study with a larger pool of subjects and to design various experiments aimed at evaluating the graphical view and usage by sighted users. [S10] will commence the experimental phase to assess the usability of their prototype. [S12] will proceed to conduct a detailed evaluation of the twelve existing textual syntaxes for UML in terms of their impact on accessibility.

[S15] intend to assess specific usability issues through a real end-user test and observations by test facilitators. [S16] plan to prototype and evaluate alternative navigation strategies for the textual language of AWMo, allowing users to navigate through the textual content in a non-sequential manner. [S22] aim to extrapolate and implement the experiments on a larger scale. [S25] plan to conduct further empirical studies involving other PoN applications empirically validated using their respective coloured versions of diagrams to explore the effectiveness of the PoN framework. [S28] plan to design additional experiments to identify improvements that would be most beneficial to various user groups. [S30] will conduct usability experiments involving blind and sighted students in a simulated academic environment to validate and refine the developed prototype. [S32] plan to perform controlled experiments with a more extensive user base. [S35] plan to test their tool, particularly among young individuals with visual impairments who are learning computer science. [S37] plan to initiate formal studies of the accessibility features to fine-tune the system and enhance user access.

Conducting further research: Eight of the studies aim to conduct further research. [S17] plan to conduct further research on the accessibility of UML for visually impaired and blind people. [S26] plan to look at the remaining projects they have from teacher-intern collaborative teams, to gain a deeper insight into the impacts of collaborative team design that complements project development. [S27] plan to extend the use of Flowgen across various aspects of the Vincia collaboration and make refinements based on their growing experience. [S30] recommend that future researchers investigate suitable interfaces and interaction styles for facilitating collaboration between blind and sighted participants. [S31] outline potential areas for future investigation by identifying various types of data that could be collected once the tool is fully developed. [S24] intend to investigate the impact of incorporating multi-touch and remote collaboration methods into OctoUML and assess their effectiveness in enhancing the software design process. [S37] plan to extend their approach to other hybrid text and block-based programming languages, such as Pencil code. [S34] aim to establish a systematic approach, similar to the PoN framework, for evaluating the usability of audio concrete syntax and interaction models.

Expanding the solution to other types of UML diagrams: Expanding the solution to other UML diagram types is another future direction discussed by three studies including [S1]. [S13]'s current focus is on UML modelling, but their ultimate objective is to broaden the applicability of their work to encompass other widely-used diagramming notations, including those used in mathematical graphing, such as MATLAB. [S15]'s future work plans to take the concept initially developed for UML and apply it to various types of graphs commonly utilised in different applications, e.g. PowerPoint.

Conducting usability tests with users from the industry: [S38] acknowledge the necessity for comprehensive usability tests, with a preference for industry users' involvement as part of their future agenda.

Making the tool open-source: [S9] plan to transform AWMo into an open-source project and release a technical manual to provide support for individuals interested in contributing to the tool.

Developing collaborative modelling: [S22] plan to conduct further research for collaborative modelling by voice.

Enabling the editing of diagrams: [S23] plan to conduct future investigations to enable the editing of diagrams.

RQ4 Summary. Most of the claimed strengths are *making MDE accessible* and *enabling collaboration among sighted and visually impaired users*. We categorised the limitations discussed in the studies in eight categories: *Limitations in the approach* (44.7%), *Limitations in the evaluation* (21%), *Limited data analysis* (5.2%), *Complexity of the approach* (2.6%), *Small sample size* (13.1%), *Limited data* (2.5%), *Requirement gathering* (2.6%), and *Lack of user study* (2.6%). Based on these limitations, different studies propose future directions. We have categorised the future directions into *Extending the tool* (55.2%), *Conducting further experimentation* (42.1%), *Conducting further research* (21%), *Expanding the solution to other types of UML diagrams* (7.8%), *Conducting usability tests with users from the industry* (2.6%), *Making the tool open-source* (2.6%), *Developing collaborative modelling* (2.6%), and *Enabling the editing of diagrams* (2.6%).

5. Future Research Recommendations

Need for to explore new approaches: There is limited research on the accessibility of low-code approaches, and the existing research mainly focuses on UML and block-based programming. There is a need for other low-code approaches to be accessible to diverse users, mostly those with visual impairments. The existing research mostly only pays attention to the readability of the diagrams, not their editability. Therefore, there is a need to better support accessible editing not only accessible reading of their models.

Need for more advanced accessibility extensions: There is a need for other types of extensions to make low-code approaches more accessible. Most of the current solutions use text-to-visual or visual-to-text conversion, which makes them quite complicated. Text-to-voice is another approach, but which may make a solution non-accessible for those with hearing impairments.

Limitations in evaluations: Most of the selected primary studies were only able to recruit a small set of participants. They were also not always able to recruit participants with special needs and had to simulate the impairments. Recruiting users with special needs for evaluations is required for future research in this field. We encourage researchers to build connections with industry, special needs schools, and organisations to further enhance the research.

The need for collecting and analysing users' point of view: There is limited knowledge of the actual challenges that low-code users with special needs have. There is research conducted by (Albusays and Ludi, 2016) on understanding the programming challenges faced by visually impaired developers, but not on the accessibility and usability of low-code IDEs. We recommend other researchers conduct empirical studies through surveys and interviews to understand the pain points of the low-code users. Other data sources, such as low-code app reviews and software repositories can also be used as a resource to investigate the challenges the users face.

Improving low-code developers' awareness: There is an urgent need to make low-code IDE developers aware of the accessibility issues. Low-code developers do not necessarily have the same characteristics as the users of those applications, in terms of suffering the same challenges and impairments. There is a need to communicate the challenges collected through the surveys and interviews conducted with the low-code users.

Exploring the impact of the technology stack on the accessibility of low-code platforms: Technological advances such as cloud-based architectures, AI/ML, and enhanced awareness about UI/UX significantly contribute to the success of low-code approaches. Low-code platforms that leverage state-of-the-art front-end technologies have a greater potential to meet accessibility requirements. This could be attributed to the existence of web-based accessibility guidelines, such as the Web Content Accessibility Guidelines (WCAG). In contrast, tools generated by platforms still relying on obsolete technology stacks, such as the desktop version of Eclipse-based Sirius, may face limitations in providing accessible solutions. This observation underscores the necessity of adopting advanced technological stacks to develop accessible low-code solutions. Further research is needed to explore the impact of these technologies on the accessibility of low-code platforms and to drive the development of more inclusive and effective technologies.

6. Threats to validity

While this SLR adhered to the established methodology for conducting SLRs in the software engineering field (Keele, 2007), it is important to acknowledge that our review does have several limitations primarily linked to our search approach and the data extraction procedure. In this section, we will explore potential factors that may affect the credibility of our study.

The most apparent source of potential bias that could impact the construct validity of this study pertains to the limitations in the search and study selection process, as outlined by (Shahin et al., 2014). A noteworthy challenge in this context is the varied terminology and categorisation of low-code approaches. To address and mitigate the potential impact of this challenge on our search strategy, we carried out multiple rounds of trial searches across five reputable digital libraries. Additionally, we employed the PICOC criteria to ensure a comprehensive search. This strategy proved effective in identifying significant papers within the relevant field. Nevertheless, it is important to note that the systematic rejection of certain papers with unvalidated references in their metadata, such as titles, abstracts, and keywords, is an inherent aspect of systematic literature reviews and was encountered during our study.

The key contribution of this study is to understand the accessibility approaches the current low-code platforms have taken into account. To prevent systematic errors, we created a data extraction template to ensure consistent data collection and analysis, facilitating the response to the RQ in our SLR; an independent reviewer extracted data from 10 percent of the selected primary studies independently and compared the extracted data to check for contradicting information; and we followed open coding technique (Glaser and Strauss, 2017). All the attributes were reviewed and refined by the two authors until both agreed. We employed different graphical representations alongside textual explanations to illustrate the data extraction outcomes. This approach aids in strengthening the connection between the extracted data and the drawn conclusions.

To mitigate selection bias, we adhered closely to the SLR guidelines in our study, ensuring the inclusion of only those studies that align with our defined scope. Any discrepancies were resolved through internal discussions. Additionally, we employed snowballing techniques to maximise the identification of pertinent papers, thereby minimising the potential for selection bias.

7. Conclusion

We presented an SLR of 38 selected primary studies related to addressing the issue of accessibility of low-code platforms. We created a taxonomy of different methods researchers have used to make their low-code approaches more accessible for users with special needs. Our analysis of 38 included primary studies allows us to conclude that there is very limited research on the accessibility of low-code approaches for users with special needs. The current research mainly focuses on UML or block-based programming, and they are limited to reading the diagrams rather than their modification. Future research should focus on making low-code platforms accessible for users with special needs, including visually impaired or blind users, users with cognitive impairments and so on.

Acknowledgment. Grundy is supported by ARC Laureate Fellowship FL190100035.

Declarations

Conflict of Interest. The authors declare that they have no conflict of interest.

Data Availability Statement. The data is provided online Khalajzadeh and Grundy (2024).

References

Adrian, B., Hinrichsen, S., Nikolenko, A., 2020. App development via low-code programming as part of modern industrial engineering education, in: Advances in Human Factors and Systems Interaction: Proceedings of the AHFE 2020 Virtual Conference on Human Factors and Systems Interaction, July 16-20, 2020, USA, Springer. pp. 45–51.

Albusays, K., Ludi, S., 2016. Eliciting programming challenges faced by developers with visual impairments: exploratory study, in: Proceedings of the 9th International Workshop on Cooperative and Human Aspects of Software Engineering, pp. 82–85.

Bock, A.C., Frank, U., 2021. Low-code platform. Business & Information Systems Engineering 63, 733–740.

Bottoni, P., De Marsico, M., Di Tommaso, P., Levialdi, S., Ventriglia, D., 2004. Definition of visual processes in a language for expressing transitions. Journal of Visual Languages & Computing 15, 211–242.

Brown, A., Stevens, R., Pettifer, S., 2004. Issues in the non-visual presentation of graph based diagrams, in: Proceedings. Eighth International Conference on Information Visualisation, 2004. IV 2004., IEEE. pp. 671–676.

Caldwell, B., Cooper, M., Reid, L.G., Vanderheiden, G., Chisholm, W., Slatin, J., White, J., 2008. Web content accessibility guidelines (wcag) 2.0. WWW Consortium (W3C) 290, 1–34.

Chisholm, W., Vanderheiden, G., Jacobs, I., 1999. W3c web content accessibility guidelines. URL: http://www/w3/org/TR/WCAG10. Cited .

Costello, K., Rimhol, M., 2021. Gartner forecasts worldwide low-code development technologies market to grow 23% in 2021.

Craig, J., Cooper, M., Pappas, L., Schwerdtfeger, R., Seeman, L., 2009. Accessible rich internet applications (wai-aria) 1.0. W3C Working Draft .

El-Attar, M., 2023. Evaluating the accessibility of a pon-enabled misuse case notation by the red–green colorblind community. Software and Systems Modeling 22, 247–272.

Glaser, B.G., Strauss, A.L., 2017. The discovery of grounded theory: Strategies for qualitative research. Routledge.

Green, T.R.G., Petre, M., 1996. Usability analysis of visual programming environments: a 'cognitive dimensions' framework. Journal of Visual Languages & Computing 7, 131–174.

Grillo, F.D.N., de Mattos Fortes, R.P., Lucrédio, D., 2012. Towards collaboration between sighted and visually impaired developers in the context of model-driven engineering, in: Joint Proceedings of Co-located Events at the 8th European Conference on Modelling Foundations and Applications (ECMFA 2012), Lyngby, pp. 241–251.

Grundy, J., Khalajzadeh, H., McIntosh, J., Kanij, T., Mueller, I., 2020. Humanise: Approaches to achieve more human-centric software engineering, in: International Conference on Evaluation of Novel Approaches to Software Engineering, Springer. pp. 444–468.

Hale, C., 2022. Low-code could replace "traditional" coding within months. techradar .

Kavcic, A., 2005. Software accessibility: Recommendations and guidelines, in: EUROCON 2005-The International Conference on" Computer as a Tool", IEEE. pp. 1024–1027.

Keele, S., 2007. Guidelines for performing systematic literature reviews in software engineering. Technical report, Ver. 2.3 EBSE Technical Report. EBSE .

Khalajzadeh, H., Grundy, J., 2024. Accessibility of Low-Code Approaches: a Systematic Literature Review. URL: https://doi.org/10.5281/zenodo.12751028, doi:10.5281/zenodo.12751028.

Khalajzadeh, H., Simmons, A.J., Abdelrazek, M., Grundy, J., Hosking, J., He, Q., 2020. An end-to-end model-based approach to support big data analytics development. Journal of Computer Languages 58, 100964.

Kirchhof, J.C., Jansen, N., Rumpe, B., Wortmann, A., 2023. Navigating the low-code landscape: A comparison of development platforms., in: LowCode workshop at the ACM/IEEE 26th International Conference on Model-Driven Engineering Languages and Systems.

Kitchenham, B., Madeyski, L., Budgen, D., 2022. Segress: Software engineering guidelines for reporting secondary studies. IEEE Transactions on Software Engineering 49, 1273–1298.

Luque, L., Brandão, L.O., Tori, R., Brandão, A.A., 2014. Are you seeing this? what is available and how can we include blind students in virtual uml learning activities, in: Proc. XXV Brazilian Conference of Informatics in Education, pp. 204–213.

Maplesden, D., Tempero, E., Hosking, J., Grundy, J.C., 2015. Performance Analysis for Object-Oriented Software: A Systematic Mapping. IEEE Transactions on Software Engineering 41, 691–710. doi:10.1109/TSE.2015.2396514.

Moody, D., 2009. The "physics" of notations: toward a scientific basis for constructing visual notations in software engineering. IEEE Transactions on software engineering 35, 756–779.

Mountapmbeme, A., Ludi, S., 2021. How teachers of the visually impaired compensate with the absence of accessible block-based languages, in: Proceedings of the 23rd International ACM SIGACCESS Conference on Computers and Accessibility, pp. 1–10.

Mountapmbeme, A., Okafor, O., Ludi, S., 2022. Addressing accessibility barriers in programming for people with visual impairments: A literature review. ACM Transactions on Accessible Computing (TACCESS) 15, 1–26.

Saleh, F., El-Attar, M., 2015. A scientific evaluation of the misuse case diagrams visual syntax. Information and Software Technology 66, 73–96.

Sarioğlu, A., Metin, H., Bork, D., 2023. How inclusive is conceptual modeling? a systematic review of literature and tools for disability-aware conceptual modeling, in: Proceedings of the 42nd International Conference on Conceptual Modeling (ER 2023), Springer.

Seifermann, S., Groenda, H., 2016. Survey on textual notations for the unified modeling language, in: 2016 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD), IEEE. pp. 28–39.

Shahin, M., Liang, P., Babar, M.A., 2014. A systematic review of software architecture visualization techniques. Journal of Systems and Software 94, 161–185. doi:10.1016/J.JSS.2014.03.071.

Torres, M.J.R., Barwaldt, R., 2019. Approaches for diagrams accessibility for blind people: a systematic review, in: 2019 IEEE Frontiers in Education Conference (FIE), IEEE. pp. 1–7.

Vincent, P., Iijima, K., Driver, M., Wong, J., Natis, Y., 2019. Magic quadrant for enterprise low-code application platforms. Gartner report .

Watson, C., Cooper, N., Palacio, D.N., Moran, K., Poshyvanyk, D., 2022. A systematic literature review on the use of deep learning in software engineering research. ACM Transactions on Software Engineering and Methodology (TOSEM) 31, 1–58.

Wiley, 2022. How wiley promotes your research | wiley. URL: https://authorservices.wiley.com/author-resources/Journal-Authors/Promotion/wiley-promotion.html.

Wohlin, C., 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. ACM International Conference Proceeding Series doi:10.1145/2601248.2601268.

World Health Organization, 2023a. Blindness and vision impairment. URL: https://www.who.int/news-room/fact-sheets/detail/blindness-and-visual-impairment#:~:text=Globally%2C%20at%20least%202.2%20billion,near%20or%20distance%20vision%20impairment.

World Health Organization, 2023b. Disability. URL: https://www.who.int/news-room/fact-sheets/detail/disability-and-health#:~:text=An%20estimated%201.3%20billion%20people%20%E2%80%93%20or%2016%25%20of%20the%20global,experience%20a%20significant%20disability%20today.

Zhuang, W., Gan, X., Wen, Y., Zhang, S., 2022. Easyfl: A low-code federated learning platform for dummies. IEEE Internet of Things Journal 9, 13740–13754.

Zubair, M.S., Brown, D.J., Hughes-Roberts, T., Bates, M., 2023. Designing accessible visual programming tools for children with autism spectrum condition. Universal Access in the Information Society 22, 277–296.

A. List of Primary Studies

Table 9: List of review articles

ID	Title	Author(s)	Venue	Year
S1	Self-Directed Creation and Editing of UML Class Diagrams on Mobile Devices for Visually Impaired People	Fabian Wildhaber, Nadim Salloum, Marcel Gygli, Andrea Kennel	IEEE Tenth International Model-Driven Requirements Engineering (MoDRE)	2020
S2	P-CUBE: Block Type Programming Tool for Visual Impairments	Shun Kakehashi, Tatsuo Motoyoshi, Ken'ichi Koyanagi, Toru Ohshima, Hiroshi Kawakami	Conference on Technologies and Applications of Artificial Intelligence	2013
S3	System Operation Improvement of P-CUBE2 for Visually Impaired People	Mariko Tsuda, Tatsuo Motoyoshi, Kei Sawai, Hiroyuki Masuta, Takumi Tamamoto, Ken'ichi Koyanagi, Toru Oshima	World Automation Congress (WAC)	2018
S4	Classification of UML Diagrams to Support Software Engineering Education	José Fernando Tavares, Yandre M. G. Costa, Thelma Elita Colanzi	International Conference on Automated Software Engineering Workshops (ASEW)	2021
S5	Improvement of P-CUBE: Algorithm education tool for visually impaired persons	Shun Kakehashi, Tatsuo Motoyoshi, Ken'ichi Koyanagi, Toru Oshima, Hiroyuki Masuta, Hiroshi Kawakami	IEEE Symposium on Robotic Intelligence in Informationally Structured Space (RiiSS)	2014
S6	Inclusion in computing and engineering education: Perceptions and learning in diagram-based e-learning classes with blind and sighted learners	Leandro Luque, Leônidas de Oliveira Brandão, Elisabeti Kira, Anarosa Alves, Franco Brandão	IEEE Frontiers in Education Conference (FIE)	2017
S7	An Auditory Interface to Workspace Awareness Elements Accessible for the Blind in Diagrams,Äô Collaborative Modeling	Márcio Josué Ramos Torres, Regina Barwaldt, Paulo Cesar Ramos Pinho, Luiz Oscar Homann de Topin, Tiago Fossati Otero	IEEE Frontiers in Education Conference (FIE)	2020
S8	Blocks4All: Overcoming accessibility barriers to blocks programming for children with visual impairments	Lauren R. Milne, Richard E. Ladner	Conference on Human Factors in Computing Systems (CHI)	2018
S9	Accessible modeling on the web: A case study	Filipe Del Nero Grillo, Renata Pontin de Mattos Fortes	Procedia Computer Science	2014
S10	Tabgo: Towards accessible computer science in secondary school	Ken H. Andriamahery-Ranjalahy, Léa Berquez, Nadine Jessel, Philippe Truillet	Springer Nature Switzerland	2021
S11	Accessibility of block-based introductory programming languages and a tangible programming tool prototype	Emmanuel Utreras, Enrico Pontelli	International Conference on Computers Helping People with Special Needs (ICCHP)	2020
S12	Towards collaboration on accessible UML models	Stephan Seifermann, Henning Groenda	Mensch und Computer	2015
S13	UML modeling for visually-impaired persons	Brad Doherty, Betty HC Cheng	International Workshop on Human Factors in Modeling / Modeling of Human Factors (HuFaMo) at MODELS	2015
S14	Exploring accessible programming with educators and visually impaired children	Ana Cristina Pires, Filipa Rocha, Antonio José de Barros Neto, Hugo Simão, Hugo Nicolau, Tiago Guerreiro	Interaction Design and Children Conference (IDC)	2020
S15	UML4ALL syntax – A textual notation for UML diagrams	Claudia Loitsch, Karin Muller, Stephan Seifermann, Jörg Henß, Sebastian Krach, Gerhard Jaworek, Rainer Stiefelhagen	Computers Helping People with Special Needs (ICCHP)	2018
S16	Tests with blind programmers using AWMo: An accessible web modeling tool	Filipe Del Nero Grillo, Renata Pontin de Mattos Fortes	Universal Access in Human-Computer Interaction. Design and Development Methods for Universal Access (UAHCI)	2014
S17	Can We Work Together? On the Inclusion of Blind People in UML Model-Based Tasks	L. Luque, E. S. Veriscimo, G. C. Pereira, L. V. L. Filgueiras	Inclusive Designing	2014
S18	StoryBlocks: A tangible programming game to create accessible audio stories	Varsha Koushik , Darren Guinness, Shaun K. Kane	Human Factors in Computing Systems Proceedings (CHI)	2019
S19	An approach for synchronizing UML models and narrative text in literate modeling	Gunnar Schulze, Joanna Chimiak-Opoka, Jim Arlow	International Conference on Model Driven Engineering Languages and Systems (MODELS)	2012
S20	Accessible block-based programming for k-12 students who are blind or low vision	Meenakshi Das, Daniela Marghitu, Mahender Mandala, Ayanna Howard	Universal Access in Human-Computer Interaction. Access to Media, Learning and Assistive Environments (UAHC)	2021

Continued on next page

Table 9 – continued from previous page

ID	Title	Author(s)	Venue	Article
S21	MUzECS: Embedded blocks for exploring computer science	Matthew Bajzek, Heather Bort, Omokolade Hunpatin, Luke Mivshek, Tyler Much, Casey O'Hare, Dennis Brylow	IEEE Blocks and Beyond Workshop	2015
S22	Towards a modelling workbench with flexible interaction models for model editors operating through voice and gestures	Jo~ao Fonseca de Carvalho, Vasco Amaral	Annual Computers, Software, and Applications Conference (COMPSAC)	2021
S23	Viable haptic UML for blind people	Claudia Loitsch, Gerhard Weber	International Conference on Computers Helping People with Special Needs (ICCHP)	2012
S24	Using voice commands for uml modelling support on interactive whiteboards: Insights and experiences	Rodi Jolak, Boban Vesin, Michel R.V. Chaudron	Conferencia Iberoamericana en Software Engineering (CIbSE)	2017
S25	Evaluating the accessibility of a PoN-enabled misuse case notation by the red–green colorblind community	Mohamed El-Attar	Software and Systems Modeling	2022
S26	Investigating the Impact of Computing vs Pedagogy Experience in Novices Creation of Computing-Infused Curricula	Amy Isvik, Veronica Cateté, Tiffany Barnes	Innovation and Technology in Computer Science Education (ITiCSE)	2021
S27	Flowgen: Flowchart-based documentation for C++ codes	David A. Kosower, J.J. Lopez-Villarejo	16th Computer Physics Communications	2015
S28	Analysing the cognitive effectiveness of the WebML visual notation	David Granada, Juan Manuel Vara, Marco Brambilla, Verónica Bollati, Esperanza Marcos	Software and Systems Modeling	2017
S29	Presenting UML Software Engineering Diagrams to Blind People	Alasdair King, Paul Blenkhorn, David Crombie, Sijo Dijkstra, Gareth Evans, John Wood	International Conference on Computers for Handicapped Persons	2004
S30	On the Inclusion of Blind People in UML e-Learning Activities	Leandro Luque, Leônidas de Oliveira Brandão, Romero Tori, Escola Politécnica, Anarosa Alves Franco Brandão	Brazilian Journal of Computers in Education	2015
S31	Towards collaboration between sighted and visually impaired developers in the context of model-driven engineering	Filipe Del Nero Grillo, Renata Pontin de Mattos Fortes, Daniel Lucr´edio	Graphical Modeling Language Development (GMLD) workshop at ECMFA	2012
S32	Mode12gether: a tool to support cooperative modeling involving blind people	Leandro Luque, Christoffer L. F. Santos, Davi O. Cruz, Leonidas O. Brandao, Anarosa A. F. Brandao	Brazilian Conference of Software	2016
S33	Cross-modal collaborative interaction between visually-impaired and sighted users in the workplace	Oussama Metatla, Nick Bryan-Kinns, Tony Stockman, Fiore Martin	International Conference on Auditory Display	2012
S34	ModelByVoice-towards a general purpose model editor for blind people.	Joao Lopes, Joao Cambeiro, Vasco Amaral	International Workshop on Human Factors in Modeling / Modeling of Human Factors (HuFaMo) at MODELS	2018
S35	Drawing and Understanding Diagrams: An Accessible Approach Dedicated to Blind People	Frederic Serin and Katerine Romeo	International Conference on Human-Computer Interaction	2022
S36	Accessible Blockly: An Accessible Block-Based Programming Library for People with Visual Impairments	Aboubakar Mountapmbeme, Obianuju Okafor, Stephanie Ludi	ACM SIGACCESS Conference on Computers and Accessibility (ASSETS)	2022
S37	Design considerations to increase block-based language accessibility for blind programmers via blockly	Stephanie Ludi, Mary Spencer	Journal of Visual Language and Sentient Systems	2017
S38	Adding speech recognition support to uml tools	Samuel Lahtinen, Jari Peltonen	Journal of Visual Languages and Computing	2004