Universidade Estadual de Feira de Santana

Programa de Pós-Graduação em Ciência da Computação

Towards Accessible ICT Learning: A Study on Assistive Tools for Visually Impaired Students

Lucas Lopes Fraga

Feira de Santana

2024

Universidade Estadual de Feira de Santana

Programa de Pós-Graduação em Ciência da Computação

Lucas Lopes Fraga

Towards Accessible ICT Learning: A Study on Assistive Tools for Visually Impaired Students

Dissertação apresentada à Universidade Estadual de Feira de Santana como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Orientadora: Larissa Rocha Soares Bastos

Feira de Santana

2024

Ficha Catalográfica - Biblioteca Central Julieta Carteado - UEFS

F87t

Fraga, Lucas Lopes

Towards accessible ICT Learning: a study on assistive tools for visually impaired students / Lucas Lopes Fraga – 2024.

121 f.: il.

Orientadora: Larissa Rocha Soares Bastos

Dissertação (mestrado) – Universidade Estadual de Feira de Santana, Programa de Pós-Graduação em Ciência da Computação, Feira de Santana, 2024.

1.Educação. 2.Tecnologia da informação. 3. Deficiência visual. 4. Tecnologia assistiva. 5. CraftPy. I. Bastos, Larissa Rocha Soares, orient. II. Universidade Estadual de Feira de Santana. III. Título.

CDU 376.32:004.9

Daniela Machado Sampaio Costa - Bibliotecária - CRB-5/2077

Lucas Lopes Fraga

Towards Accessible ICT Learning: A Study on Assistive Tools for Visually Impaired Students

Dissertação apresentada à Universidade Estadual de Feira de Santana como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Feira de Santana, 13 de agosto de 2024

BANCA EXAMINADORA

Documento assinado digitalmente
gov.br
LARISSA ROCHA SOARES BASTOS
Data: 15/08/2024 22:22:45-0300
Verifique em https://validar.iti.gov.br

Larissa Rocha Soares Bastos (Orientador(a))
Universidade do Estado da Bahia

Documento assinado digitalmente
gov.br
WINDSON VIANA DE CARVALHO
Data: 19/08/2024 10:03:29-0300
Verifique em https://validar.iti.gov.br

Windson Viana de Carvalho
Universidade Federal do Ceará

Documento assinado digitalmente
gov.br
RODRIGO SILVA DURAN
Data: 16/08/2024 10:35:11-0300
Verifique em https://validar.iti.gov.br

Rodrigo Silva Duran
Instituto Federal de Mato Grosso do Sul

Abstract

Despite significant advances in promoting equitable quality education for students with visual impairments (SVI), many barriers persist, particularly in highly visual fields such as Information and Communications Technology (ICT). To address these barriers, various tools have been developed to help SVI understand on-screen content. However, many SVI still face significant challenges in computing courses, indicating that current tools are insufficient. This highlights the need for studies to identify and analyze existing tools and determine why they fall short. The main objective of this work is to identify and analyze tools that aid SVI in university-level computing courses. To achieve this, we (i) performed a systematic literature review using primary data sources in the computing field to identify relevant studies and (ii) proposed and developed a tool to address one of the gaps identified during the review. We found 47 studies that met our inclusion and exclusion criteria and extracted information about their publishing details, area, availability, evaluation methods, Technology Readiness Level (TRL), and target audience. Our review revealed that approximately 80% of the studies lacked ways to access the tool, indicating a significant availability issue. Additionally, more than 23% of the tools had no evaluation process. These findings highlight gaps and deficiencies in tools designed for SVI in university-level computing courses. For example, many studies do not include VI participants in their evaluations or lack any evaluation altogether. To address these gaps, we developed CraftPy, an accessible tool compatible with screen readers for creating multiple types of diagrams. We conducted an experiment with eight participants with VI, and CraftPy proved to be accessible and effective, enabling participants to create diagrams using screen readers. By developing CraftPy, we aim to promote equity in higher education, offering SVI enhanced opportunities to succeed in ICT courses.

Keywords: Tools, Visual Impairment, Information and Communications Technology, Assistive Technology

Resumo

Apesar de avanços na equidade educacional para estudantes com deficiência visual (EDV), eles ainda enfrentam muitas barreiras em disciplinas excessivamente visuais, como Tecnologia da Informação e Comunicação (TIC). Para superar essas barreiras, foram desenvolvidas ferramentas para ajudar esses alunos a entender o conteúdo exposto no quadro e na tela. No entanto, essas ferramentas ainda não garantem uma educação de qualidade para EDV, indicando a necessidade de mais estudos para identificar e analisar as ferramentas existentes. O objetivo deste trabalho é identificar e analisar ferramentas que auxiliem EDV em disciplinas de informática no nível universitário. Para isso, realizamos uma revisão sistemática da literatura, utilizando as principais fontes de artigos em computação para identificar estudos relevantes. Além disso, desenvolvemos uma ferramenta para preencher uma das lacunas identificadas durante a revisão. Encontramos quarenta e sete estudos que atenderam aos critérios de inclusão e exclusão, extraindo informações sobre publicação, área, disponibilidade, métodos de avaliação, Nível de Prontidão Tecnológica (Technology Readiness Level) e público-alvo. Descobrimos que cerca de 80% dos artigos não fornecem meios para utilização das ferramentas, indicando falta de disponibilidade. Além disso, mais de 23% das ferramentas estudadas não foram avaliadas. Esses resultados ajudam a identificar lacunas nos estudos sobre ferramentas para EDV em cursos de computação universitários, como a ausência de participantes com deficiência visual nos processos de avaliação ou a falta de avaliação das ferramentas. Através das lacunas encontradas na revisão sistemática, nós criamos CraftPy, uma ferramenta acessível compatível com leitores de tela para a criação de múltiplos diagramas. Um experimento foi feito com oito participantes com deficiências visuais diversas utilizando a ferramenta, CraftPy demonstrou ser uma ferramenta acessível para o público alvo e possibilitou os participantes a criarem diagramas através do uso de leitores de tela. Através do desenvolvimento de CraftPy, pretendemos aumentar a equidade no ensino superior, auxiliando EDV nos cursos de computação.

Palavras-chave: Ferramenta, Deficiência Visual, Tecnologia da Informação e Comunicação, Tecnologia Assistiva

Preface

This master's thesis was submitted to the State University of Feira de Santana (UEFS) as a partial requirement to obtain the Master's degree in Computer Science.

The dissertation was developed in the Postgraduate Program in Computer Science (PGCC), with Prof. Dr. **Larissa Rocha Soares Bastos**.

This research was funded by CAPES.

Acknowledgments

Primeiramente, agradeço à minha família pelo suporte por todos esses anos. Agradeço aos meus pais pelos sacrifícios que fizeram para que eu pudesse chegar onde cheguei, mesmo diante de tantas adversidades, por estarem ao meu lado nos momentos em que mais precisei de apoio e também por fazerem o possível para que eu pudesse ter um futuro melhor. Dedico grande parte das minhas conquistas e esforços a ambos.

Também agradeço às minhas irmãs, Natalia e Alice, e ao meu primo Demerson por estarem ao meu lado tanto na infância quanto na vida adulta, trazendo leveza e felicidade aos meus dias.

Um grande agradecimento à minha orientadora Larissa Rocha por me dar todo o apoio necessário para a construção desta dissertação e por ser uma professora incrível, ajudando e me guiando desde os primeiros passos com a estruturação do trabalho até tudo o que veio a seguir. As minhas experiências no mestrado e a construção deste trabalho foram uma grande jornada na minha vida, que só foi possível devido à sua assistência. Sou muito grato pelo apoio tanto educacional quanto motivacional!

Agradeço bastante ao professor Rafael Tosta por me dar uma ajuda tremenda tanto na minha jornada acadêmica quanto na ferramenta construída por este trabalho, ajudando bastante a tornar a teoria em realidade e me dando conselhos indispensáveis para a realização deste trabalho. Um grande abraço!

Agradeço também aos professores do PGCC da UEFS em geral pelas aulas e pela didática incríveis. Aprendi muito durante as aulas do mestrado e os considero inspirações para a área da educação em computação.

"It is much more difficult to judge oneself than to judge others. If you succeed in judging yourself rightly, then you are indeed a man of true wisdom."

– Antoine de Saint-Exupéry

Contents

Alignment with the Research Line

Research Line: Software and Computer Systems

This dissertation is within the line of Software and Computer Systems because it seeks to contribute to Computing Education by analyzing the assistive technologies that help SVI learn the content present in universities promoting equity among students and creating an assistive technology to fill one of the gaps currently present in the field.

The purpose of this research is to identify the reason why assistive technologies in ICT courses are scarce, searching for possible gaps and weak points in the process of evaluation and creation of these tools and verifying their availability.

After identifying the gaps present in the current works about assistive technology in university-level ICT courses, we developed a software engineering tool called CraftPy. CraftPy is an accessible web tool fully compatible with screen readers, it enables SVI to create various types of diagrams using Python code, employing an object-oriented approach to design classes, actors, entities, attributes, and relationships.

Bibliographic Productions, Technical Productions and Awards

0.1 Author Production

Lucas Lopes Fraga, Rafael Tosta Santos, and Larissa Rocha. 2024. CRAFTPy: allowing people with visual impairments to create diagrams. In Anais do XXXVIII Simpósio Brasileiro de Engenharia de Software, setembro 30, 2024, Curitiba/PR, Brasil. SBC, Porto Alegre, Brasil, 727-733.

0.2 Author Co-Production

Alisson Souza Vilas Verde, Lais Farias Alves, Lucas Lopes Fraga, and Larissa Rocha. 2023. Py2UML: a Tool for Visually Impaired Students to Build UML Diagrams from Python Coding. In SBES'23 Tools Track. Association for Computing Machinery (ACM), New York, NY, USA.

List of Tables

List of Figures

List of Abbreviations

Abbreviation	Description
AT	Assistive Technology
ICT	Information and Communications Technology
TRL	Technology Readiness Level
SVI	Students with Visual Impairments
VI	Visual Impairment

Chapter 1

Introduction

According to the World Health Organization (WHO) in its latest world report focused on vision (WHO et al., 2019), it is estimated that globally more than 2.2 billion people have some visual impairment (VI) ranging from mild distance visual impairments to blindness, a number that tends to increase with population growth and lifestyle changes in the coming decades. As the number of people with VI increases, it is expected that the number of SVI entering higher education courses will also increase across the world.

As remarked in the United Nations Convention on the Rights of Persons with Disabilities, inclusive education is every student's right, not a privilege (Nations, 2023). Thus, the inclusion of students with visual impairments (SVI) has been a source of discussion in the literature, through research aimed at understanding the gaps and existing accessibility barriers and how these impact their lives (Miyauchi, 2020). To assist SVI in their daily activities, several tools known as assistive technologies (AT) have been and continue to be developed in various areas such as urban and indoor navigation (Kuriakose et al., 2022), object identification (Lin et al., 2019) and tactile maps (Ungar et al., 1993).

Despite the development of ATs that has helped people with VI in several areas, there are still fields where even with the presence of ATs to help them, there are still difficulties in breaking the visual barrier like in computing (Hadwen-Bennett et al., 2018). As computing is highly visual, many SVI often face barriers during their learning process due to the difficulty in interacting with digital interfaces even with the help of screen reading tools such as JAWS[1] and NVDA[2].

Even though many advances have been made to make computing education more inclusive for people with VI, there are gaps for advancement. Whether we want to guarantee the permanence of these students in higher education, simply providing

[1]https://www.freedomscientific.com/products/software/jaws/

[2]https://www.nvaccess.org

access to higher education may not be enough; this access must guarantee high-quality education. A way to achieve this is by employing tools to support the learning process.

In a survey applied to Brazilian SVI (Alves et al., 2022), more than 40% of respondents stated that there was a lack of instrumental accessibility in Brazilian universities. Instrumental accessibility is intended to bring improvements in the adaptation of materials, devices, utensils, and assistive technologies (Sassaki, 2003). Hence, the respondents stated that there was little or no accessibility to materials, devices, utensils, and technologies during their studies. They also highlighted the lack of accessibility in operating systems, development tools, and Web applications, especially for those who used a screen reader tool.

Although Alves et al. (2022) results indicate that there is still a need for ATs to support SVI in higher-level ICT courses, there are several tools presented in the literature that propose this specific support for different ICT fields. Thus, to better understand this scenario, it is crucial to investigate the proposed tools that aim to support the teaching and learning process in computing higher education courses for SVI.

The aim of this study is twofold. First, we want to present a holistic overview of the tools that aid SVI in subjects present in university-level computing courses. Thus, we performed a systematic literature review (SLR) to investigate which tools have been proposed and for which areas of ICT alongside other relevant information to the study, such as their technology readiness levels and processes of evaluation. Since the tools are likely built for academic purposes, this study also investigated whether they are available online. Through the SLR, we intend to identify possibilities for future research. Second, based on the SLR results, we developed a tool named `CraftPy`, a tool freely available on the internet that currently supports three types of diagrams: Class Diagrams, Use Case Diagrams, and Entity-Relationship Diagrams.

In an evaluation involving eight people with VI who are either current students or graduates of higher education ICT courses, most participants were able to complete assigned tasks with minimal difficulty using `CraftPy`. This tool aims to foster equity in higher education by empowering SVI to create their own diagrams, thereby promoting a more inclusive learning environment. By providing accessible tools like `CraftPy`, we can ensure that SVI has the necessary resources to succeed in visually demanding fields, ultimately contributing to a more inclusive and equitable educational landscape.

1.1 Motivation and Problem

Eyesight plays a critical role in the lives of human beings, which is why visual impairments have consequences for individuals, causing difficulties in many different situations in everyday life such as walking, reading, working, and even studying,

reducing their quality of life. Worldwide, billions of people have some type of visual impairment or are completely blind and projections from WHO show that this number will continue to grow due to changes in people's lifestyles and population growth (WHO et al., 2019).

To guarantee the right to education by law for this portion of the population that has some type of visual impairment, many public policies and institutional changes were adopted to include people with VI in higher education. With the increase in SVI enrolled in higher education courses, it is necessary to ensure that they have quality education inside and outside universities (Alves et al., 2022).

To promote the inclusion of this public, improve social equity, and enable a more independent life for people with VI, assistive technologies are developed. According to Bersch (2008), assistive technologies are the arsenal of tools that expand the functional abilities of people with a disability or enable the performance of a task that is impossible due to some disability, including visual impairments.

Due to the area of computing being highly visual and using many graphic artifacts such as diagrams and algorithms, many of these students enrolled in computing courses face barriers daily due to the lack of access to the content displayed on computer screens. To break the visual barrier in computing teaching, assistive technologies are used to provide quality learning to SVI (Konecki et al., 2016).

Through the recent study carried out by Alves et al. (2022), it is possible to identify that the biggest barrier to SVI in learning computing exists due to accessibility problems of the Instrumental type, that is, problems related to the lack of adaptation of materials, machines, utensils, and ATs.

This shows that there is still a knowledge gap in relation to the tools that are being developed for SVI in computing education, as even with the existence of several primary studies on assistive technologies that help in teaching programming, there is still a deficiency concerning their use to promote quality education for these students.

1.2 Goals

The general goal of this work is to identify and analyze the tools that facilitate teaching computing to SVI in higher education. These findings were used to build our own freely distributed online tool.

As specific objectives, this work aims to:

- Identify and analyze existing software tools published in the literature through a systematic literature review to verify the purpose and area of application of these tools, what types of evaluation were performed on these tools, their technology readiness levels, and their availability at the time of writing this work. The systematic review method was selected because it allows the creation of

a summary of existing technologies and thus identifies gaps in this field of research (Kitchenham et al., 2007).

- Develop a software tool for SVI to fill one of the gaps found in the systematic review.
- Evaluate the tool developed to check whether it meets its goals. Therefore, the aim is to make the tool available to a group of users with VI to conduct the evaluation.

1.3 Research Questions

The main research question of this research is: **What can be done to reduce the instrumental barriers that SVI deals with when studying ICT courses?**

This study seeks to answer the following research questions:

- **RQ1:** What are the gaps that currently exist regarding assistive technologies for SVI in ICT courses?
- **RQ2:** How can we enable SVI to create diagrams without the need to learn a new specific programming language for it?

To answer such questions, a systematic review was carried out with a final set of 47 articles that deal with assistive software technologies to assist the education of SVI in learning computing in higher education. A tool was developed to reduce the gap found in the systematic review: A big scarcity of working diagramming tools designed for SVI in ICT courses. For that reason, `CraftPy` was developed to aid them in the process of creating software engineering diagrams. Diagrams in `CraftPy` are created from the Python programming language, removing the need to learn a new specific programming language to use it. Also, the tool is available online, thus it is not necessary to install it.

1.4 Research Contribution

This work's contributions are separated into two parts: the contributions of the systematic review and the contributions of the developed tool.

The systematic review has the main focus on first identifying and showing the current landscape of assistive technologies in ICT courses for SVI by both analyzing the superficial and deep information about these works such as: how many works there are, their year of publication, which conferences they were published, which area of the ICT courses they belong to, their current availability at the time this work has been written, their technology readiness levels and how they were evaluated.

As a result, 9066 studies were found in six different databases. However, after applying the inclusion and exclusion criteria and doing the *snowballing* (Wohlin, 2014) of the studies, 47 primary works were selected to be analyzed.

From the studies, it was identified that 33 out of the tools developed for SVI in university-level computer courses focus on programming and diagramming while the other fourteen are spread in many other different areas such as computer networks, electronics, and robotics. This shows that most of the tools are created to help with either programming or diagramming while the other areas are much more scarce when it comes to tools.

This can bring light to other researchers about how works in this area have been done, showing more of their strengths and flaws present in the lack of uniformity found in the process of evaluation done in these works.

As for the `CraftPy` tool, it has a direct impact on SVI by actively aiding them in the process of creating multiple kinds of diagrams inside and outside the university. `CraftPy`'s main focus was accessibility towards SVI with an interface completely readable by screen readers and big components to allow those with low vision to read its contents properly.

`CraftPy` is an effort to reduce the barriers that SVI faces when studying in ICT courses and we hope that other researchers continue to create tools that promote equity in higher education.

1.5 Work Structure

The chapters of this work are structured as follows:

- **Chapter 2** presents the bibliographical review of the study, where the main concepts behind this research are explained thoroughly.
- **Chapter 3** explains the steps taken in this study, for the planning phase, systematic review, and the development and evaluation process of `CraftPy`.
- **Chapter 4** shows the methodology, discussion, and results of the systematic review divided by each one of the research questions and contains sections about the systematic review's related works and threats to validity.
- **Chapter 5** presents the process behind the development of the tool, the details about its operation, and the evaluation results.
- **Chapter 6** presents this dissertation's conclusions and possible future work.

Chapter 2

Study Background

2.1 Visual Impairment

Vision impairments are conditions that afflict billions of people globally. It occurs when an eye condition affects the visual system and its functions. It has serious consequences for individuals and affects their ability to see the world around them (WHO et al., 2019). The World Health Organization classifies vision impairment into two groups, distance and near vision impairment based on visual acuity, which is the ability to see details clearly, regardless of the object's distance.

Distance visual impairment is further classified into four groups by WHO based on the visual acuity in the better eye (WHO et al., 2019):

- Mild, where the visual acuity is worse than 6/12 to 6/18.
- Moderate, where the visual acuity is worse than 6/18 to 6/60.
- Severe, where the visual acuity is worse than 6/60 to 3/60.
- Blindness, where the visual acuity is worse than 3/60.

The definition for near vision impairment is where the near visual acuity is worse than N6 or M.08 at 40cm (WHO et al., 2019). In addition to these two types of visual impairment, there are more impairments such as colorblindness, which according to Salih et al. (2020), is an eye disorder that prevents the person from being able to distinguish tones of certain colors. The most common form of colorblindness affects the red and green colors, resulting in a missing or defective red or green photo-receptor cone.

Adults are severely affected by visual impairments having their quality of life reduced. Adults with visual impairments are generally less present in workforce participation and have lower productivity, they also tend to have higher rates of depression and anxiety compared to those with normal vision (WHO et al., 2019).

According to the World Health Organization (WHO et al., 2019), the leading causes of visual impairments are age-related macular degeneration, cataracts, diabetic retinopathy, glaucoma, and uncorrected refractive errors.

In computing-related courses, visual impairments pose a major barrier to students due to the accessibility issues present in both programming languages and programming environments (Mountapmbeme et al., 2022b). To break those barriers and allow students to have quality education in the computing field, there is an increase in the development of ATs.

In this work, the term visual impairments will be used to refer to all four groups of visual impairment, ranging from mild to moderate. As for blindness, it will strictly refer to the last classification where the individual visual acuity is worse than 3/60.

2.2 Assistive Technology

According to Bersch (2008), Assistive Technology is the term used to represent a range of resources (items, tools, objects, accessories) and services that help provide or expand functional abilities of people with disabilities, thus promoting a more independent life and assisting in the process of inclusion of people with disabilities in schools, universities, and companies.

ATs are not only limited to computers, in reality, ATs have been used for a long time in the fields of special education and rehabilitation with devices such as adaptive feeding instruments, wheelchairs, and vision aids (Edyburn, 2000).

The main objective of ATs is to ensure greater independence, quality of life, and social inclusion for people with VI through improvements in their communication skills, mobility, perception of different environments, reading, writing, learning, and work skills. They promote equity and can potentially improve their users' quality of life, increasing their autonomy and allowing them to interact with the environment around them (Bersch, 2008).

ATs can assist individuals with a wide range of disabilities. Examples of these ATs include walkers and wheelchairs for those with movement impairments, hearing aids that enhance the hearing capabilities of people with hearing loss, and white canes and screen readers that enable people with VI to navigate the world around them and use computers.

Currently, many other kinds of ATs are being created, such as mobile applications and desktop software that allow people with VI to "see" their surroundings through a phone and understand what is behind the screens of computers, terminals, and other machines (Hakobyan et al., 2013). Bersch (2008) classifies ATs into twelve categories:

1. Aids for daily life and practical life, such as custom-made utensils.

2. AAC - Augmentative and Alternative Communication, such as printed communication boards and recorded message vocalizers.
3. Computer accessibility features, such as modified keyboards and screen readers.
4. Environment control systems, such as smart home appliances.
5. Architectural projects for accessibility, such as ramps and elevators.
6. Orthoses and prostheses.
7. Postural Adequacy, such as orthostatic stabilizers.
8. Mobility aids such as canes, crutches, walkers, strollers, and manual or power wheelchairs.
9. Aids for expanding images and resources that translate visual content into audio or tactile information, such as magnifying glasses, screen magnifiers, and tactile maps and graphics.
10. Aids to improve auditory function and resources used to translate audio content into images, text, and sign language.
11. Mobility in vehicles, such as wheelchair lifts on buses.
12. Sports and Leisure, such as sound balls.

To break down barriers in the field of computer education concerning SVI, the most helpful AT resources are the ones in the following categories: Computer accessibility features and aids for expanding images and resources that translate visual content into audio or tactile information.

2.3 Assistive Technologies in Computing Education

Assistive technologies have been applied to most fields of education, including computing. Studies show that the number of assistive technologies in computing education is continually growing (Konecki et al., 2016) and is capable of improving the teaching and learning of computing for SVI (Adebayo and Ayorinde, 2022).

Computing education is an area that requires a lot of tools to make human-machine interaction easier for humans, but that is usually done in a way where most of the things that are returned to the user are visual such as charts, graphs, and diagrams, thus turning the computing field into a very visual field making it challenging for those with visual impairments (Luque et al., 2018).

The visual aspects of the computing field pose a big challenge for those who cannot see the contents on the screen and thus have difficulties using some of the technologies created to aid in human-machine interaction such as the mouse. Therefore, ATs were

created to allow SVI to perceive the contents of the screen in other alternative ways such as audio or tactile apparatuses.

The work done by Konecki et al. (2016) shows that all students benefit from assistive technology tools for the education of programming, especially those who are visually impaired pointing out that assistive technologies promote an improvement of the education process, better inclusion, and higher quality of gained knowledge.

As for the study done by Alves et al. (2022), the biggest problem faced by SVI in the computing field was instrumental problems, which according to Sassaki (2003) means that those problems derive from issues with the adaptation of materials, devices, utensils, and assistive technologies.

Therefore there is still a need for useful assistive technologies in the field of computing that goes beyond simply enabling access for those students (Alves et al., 2022), many other measures need to be taken such as asserting that those tools have quality and are easy to use, and the training of the teachers and students about the use of assistive technologies (Adebayo and Ayorinde, 2022).

2.4 Software Modeling and Diagrams

Software modeling is the creation of a physical view of a software system, an abstraction with a predetermined purpose such as the description of structural and behavioral aspects of a software (Guedes, 2018). Software modeling is often done through the use of diagrams showing a simplified overview of a system.

2.4.1 Class Diagrams

The Class Diagram is one of the most important and widely used diagrams in UML. Its primary purpose is to visualize the classes that will compose the system, along with their respective attributes and methods. It also demonstrates how the classes relate, complement, and communicate with each other. This diagram provides a static view of the class organization, focusing on defining their logical structure (Guedes, 2018).

Class Diagrams describe the types, properties, and operations of classes in a system and the relationships that exist between them (Fowler, 2018).

A class in a Class Diagram is a rectangle typically, but not necessarily, divided into three sections: The first one is the class name; the second is its attributes and data types; and the third and last section contains the methods of the class (Guedes, 2018). The class attributes and methods can also have other details, such as the visibility marker that indicates whether the attribute is private (-) or public (+) (Fowler, 2018).

To demonstrate the relationships between classes diagram classes represent them through associations. Associations are represented by lines that, in most of the

cases, connect two or more classes. They can have both navigability, represented by an arrow at one of the ends of the line, which determines in which direction the association goes; and multiplicity (cardinality), where it clarifies how many objects of a class are part of the association (Guedes, 2018).

The most common multiplicities are 1, where there is only one object of that class in that association; 0..1, where there may be one or zero objects of that class similar to boolean values; and an asterisk (*) when there is no upper limit to how many objects of that class in the association (Fowler, 2018).

There are special types of associations, such as the following (Guedes, 2018):

Aggregation

Aggregation is a special type of association that indicates that the information about an object is completed by the object at the other end making it a whole-part object relationship (Guedes, 2018). The association is represented by a white rhombus on the side of the object that represents the whole. An example of aggregation can be found in Figure 2.1.

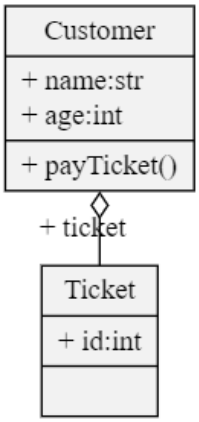

Figure 2.1: Aggregation between classes

Composition

Composition is a variation of an aggregation where the relationship between the two objects is stronger carrying a notion of ownership. According to (Fowler, 2018), composition carries a rule of "no-sharing", which means that the object that is part of the whole cannot be part of other objects. Another peculiarity is that if you delete the whole, you also delete all of its parts. The composition is represented by a black rhombus on the side of the object that represents the whole. An example of composition can be found in Figure 2.2.

Generalization/Specialization

Generalization and specialization allow the user to apply the concepts of inheritance in object-oriented programming. It is represented by a white arrow coming from

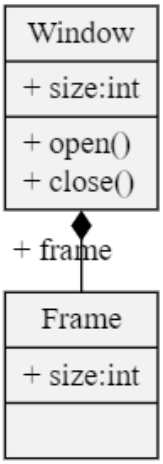

Figure 2.2: Composition between classes

the subtype and pointing at the supertype. An important aspect of generalization is substitutability, the ability to substitute the subtype class with the supertype class without stopping the code from working (Fowler, 2018). An example of specialization can be found in Figure 2.3.

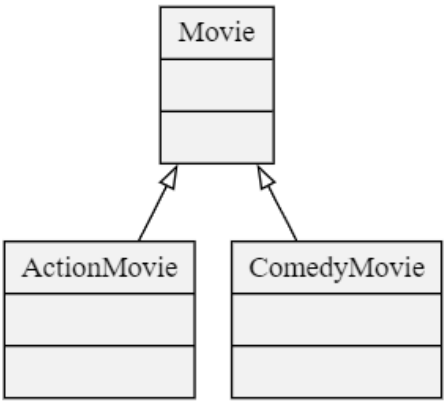

Figure 2.3: Generalization and Specialization between classes

Dependency

A dependency exists between two classes when changes regarding one of the elements change the other. Dependencies can exist for various reasons such as: Sending messages to another class, having that class as part of its data, or using that class as a parameter to an operation (Fowler, 2018). An example of dependency can be found in Figure 2.4.

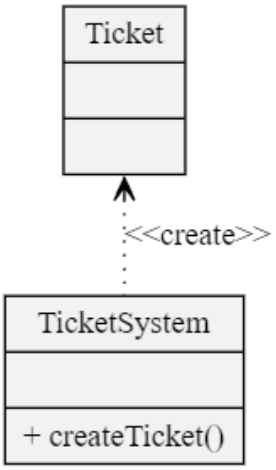

Figure 2.4: Dependency between classes

2.4.2 Use Case Diagrams

The use case diagram aims to present an overall external view of the system's functionalities for users, without focusing on implementation details. It is primarily used during the phases of system requirements elicitation and analysis, although it is consulted throughout the modeling process and can serve as a basis for various other diagrams. The goal is to use simple and easily understandable language so that users can grasp how the system will behave. The diagram identifies the actors (users, other systems, or special hardware) that will interact with the software, as well as the services or functionalities, known as use cases, that the system will provide to these actors (Guedes, 2018).

Actors and Use Cases

The two main items in use case diagrams are actors and use cases. Actors and use cases can connect to other actors and use cases through associations.

Actors represent roles that users or entities play within a system. They can execute one or more use cases and do not necessarily have to be human. A single individual can assume multiple actor roles within a system, and a single actor role can encompass multiple individuals (Fowler, 2018). An example of an actor can be found in Figure 2.5.

Figure 2.5: Example of an Actor

Use cases are services, tasks, or functionalities deemed necessary for the software, which actors can utilize when interacting with the system. Essentially, use cases define the functions of the system. They are typically named using a verb that describes the action to be performed, sometimes followed by the object affected by the action (Guedes, 2018). An example of a use case can be found in Figure 2.6.

Figure 2.6: Example of a Use Case

Inclusion

Inclusion is an association between use cases when a scenario, situation, or routine involves multiple use cases. This means that when a use case is executed, all the use cases it includes are also executed. Inclusions are represented by an association labeled ≪**include**≫ (Guedes, 2018). An example of inclusion can be found in Figure 2.7.

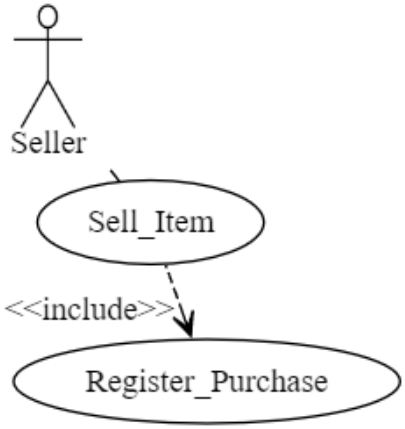

Figure 2.7: Example of an Inclusion of Use Cases

Extension

Extension is another type of association in use case diagrams. Extensions represent optional scenarios that are triggered only if certain conditions are met, as they are an optional step, they are not always executed when the use case that extends them is executed. Extensions are represented by an association labeled ≪**extend**≫ (Guedes, 2018). An example of an extension can be found in Figure 2.8.

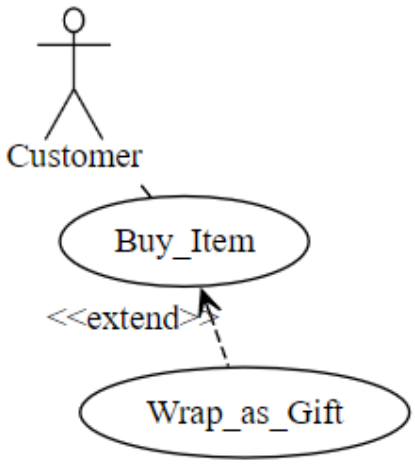

Figure 2.8: Example of an Extension of Use Cases

2.4.3 Entity-Relationship Diagrams

The entity-relationship diagram is one of the most widely used tools in conceptual modeling, particularly for entity-relationship modeling. It provides a high-level description of the database, independent of its implementation in a DBMS (Database Management System). The conceptual model outlines what data can appear in the database without specifying how this data is stored at the DBMS level (Heuser, 2009).

Entities, Attributes, and Relationships

The most fundamental element in an Entity-Relationship Diagram is an entity. Entities represent individual objects or concepts, which can be either tangible or abstract, such as people, cars, companies, and jobs. Entities have properties known as attributes that describe them. Attributes can be either simple or composite, depending on whether they contain other attributes within them (Takai et al., 2005).

Entities have key attributes, a unique identifier that distinguishes them from other entities of the same type. In the diagram, entities are represented as rectangles, and attributes are depicted as ellipses. Key attributes are underlined.

Relationships are a set of associations between entities, they can connect different entities or an entity to itself. These connections are represented as diamonds in the diagram, linked through lines to one or more entities (Heuser, 2009). Figure 2.9 illustrates an example of entities, attributes, and relationships. The example has two entities: ‘House’ and ‘Owner’. The ‘House’ entity has two attributes: a key attribute called ‘number’ and an attribute called ‘location’. The ‘Owner’ entity also has two attributes: a key attribute called ‘id’ and an attribute called ‘name’. The relationship between the two entities is labeled ‘Has’, indicating that a house has an owner, and an owner has a house. This example is shown in Figure 2.9.

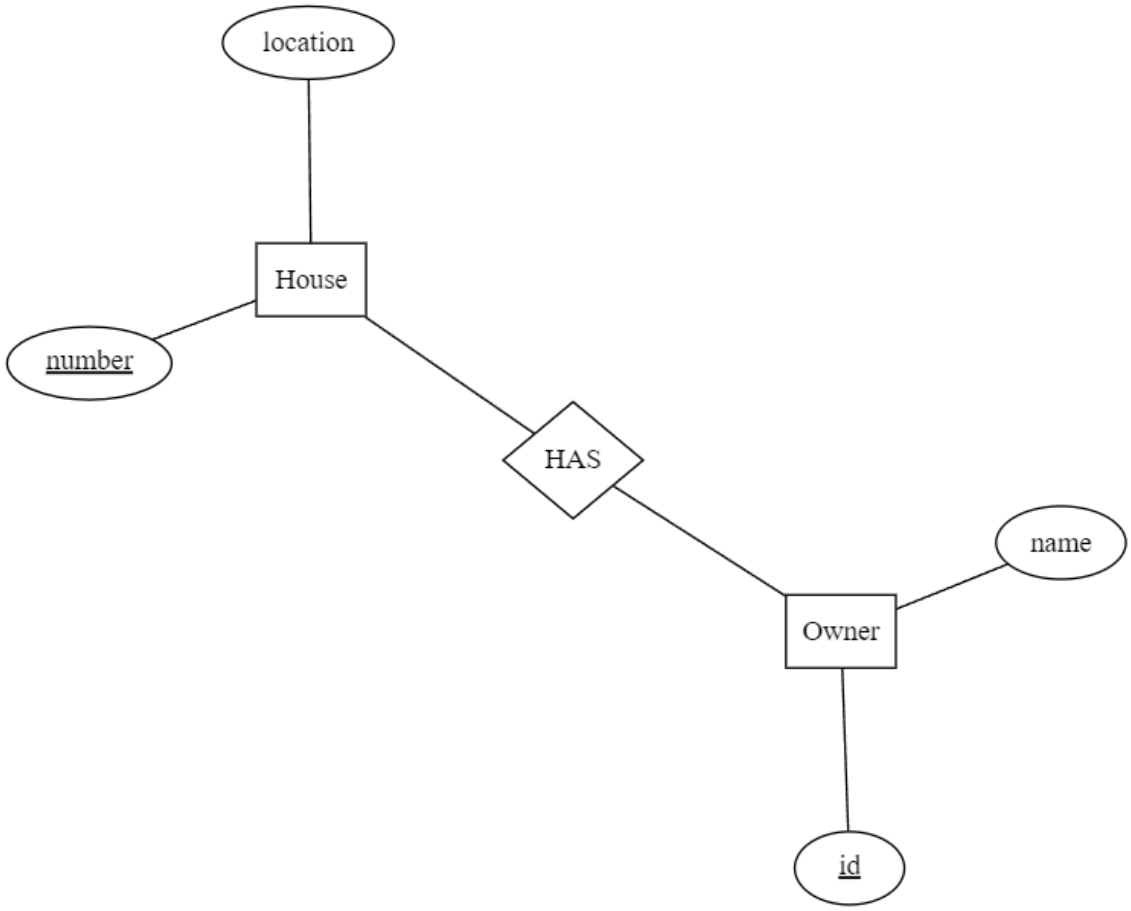

Figure 2.9: Example of Entities, Attributes, and Relationships

Weak Entities, Foreign Keys and Identifying Relationships

Weak entities are entities that lack a primary key and, therefore, cannot be uniquely identified on their own. To become distinguishable, a weak entity uses a foreign key in conjunction with its attributes. This foreign key is typically associated with an owner entity, which is linked to the weak entity through an identifying relationship. A weak entity has an existential dependence on the entity it is related to in an identifying relationship (Takai et al., 2005).

A weak entity is represented by a double rectangle and an identifying relationship is represented by a double rhombus. Figure 2.10 illustrates an example of weak entities and identifying relationships. The owner entity 'Department' has two attributes, a key called 'number' and an attribute called 'name', it has an identifying relationship called 'Create' towards the weak entity 'Project', meaning that departments can create projects and the project disappears if said department stops existing. 'Project' is a weak entity with two foreign keys 'name' and 'number' taken from its owner entity, 'Department'. 'Project' is considered a weak entity due to its lack of a primary key. This example can be found in Figure 2.10.

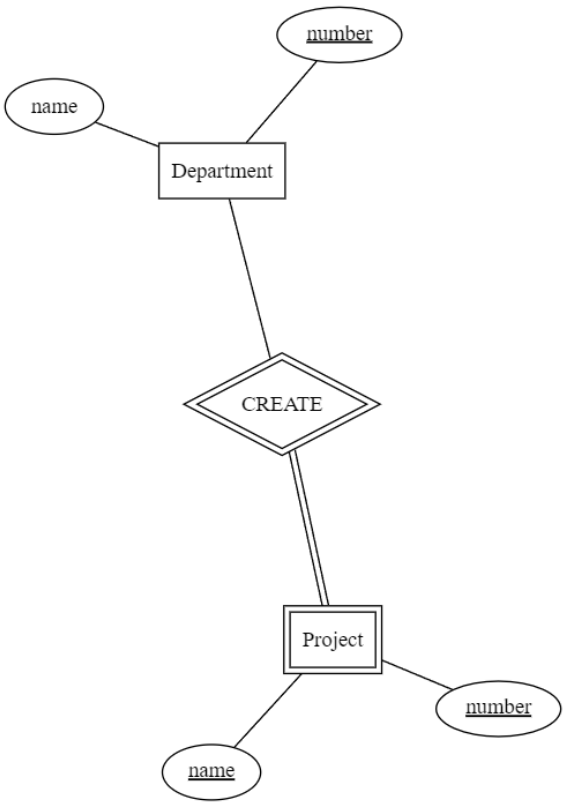

Figure 2.10: Example of Weak Entities and Identifying Relationships

Multiplicity and Multivalued Attributes

Relationships can carry different multiplicities (also known as cardinalities) to the entities involved, the multiplicity refers to the number of instances of one entity that are associated with other instances of another entity through a relationship. The most common types of multiplicity are 1:1 (One-to-one) where one single instance is associated with a single instance of another entity, 1:n (One-to-many) where a single instance is associated with many others, and n:m (Many-to-many) where many instances of an entity are associated with many instances of another entity (Heuser, 2009).

Multivalued Attributes are attributes that have a set of values. They are represented in the diagram by double ellipses. Figure 2.11 shows an example of it, 'Course' and 'Student' have a many-to-many relationship between them, which means a course can have multiple students, and students can have multiple courses. One of the course's attributes is a list of subjects represented by a multivalued attributes symbol in the diagram. This example can be found in Figure 2.11.

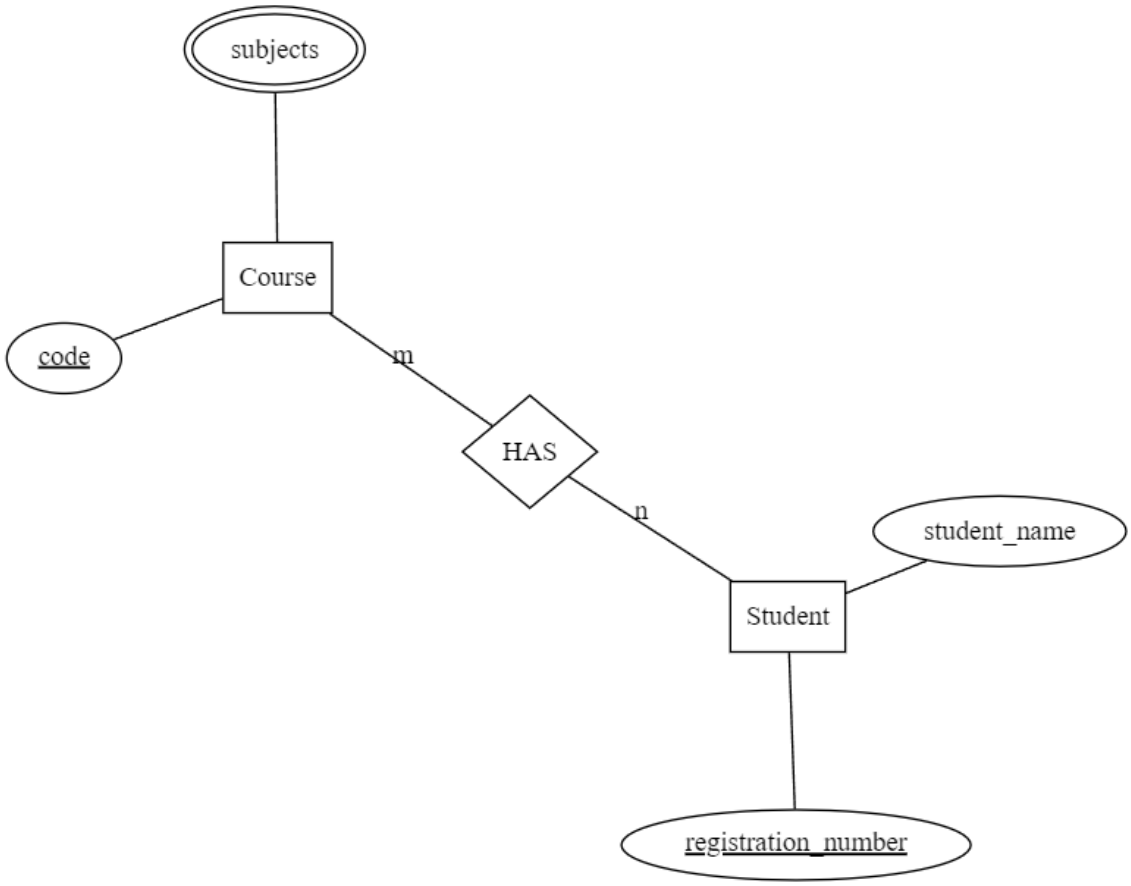

Figure 2.11: Example of Multiplicity and Multivalued Attributes

2.5 Chapter Summary

This chapter outlined the key conceptual foundations necessary for understanding this research, beginning with broader themes related to accessibility, such as visual impairments and assistive technologies, and then narrowing down to topics more directly related to the research. Understanding the current state of assistive technologies in computing education is essential for recognizing the need for such tools in this field.

There is also a brief explanation of the diagrams in software modeling that can be created by the `CraftPy` tool and the elements that can be implemented in those diagrams. The next chapter presents the methodology of this study.

Chapter 3

Methodology

This section presents the steps and classification of our literature review. In this way, the research is categorized based on the following aspects (Moresi et al., 2003):

- **Nature**: This research has a **basic** nature due to its objective of generating new knowledge useful for the advancement of science without the presence of a practical application aimed at a specific problem.
- **Approach**: The research approach is **quali-quantitative** due to the use of both qualitative and quantitative methods to analyze the results of the systematic review. The tool's evaluation involved an experiment that gathered information about the participants through two forms, collecting quantitative and qualitative data for the results.
- **Objectives**: The research has a **exploratory** content with the use of a systematic review to accumulate and systematize the primary works dispersed in the area, another characteristic that defines this research as exploratory is the absence of hypotheses and the presence of research questions.
- **Means of Investigation**: Regarding the means of investigation, two of them are used during this research. The first will be the systematic review, which is classified as **bibliographical research** because the systematic review is a systematized study made from material posted in digital libraries. The second means of investigation will be **field research** that was carried out through a preliminary experiment applied after the development of the tool.

The research is divided into two main parts: the systematic review and the tool development and evaluation. Figure 3.1 shows the stages of the methodology of this work.

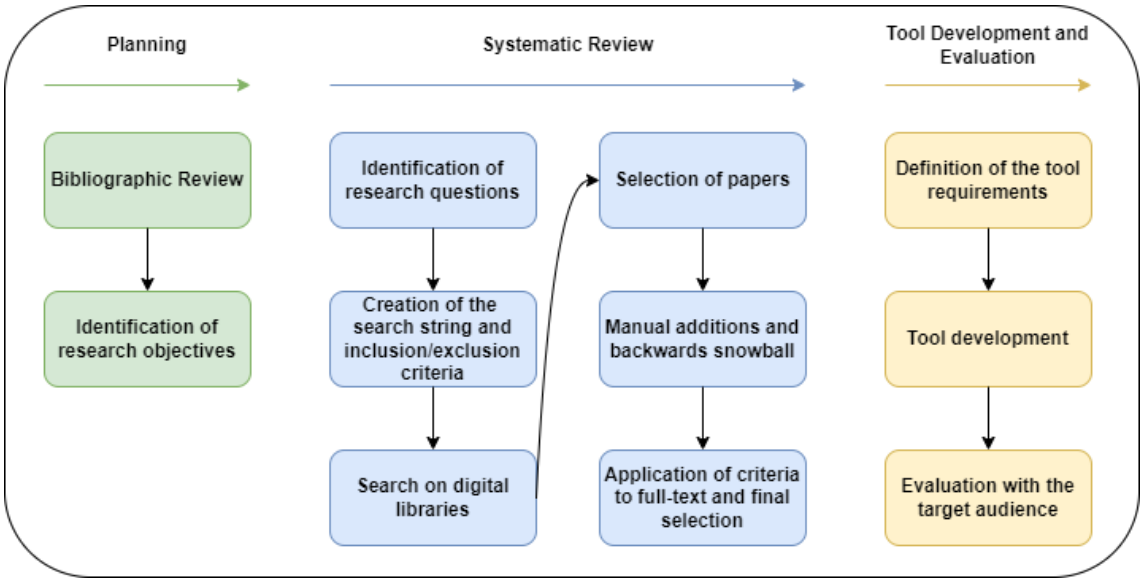

Figure 3.1: Steps of the methodology

- **Planning**: This phase of our study focuses on the planning phase, which includes searching for foundational material and identifying the research objectives. This phase consists of two main steps:
  1. **Bibliographic Review**: The planning phase began with a bibliographic review to understand the fundamental concepts used in this work, as detailed in Chapter 2. This was followed by an ad-hoc search for systematic reviews in digital libraries such as *IEEE Xplore* and *ACM Digital Library* on assistive technologies. Those related works are discussed in Section 4.4 of this dissertation.
  2. **Identification of research objectives**: The research objectives were identified by analyzing the results of the ad-hoc search. This analysis highlighted details not covered in existing works, such as the evaluation processes of the tools, their availability, their Technology Readiness Levels, and their target audiences.
- **Systematic Review**: In this phase, we conducted a systematic literature review using primary data sources in the ICT area. The systematic review included the following steps:
  1. **Identification of research questions**: This step involved formulating the research questions for our systematic review. We focused on addressing gaps not covered by previous systematic reviews in the same area, as identified during the planning phase.
  2. **Creation of the search string and inclusion/exclusion criteria**: The goal here was to define the search string and criteria for inclusion and exclusion to be used in database searches.

3. **Search on digital libraries**: In this step, we conducted searches for relevant papers across six databases: ACM Digital Library, IEEE Xplore, Scopus, Springer, Google Scholar, and SOL.

4. **Selection of papers**: Using the established inclusion and exclusion criteria, we screened the titles and abstracts of the retrieved papers to select relevant studies.

5. **Manual additions and backward snowball**: In this step, we employed backward snowballing techniques (Wohlin, 2014) on the selected papers to uncover additional relevant studies. We also manually added works from related systematic and bibliographic reviews, as discussed in Section 4.4.

6. **Application of criteria to full-text and final selection**: Finally, we applied the inclusion and exclusion criteria to the full texts of the remaining papers and made the final selection of studies to include in the systematic review.

- **Tool Development and Evaluation**: During this phase, our efforts went into developing and evaluating the `CraftPy` tool. The process involved the following steps:

  1. **Definition of the tool requirements**: We outlined the requirements for `CraftPy`, including its primary functionalities and limitations. Based on these requirements, we selected the types of diagrams most relevant for creation using the tool and determined how to translate Python code into diagrams.

  2. **Tool development**: This step involved the implementation of `CraftPy`, focusing on its user interface, screen reader accessibility, and logical structure in line with the defined requirements.

  3. **Evaluation with the target audience**: To ensure the tool met its requirements, we designed an experiment aimed at individuals with VI who are either studying or have completed a higher education ICT course. The evaluation process was divided into two parts: planning and creating three activities for participants to complete using the tool. We then sent the experiment via email to 45 potential participants, receiving 8 responses.

Chapter 4

Systematic Review

4.1 Systematic Review Methodology

Following the definition from Kitchenham et al. (2007), a systematic literature review (SLR) is a secondary study that aims to identify, evaluate, and interpret all available research relevant to a given subject to answer a specific research question, topic area, or phenomenon of interest.

The advantage of doing an SLR is that due to its well-defined structure, the results are less likely to be biased although this does not completely remove bias. A systematic review can also provide information about the effect of some phenomenon in a robust and transferable way if the studies provide consistent results. In quantitative studies, the systematic review makes it possible to combine data through meta-analytical techniques, making it possible to find effects that smaller studies cannot.

A systematic review differs from a conventional literature review due to the following factors:

- Systematic reviews contain a well-defined review protocol specifying the research question that is going to be addressed and the methods that will be used.
- Systematic reviews aim to identify as much literature as possible relevant to the research question.
- Systematic reviews are well documented regarding their search strategy so that readers can evaluate the rigor, completeness, and repeatability of the review.
- Systematic reviews contain well-defined inclusion and exclusion criteria.
- Systematic reviews define the information that will be obtained from the researched papers.
- Systematic reviews are also a prerequisite for quantitative meta-analysis.

One of the main reasons for doing a systematic review is to synthesize a large amount of work in a way that is considered fair and less biased, where the process behind the research is carefully elaborated because a systematic review must follow a predefined search strategy.

The goal of the systematic literature review performed is to identify and analyze existing software tools published in the literature to verify the purpose and area of these tools, what types of evaluation were carried out on them, their technology readiness levels, and their availability at the time of writing this work. This systematic literature review serves as an in-depth analysis of the current existing knowledge on the comprehension of the area.

4.1.1 Research Questions

Four questions were formulated to characterize the available evidence on tools for teaching computing to SVI in higher education, addressing different aspects of this area. The research questions (RQ) and corresponding rationale are described as follows:

[RQ.1:] **Which assistive technologies have already been created and are currently available to assist SVI in studying computing higher education courses?** This question focuses on listing the tools that have already been created and published in studies to meet the needs of SVI in higher education in computing fields.

[RQ.2:] **Which methods were used to evaluate the tool proposed in the paper?** To verify how these tools proposed in the study had their effectiveness validated through a solid evaluation, this question has the goal of discovering the methods used and the number of participants in the evaluation processes.

[RQ.3:] **What is the Technology Readiness Level (TRL) of the tools?** The focal point of this research question is finding the Technology Readiness Levels of the tools to discover how far in development they have gone.

[RQ.4:] **What was the target audience that the tools focused on?** To discover the target audience of the developed tools, this question focuses on finding who these tools were made for and what kind of environment they focused on (Academia or Industry).

4.1.2 Review Strategy

The research strategy and selection of papers from the primary studies included the search in the research bases for the papers, a preliminary application of the criteria to the title and abstract, manual additions, backward snowballing, application of

the exclusion criteria with *skimming* with the removal of duplicate works, and then the complete reading of the study as shown in Figure 4.1.

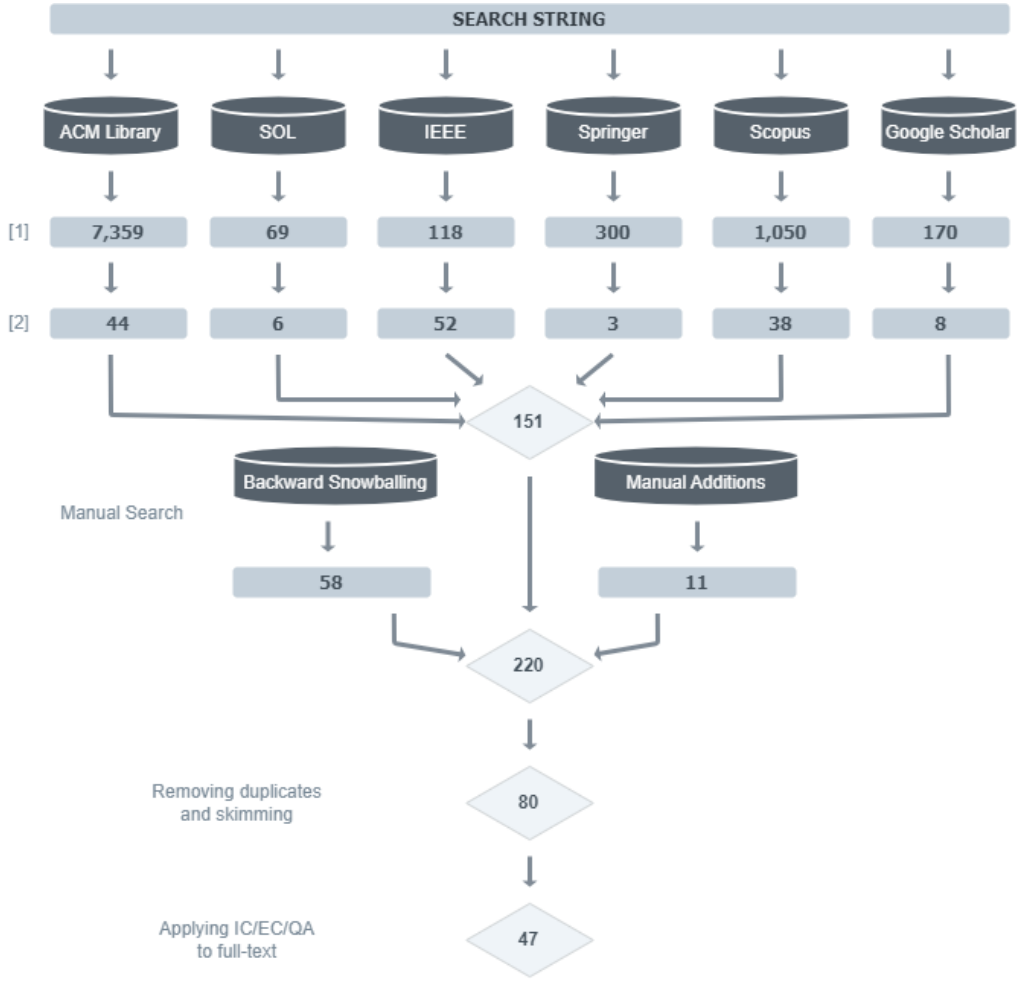

Figure 4.1: Steps of the systematic review
[1] = Papers, [2] = Applying inclusion criteria, and exclusion criteria to title and abstract.

The digital libraries used for the first search stage in the research databases are listed in Table 4.1. These research bases were used due to their large number of works present in both Portuguese and English. The objective was to find the largest number of works that could answer the research questions.

The search included all studies published up to the first quartile of 2023. Two search strings were created for the search in the databases, one in English and the other in Portuguese. The terms used in the search strings were terms that are widely used in works related to accessibility and assistive technologies. Due to its scope being more focused on students, the main keywords used to create the search string were **Visual Impairment** and **Assistive Technology**, from which other related terms and synonyms of those words were added to make it possible to find all relevant works.

Table 4.1: Digital libraries

Library	Link
ACM Digital Library	https://dl.acm.org/
IEEE Xplore	https://ieeexplore.ieee.org/
Scopus	https://www.scopus.com/
Springer	https://www.springer.com/
Google Scholar	https://scholar.google.com/
SOL	https://sol.sbc.org.br/

The term **Student** was added to the search strings to improve filtering due to the focus of the research question being on higher education students.

Search string in English:

[Abstract:(Visual Impairments) **OR** Abstract:(Visually Impaired) **OR** Abstract:(VI) **OR** Abstract:(Blind) **OR** Abstract:(Blindness) **OR** Abstract:(Low Vision) **OR** Abstract:(Nearsighted) **OR** Abstract:(Poor Eyesight)]

AND

[Abstract:(Assistive Technologies) **OR** Abstract:(Assistive Technology) **OR** Abstract:(Assistive Tool) **OR** Abstract:(Tool)]

AND

[Abstract:(Student)]

Search string in Portuguese:

[Abstract:(Deficiência Visual) **OR** Abstract:(Deficiente Visual) **OR** Abstract:(DV) **OR** Abstract:(Cego) **OR** Abstract:(Cegueira) **OR** Abstract:(Baixa Visão)]

AND

[Abstract:(Tecnologias Assistivas) **OR** Abstract:(Tecnologia Assistiva) **OR** Abstract:(Ferramenta Assistiva) **OR** Abstract:(Ferramenta)]

AND

[Abstract:(Aluno) **OR** Abstract:(Estudante)]

After applying the search string in the databases present in Table 4.1, 9,066 works were found. Most of the papers were found in the ACM Digital Library, as Table 4.2 shows. However, it was only possible to check 2,000 of the 7,178 works present in the ACM Digital Library due to a restriction by the library itself where it stops the results past the two-thousandth. Regarding Google Scholar, the works were filtered

by relevance and only 170 of them were investigated due to the massive number of results due to the results past that point being repetitive or irrelevant to the research questions. Many of the results found in the databases were not relevant to the topic. To better filter the amount of work, inclusion and exclusion criteria were created to be applied when reading the title and abstract of these primary studies.

Table 4.2: Distribution of primary studies by database

Library	Quantity
ACM Digital Library	7,178 in English and 181 in Portuguese
IEEE Xplore	118 in English
Scopus	1,050 in English
Springer	300 in English
Google Scholar	170 in English
SBC OpenLib (SOL)	69 in Portuguese

The defined inclusion and exclusion criteria were:

Inclusion criteria

- IC.1. Studies with any aspect of software tools for teaching computing to SVI in higher education as the main focus.
- IC.2. Papers written mainly in English or Portuguese.
- IC.3. Research papers.

Exclusion criteria

- EC.1. Studies outside the scope of this research.
- EC.2. Studies that do not provide enough information to answer the research questions in this systematic review.
- EC.3. Duplicate studies. When a study has been published in more than one conference, workshop, or journal, the most complete version will be used, namely, the one that explains it in more detail.
- EC.4. Grey literature (e.g., books and technical reports).
- EC.5. Papers that focus on fully tactile solutions instead of software solutions.
- EC.6. Papers with less than five pages.

After preliminary filtering using the inclusion and exclusion criteria by reading the titles and abstracts of the primary studies found, 151 studies were selected, as Figure 4.1 shows. Then, the *snowballing* (Wohlin, 2014) technique was applied in the 151 selected studies. The *snowballing* was done in two iterations, we checked the

references of the 151 primary studies (1st iteration) and repeated the process once more with the articles found in the first iteration (2nd iteration). The *snowballing* process brought over 58 relevant primary studies. Parallel to this process, it was carried out a manual analysis of the systematic reviews and mapping studies that these papers contained in the related work section, named *Manual Additions* in Figure 4.1. By reading these secondary studies found in the 151 selected papers, 11 more primary studies were added. Hence, at this stage, the total of selected papers was updated to 220.

In the next step, duplicate studies that were found in the previous steps were removed. After that, the reading strategy *skimming* (Masson, 1983) was applied to read the entire text to apply the exclusion criteria not only based on the title, *abstract* and number of pages. After these two processes, 80 studies remained for a full reading.

The final stage of the process corresponded to the complete reading and analysis of the papers and the extraction of the information related to the proposed research questions. During this step, some papers were still removed, mainly the ones: (i) focused on tactile tools; (ii) explained the same tool as another study, usually written by the same authors, but was outdated compared to the other study; and (iii) the tool presented was not focused on the computing area. At the end of the process, **47 papers** were selected to be included in this research. The papers are presented in Table 4.3.

The selection of the papers in the databases was performed by one researcher. In contrast, the analysis and reading of the papers, including the extraction of the relevant information, was performed by 3 researchers, one student, and two experienced professors.

4.2 Systematic Review Results

We selected 47 conference and journal papers (Table 4.3). Figure 4.2 shows the 25 conferences where the papers were published. The most common conferences were the ACM SIGACCESS Conference on Computers and Accessibility (ASSETS), the Special Interest Group on Computer Science Education (SIGCSE), the Conference on Human Factors in Computing Systems (CHI), and the International Conference on Computers Helping People with Special Needs (ICCHP).

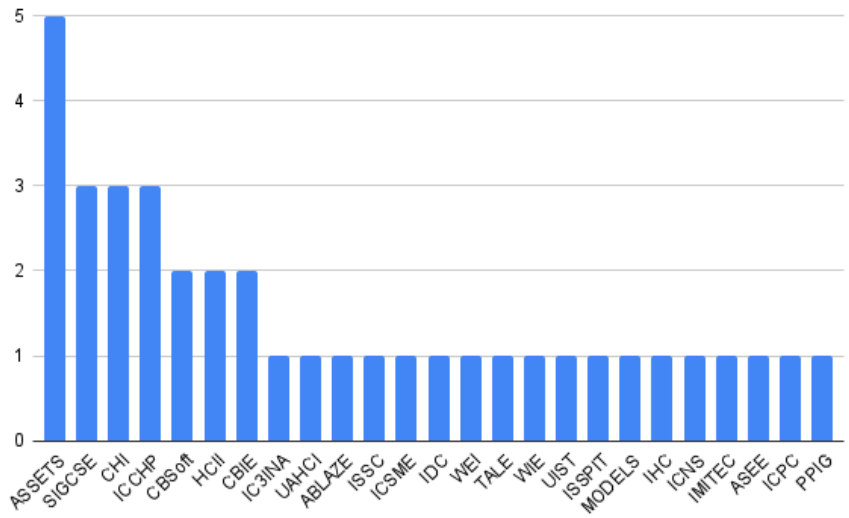

Figure 4.2: Conferences where the papers were published

Regarding journals, we found one paper for each of the following journals: Journal of Computers, IEEE Latin America Transactions, Information Systems Education Journal, Journal of Network and Computer Applications, Journal of Computing Sciences in Colleges, New Review of Hypermedia and Multimedia, *Revista Eletrônica De Iniciação Científica Em Computação* (Portuguese) and IEEE Access.

Figure 4.3 shows the year of publication of the 47 papers based on how many works were published in that year. Through the image, we can see that from 2014 onward, there has been an increase in papers regarding ATs for SVI, 30 of the selected papers were published in the year 2014 or later while only seventeen of them were released before 2014.

Table 4.3: Primary studies of the systematic literature review

ID	Title	Reference
S01	A framework for automatic text-to-flowchart conversion: A novel teaching aid for novice programmers	Hooshyar et al. (2014)
S02	A Java programming tool for students with visual disabilities	Smith et al. (2000)
S03	A JBrick: Accessible Robotics Programming for Visually Impaired Users	Ludi and Jordan (2015)

Continues in the next page

ID	Title	Reference
S04	A novel E-learning framework to learn IT skills for visual impaired	Kishore and Raghunath (2015)
S05	Accessible AST-Based Programming for Visually-Impaired Programmers	Schanzer et al. (2019)
S06	Accessible Blockly: An Accessible Block-Based Programming Library for People with Visual Impairments	Mountapmbeme et al. (2022a)
S07	Accessible Circuit Diagrams	Pender and Healy (2022)
S08	Ambient Intelligence System Enabling People With Blindness to Develop Electrotechnical Components and Their Drivers	Hudec and Smutny (2022)
S09	An Initial Investigation into Non-visual Code Structure Overview Through Speech, Non-speech and Spearcons	Hutchinson and Metatla (2018)
S10	APL: Audio Programming Language for Blind Learners	Sánchez and Aguayo (2006)
S11	AprenDER: Ferramenta de apoio à construção de diagrama entidade relacionamento para deficientes visuais	Magalhães and Neto (2010)
S12	Audiograf: a diagram-reader for the blind	Kennel (1996)
S13	AudioHighlight: Code Skimming for Blind Programmers	Armaly et al. (2018)
S14	Automated interpretation and accessible presentation of technical diagrams for blind people	Horstmann* et al. (2004)
S15	B-Model Uma ferramenta para auxiliar estudantes com deficiência visual na modelagem de sistemas	de Azevedo et al. (2021)
S16	Bonk: accessible programming for accessible audio games	Kane et al. (2018)
S17	CodeTalk: Improving Programming Environment Accessibility for Visually Impaired Developers	Potluri et al. (2018)
S18	deficiencia.org: Relato Sobre o Emprego de Ferramentas Computacionais Enquanto Tecnologias Assistivas no Ensino/Aprendizagem Para Pessoas com Deficiência Visual	Reis and Silva (2022)
S19	Developing methodologies for the presentation of graphical educational material in a non-visual form for use by people with vision impairment	Mohammadi and Murray (2013)
S20	Drawing and Understanding Diagrams: An Accessible Approach Dedicated to Blind People	Serin and Romeo (2022)
S21	Ferramenta para Mediação do Processo de Desenvolvimento do Pensamento Algorítmico contemplando Preceitos de Acessibilidade	Santos et al. (2019)
S22	Grid-Coding: An Accessible, Efficient, and Structured Coding Paradigm for Blind and Low-Vision Programmers	Ehtesham-Ul-Haque et al. (2022)
S23	GUIDL as an Aiding Technology in Programming Education of Visually Impaired	Konecki (2014)
S24	How Can Java Be Made Blind-Friendly	Markus et al. (2008)
S25	Including blind people in computing through access to graphs	Balik et al. (2014)
S26	Learning electronics using image processing techniques for describing circuits to blind students	Zapirain et al. (2010)
S27	Lessons and tools from teaching a blind student	Connelly (2010)
S28	Making turing machines accessible to blind students	Crescenzi et al. (2012)
S29	Model2gether: a tool to support cooperative modeling involving blind people	Luque et al. (2016)
S30	ModelByVoice - towards a general purpose model editor for blind people	Lopes et al. (2018)
S31	NetAnimations Mobile App: Improvement of Accessibility and Usability to Computer Network Learning Animations	Svaigen and Martimiano (2018)
S32	Node-read: a visually accessible low-code software development extension	Anderson et al. (2022)
S33	Nonvisual tool for navigating hierarchical structures	Smith et al. (2003)
S34	On the design of an educational infrastructure for the blind and visually impaired in computer science	Stefik et al. (2011)
S35	Projeto D4ALL: acesso e manipulação de diagramas por pessoas com deficiência visual	Pansanato et al. (2012)
S36	Remote Laboratory Access for Students with Vision Impairment	Murray and Armstrong (2009)
S37	StructJumper: A Tool to Help Blind Programmers Navigate and Understand the Structure of Code	Baker et al. (2015)

Continues in the next page

ID	Title	Reference
S38	TalkSQL: A Tool for the Synthesis of SQL Queries from Verbal Specifications	Obaido et al. (2020)
S39	Teaching modern object-oriented programming to the blind: An instructor and student experience	Owen et al. (2014)
S40	Teaching the blind to program visually	Siegfried et al. (2004)
S41	UML Modeling for Visually-Impaired Persons	Doherty and Cheng (2015)
S42	UML4ALL Syntax – A Textual Notation for UML Diagrams	Loitsch et al. (2018)
S43	Usable and Accessible Robot Programming System for People Who Are Visually Impaired	Damasio Oliveira et al. (2020)
S44	Using speech and touch to enable blind people to access schematic diagrams	Blenkhorn and Evans (1998)
S45	WAD: A Feasibility study using the Wicked Audio Debugger	Stefik et al. (2007)
S46	Work in Progress Report: Nonvisual Visual Programming	Lewis (2014)
S47	Workspace Awareness Acessível: estratégias de sonificação para projetar interfaces colaborativas acessíveis aos cegos	Torres and Barwäldt (2021)
End of Table		

The results will be presented in the following subsections for each research question defined in Section 4.1.1.

4.2.1 Assistive Technologies Tools (RQ1)

The 47 papers were divided by their areas, i.e.: 20 tools were found for Programming, 14 for Diagrams, 3 for Computer Networks, 3 for Electronics, 2 for Robotics, and one tool for each of the following categories: Database, Information Technology, Game Development, Algorithm Thinking, Graphs, and Turing Machines. Table 4.4 shows the papers and their categories. They were divided into the aforementioned categories according to either how the study itself categorized the tool or which categories were most suitable for the tool. In the case of a tool not fitting a current category, a new one was created for it. Next, we explore each area identified in this study and then we present the availability of the tools found.

1. Programming

21 primary studies were found to help programming in general, such as tools that work on Integrated Development Environments (IDEs) to help the user navigate through the code and tools that help teach concepts of programming in general. We next present these tools divided into Java Programming, Block-Based Programming, Code Navigation, IDE Plugin, New Programming Language, and Others.

Java Programming (S02, S24). Smith et al. (2000) created a tool called **Javaspeak** (S02) to help SVI learn how to program in Java. The prototype of the tool mentioned in the primary work was designed to act as a specialized programming environment, which is similar to an IDE, to help those students learn the programming Java through aural feedback given by the tool. Also focused on Java programming, the

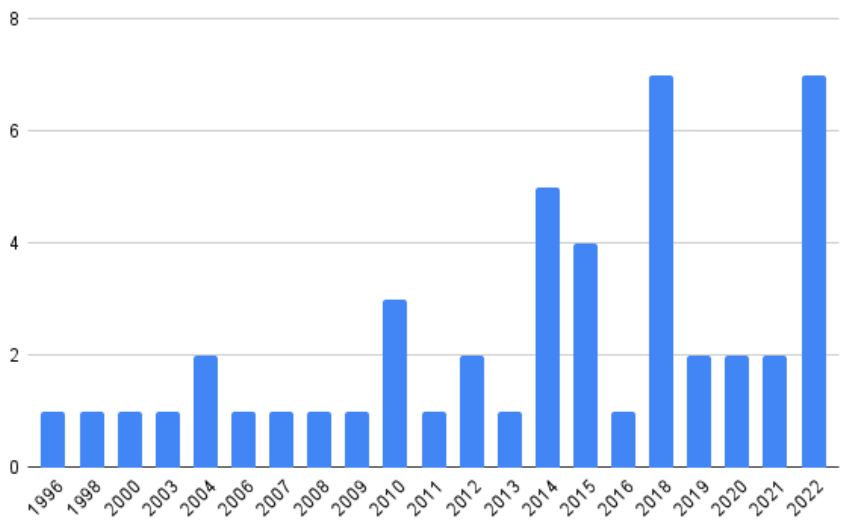

Figure 4.3: Years that the papers were published

MObile Slate Talker (MOST) (S24) tool was developed by Markus et al. (2008) and focuses on making Java more accessible for users who are blind. The MOST framework is a blind-friendly environment that requires no screen reader to function, it comes with an integrated text-to-speech, Braille entries, and easy navigation.

Block-Based Programming (S05, S06). Even though block-based programming was developed to help novice programmers understand how programming works with visuals and through the use of the mouse, those environments are very inaccessible to SVI. The following tools were proposed to aid them in using those kinds of environments.

The toolkit **CodeMirror-Blocks** (S05) is the focus of the work done by Schanzer et al. (2019), it has the purpose of creating an accessible block-based programming environment that uses spoken descriptions and navigation through keyboard shortcuts to allow VI programmers to navigate through the code more easily. It is a browser-based tool coded in JavaScript. The tool has functions such as syntax highlighting, bracket-matching, and auto-indenting.

Mountapmbeme et al. (2022a) presented a prototype of a programming library focused on block-based programming, the **Accessible Blockly** (S06). It focuses on making the Blocky library accessible through the use of screen readers and keyboard alone, thus allowing SVI to understand and code in block-based programming.

Code Navigation (S09, S13, S22). Ehtesham-Ul-Haque et al. (2022) developed a prototype tool that allows easier code navigation for blind users through the use of grids. **Grid-Coding** (S22) is a web-based system that shows Python code in a grid, turning each line of code in a row and indentation levels into columns of the grid.

Table 4.4: Distribution of primary studies by areas and subareas

Area	ID
Programming	
- *Java Programming*	S02, S24
- *Block-Based Programming*	S05, S06
- *Code Navigation*	S09, S13, S22
- *IDE Plugin*	S13, S17, S32, S33, S37
- *New Programming Language*	S10, S34
- *Others*	S18, S23, S27, S39, S40, S45, S46
Diagrams	
- *UML*	S14, S15, S20, S29, S35, S39, S41, S42
- *Others*	S01, S11, S12, S30, S44, S47
Computer Networks	S19, S31, S36
Electronics	S07, S08, S26
Robotics	S03, S43
Information Technology	S04
Game Development	S16
Algorithmic Thinking	S21
Graph (Discrete Mathematics)	S25
Turing Machines	S28
Database	S38

The **Auditory Code Overview and Navigation Tool (ACONT)** (S09), proposed by Hutchinson and Metatla (2018), focuses on helping VI users to navigate code and quickly understand its structure. The tool utilizes a timeline navigation technique that allows users to navigate through the code line-by-line at different speeds through the arrow keys. The code is read through both speech and non-speech elements representing the code itself and elements of the code, such as a semi-colon. Another tool that focuses on code navigation is **AudioHighlight** (S13), created by Armaly et al. (2018) to allow blind users to code skim through audio using a web service or as a plugin for the Eclipse IDE.

IDE Plugin (S13, S17, S32, S33, S37). Aside from the aforementioned **AudioHighlight** (S13) that fits in both categories, we have more tools that are plugins for IDEs such as **Codetalk** (S17), presented by Potluri et al. (2018), which is a plugin developed to aid VI programmers focused on the Visual Studio IDE. By the time the study was done, it supported two programming languages: C# and Python. Codetalk concentrates on extracting the relevant code information and sending it to the user by audio through keyboard shortcuts to reduce the barriers that VI programmers face when using screen readers to read the code line by line.

StructJumper (S37), a plugin for the Eclipse IDE created by Baker et al. (2015). It creates a hierarchical tree-based composition that uses the nesting structure of a

Java class to help blind programmers get a better grasp of how the code is nested. StructJumper helps users to better navigate the code and get more contextual information about the way the code is presented. It allows users to switch between the tree view and the textual editor quickly.

The **Node-Read** extension (S32), presented by Anderson et al. (2022), is a Node-RED extension for IDEs. It helps developers that need screen readers to code focusing on increasing the compatibility with screen readers such as JAWS and NVDA. Also developed as a plugin for the Eclipse IDE, the **Aural Tree Navigator** (S33) made by Smith et al. (2003) focuses on helping blind users understand the hierarchical structure of a program that is often represented through visual cues. The Aural Tree Navigator offers an accessible tree navigation strategy for non-sighted users.

New Programming Languages (S10, S34). Some of the tools developed to aid SVI in learning the concept of programming were new programming languages that use auditory elements to allow them to understand programming concepts. One of the programming languages developed was the **APL - Audio Programming Language** (S10), created by Sánchez and Aguayo (2006). APL was designed to allow blind programmers to construct programs in Java code through a code generator built into the language that converts the APL code into Java code and adds the necessary functions for it to work.

Stefik et al. (2011) created two tools: **Sodbeans**, an auditory programming environment, and **Hop** a programming language (S34). Sodbeans was created to work alongside screen readers in the process of reading the code for developers with VI. It includes a virtual machine, compiler, and debugger for the programming language they developed, Hop.

Others (S18, S23, S27, S39, S40, S45, S46). Reis and Silva (2022) developed a website in Portuguese called **deficiencia.org** (S18) to compile inside of a website content that is helpful for SVI in a single place to encourage academic formation in the computing areas for those students.

The **GUIDL (Graphical User Interface Description Language)** (S23) system proposed by Konecki (2014) focuses on aiding SVI by helping them to develop graphical user interfaces through the GUIDL language.

To help with programming with C++, Connelly (2010) created a header file called **pcspeak.h** (S27) that uses an open-source text-to-speech system to give audio feedback to the user about the prompts, inputs, and results of the code. In the work done by Owen et al. (2014), they create a drop-in component called **CSpeech** (S39) for code in C that allows to add speech to an application using a speech library for voice synthesis.

Siegfried et al. (2004) created a prototype compiler that makes the task of creating Visual Basic forms accessible for blind users. The prototype they create is called

molly.exe (S40) and has the function of making it possible to use Visual Basic without relying on its "point and click" design method.

Stefik et al. (2007) presented **WAD (Wicked Audio Debugger)** (S45), a debugger for Microsoft Visual Studio 2005 that gives audio feedback to the user about the computer code to assist in debugging the behavior in programs. As for the paper written by Lewis (2014) (S46), they mention their work in progress, more specifically a tool called **Noodle**. Noodle is designed to be a nonvisual dataflow programming tool that uses keyboard commands to create, execute, and modify dataflow programs without the need for visual presentation according to the primary work.

Only five out of the 20 programming tools (S02, S05, S06, S17, S22, and S34) are currently available for use, as shown in Table 4.5.

2. Diagrams

A total of 14 primary works were found for the creation, editing, and readability of diagrams. In the following, we discuss the tools found. First, we discuss about tools for UML diagrams (e.g., use case, class, and sequence diagrams). Second, we present the other kinds of diagramming.

Eight papers that present UML Diagrams (S14, S15, S20, S29, S35, S39, S41, S42) are presented as following.

TeDUB (Technical Drawings Understanding for the Blind) (S14) (Horstmann* et al., 2004) was made to make graphical information accessible through the analysis of graphical content with the use of text recognition and the identification of the diagram elements. It consists of two different modules, one that analyses the drawings and another that presents the results of said analysis. Results are then imported into formats that contain semantically enriched information.

The **B-Model** tool (S15) proposed by de Azevedo et al. (2021) was developed to help SVI model their own UML diagrams, a language called BLM (Blind Modeling Language) was also implemented to standardize the functional requirement specification for the generation of the diagrams. The process is done in three main steps: The functional requirement specification, the functional requirement interpretation, and the diagram generation. In the first step of the process, the student specifies the functional requirements in Portuguese and saves it in a text file. For the second step of the process, BLM is used to standardize the functional requirements written in the previous step to generate the diagram for the next step. In the third and last step, the diagram is generated based on the lexical processing and semantic conversion of the specified functional requirements.

Serin and Romeo (2022) developed a tool called **Latitude, Light and Accessible Tags Into Plain Text using Universal Design** (S20), to create UML class diagrams. The

developed tool that created HTML documents and SVG diagrams from plain text making those files completely readable by screen readers. The plain text files follow a format denominated the Latitude Format that holds semblance to PlantUML[1] (an open-source tool that allows users to create diagrams through coding) with a few alterations to make it more easily readable by screen readers. Those files are then compiled and turned into the final artifacts, the HTML documents, and SVG diagram images.

The study written by Luque et al. (2016) (S29) showcases the **Model2gether** tool, a web-based tool that focuses on cooperative modeling between sighted users and users who are blind to include SVI in the process of development of UML, data flow, and entity-relationship diagrams. The tool implements two interfaces, one for blind users with a screen reader compatible interface and one graphical interface for sighted users that shows the case models in their diagrammatic format. The aforementioned tool also allows the user to explore the diagrams with a hierarchical presentation starting from a high level in the hierarchy and then going deeper into the elements of the diagram to know more about their composition to reach the lower levels of the hierarchy.

Pansanato et al. (2012) created a prototype of a tool called **D4ALL** (S35) that can help blind users create use case UML diagrams through a conversion tool that turns UML diagrams in an XML format into the same diagram in a table format. This table contains information such as the actors, relationships, and use cases of the XML diagram allowing it to be more easily readable by screen readers. Besides that, the prototype also contains an accessible diagram editor that navigates through the entire diagram to allow easy access to certain elements for faster use of the tool.

In the work done by Owen et al. (2014) (S39), there are multiple tools, one of them, **Audible Browser**, can read XMI 2.1 files for class and state diagrams, where it plays different tones for each node referred to in the paper as 'stars' present inside of a 'constellation'. It uses arrow keys and the mouse for the navigation inside of the diagram and the higher the note is, the higher the node that represents is inside of the page.

The **PRISCA** tool (S41) was present in the primary work done by Doherty and Cheng (2015), the PRISCA tool can generate a haptic 3D representation of the UML diagrams from the output of an existing UML diagram editor also converting textual notations present in such diagrams into braille to enable blind users to understand the contents of the diagrams. The tool processes XML files to produce STL files containing a haptic representation of the UML diagrams, the STL extension is widely used for 3D printing and can be sent to other software, such as *MakerBot* mentioned in the paper, to print those haptic representations generated by PRISCA.

The last resource that was created for UML diagrams is the **UML4ALL** (S42) syntax present in the work done by Loitsch et al. (2018). The UML4ALL syntax is designed

[1]https://plantuml.com/

to help blind users who use screen readers to understand UML diagrams. The syntax can be read line by line by text-to-speech or braille and focuses on four main aspects: unique identification of diagram types, usage of meaningful keywords, clear and consistent structure within each diagram, and a prefix-like notation of elements. This tool aims to enable cooperation between blind and sighted users through a notation that will help blind users comprehend and create their own UML diagrams in a syntax format.

The following tools focus on other types of diagrams such as entity-relationship diagrams.

Other ways of diagramming (S01, S11, S12, S30, S44, S47)

An unnamed conceptual framework was created by Hooshyar et al. (2014) (S01) to help SVI create flowcharts through text. It concentrates on novice programmers without much computing knowledge, presenting only simple basic algorithmic programming problems as possibilities for input by the user.

AprenDER (S11), developed by Magalhães and Neto (2010) allows SVI to build entity-relationship diagrams, a graphical interpretation that is often utilized in database courses.

Kennel (1996) proposed the **Audiograf** tool (S12). It enables users who are blind or visually impaired to read diagrams present in technical reports in an audio-tactile way. Another tool that makes technical diagrams accessible to users who are blind or visually impaired is the **ModelByVoice** prototype (S30), proposed by Lopes et al. (2018), which aims to allow users with VI to model diagrams and systems from any language through their voice by using a speech recognition system present in the tool.

Kevin (S44), developed by Blenkhorn and Evans (1998), is a CASE tool that enables blind users to read, edit, and create diagrams through N^2 charts and use the keyboard to control the operations. Kevin allows blind users to read, create, and edit data flow diagrams used in software engineering.

Both **SONIv1** and **SONIv2** (S47), presented by Torres and Barwäldt (2021), are sonified interfaces that allow blind users to hear the modifications made by another user in collaborative resources allowing them to distinguish which action was performed, where it took place and made by whom allowing better cooperation during the use of tools that deal with diagrammatic information in collaborative resources.

Of the 14 tools focused on diagramming, only one (S42) of them is currently available to be used, as shown in Table 4.5.

3. Computer Network

A total of 3 primary works (S19, S31, and S36) were found about education in computer network systems.

Accessible Packet Tracer (S19), developed by Mohammadi and Murray (2013), is an accessible version of Packet Tracer that aims to improve accessibility in the graphical presentation of educational information of Cisco Networking Academy content with an accessible interface. Due to the mainstream tools used for computer network teaching being usually inaccessible to blind students,

Murray and Armstrong (2009) created **iNetSim** (S36), an accessible network simulator that makes simulation packages accessible for SVI. To help students struggling with computer network content Svaigen and Martimiano (2018) developed **NetAnimations** (S31), a computer network animation mobile app that contains learning animations to help with the content present in computer network courses. To increase the accessibility of the tool, they implemented functions to allow an easier understanding of the animations for blind users, such as the use of checkpoints and the implementation of descriptive audio.

Of the 3 tools focused on computer networks, only one (S31) of them is currently available to be used, as shown in Table 4.5.

4. Electronics

A total of 3 studies (S07, S08, and S26) were found about electric circuits, electrotechnical components, and electronics in general.

The first tool is a software called **NetlistToText** (S07) created by Pender and Healy (2022) to describe nodal connections. Those descriptions can replace visual representations of electrical circuits in academic material, thus allowing screen readers to read information that was previously inaccessible.

AMI System RUDO (S08), developed by Hudec and Smutny (2022), supports users with VI with the design and development of electrotechnical components allowing them to: connect electrical circuits; do the technical preparation and machining of mechanical components for device construction; program software drivers in a unified user interface adjusted for people who are blind; test and debug of drivers; measure electrical circuits of the developed hardware using a multimeter; and monitor the waveform of the electrical signal using an oscilloscope.

An unnamed open-source algorithm (S26) integrated into a tool compatible with OpenOffice was presented by Zapirain et al. (2010). It applies image processing and computer vision techniques to circuit schematics and creates an automatic textual description of both the sequence of the components and the position they are in the circuit schematics to make it accessible to users who are visually impaired or blind.

Of the 3 tools focused on electronics, none of them are available online but one of them (S08) has material related to it available with its link written in the article showing the use of the tool.

5. Robotics

A total of 2 tools (S03 and S43) were found about software that helps with learning robotics.

The **JBrick** tool (S03), proposed by Ludi and Jordan (2015), was implemented in Java to create a more accessible way for Lego Mindstorms programming. Through interfaces that can be used for those with or without sight, due to the compatibility with screen readers and refreshable braille displays, JBrick supports those with diverse visual acuity.

Two other tools created for robotics are **GoDonnie**, a programming language based on the Logo language, and **Donnie** (S43), a programming environment that runs GoDonnie, both created by Damasio Oliveira et al. (2020). They both have the purpose of teaching programming and the exercise of orientation and mobility to SVI through cognitive maps. Those maps are created as the user explores different scenarios through commands that are executed by the robot in a virtual environment.

Of the 2 tools focused on robotics, one (S43) of them is available.

6. Others

Only one primary work was found about the following categories: Database (S38), Information Technology (S04), Game Development (S16), Algorithm Thinking (S21), Graphs (S25), and Turing Machines (S28).

To help with the content of database courses, Obaido et al. (2020) created **TalkSQL** (S38), a voice-based query system that converts words into SQL queries and returns aural feedback to the user, allowing SVI to create SQL queries with more ease.

In the Information Technology area (S04), we found an unnamed E-learning environment created by Kishore and Raghunath (2015). This environment is a web system in Java that provides quality interaction between users with VI and e-learning platforms.

For the Game Development area, we found the **Bonk** tool (S16), created by Kane et al. (2018). Bonk is an accessible programming environment that allows blind and novice programmers with VI to develop interactive audio games using JavaScript.

As for Algorithm Thinking, the **CCEduc** tool (S21), created by Santos et al. (2019), is a software written in Java that helps in the process of developing the algorithm

thinking skill of the students. It takes some measures to improve its accessibility such as the implementation of Text-to-Speech options and the development of an accessible interface with useful keyboard shortcuts for blind users.

For the creation and drawing of graphs, the **Graph Sketching Tool - GSK)** (S25), developed by Balik et al. (2014), is a tool that allows users with VI to create and access graphs as node-link diagrams and share them with other sighted users in real-time.

About Turing Machines, **Accessible JFLAP** (S28), developed by Crescenzi et al. (2012), turns the well-known JFLAP Turing Machine simulator into a more accessible tool for blind students in computing courses.

Out of the other diverse tools, none of them is currently available. Figure 4.4 shows the distribution of tools for each area found by the systematic review.

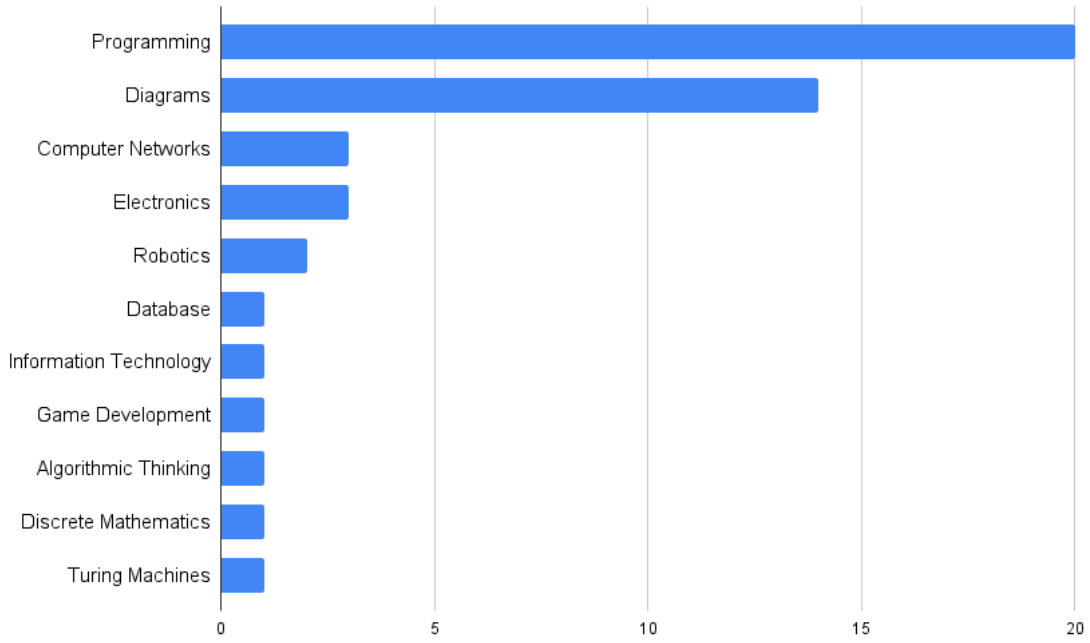

Figure 4.4: Distribution of the tools for each area

Availability

For a more general view of which of those tools were still available online at the time this research took place, Table 4.5 shows which tools were found either in or outside of the papers and which ones have links that no longer work or are not present in the paper and were not found online.

Out of the 47 papers, only seven of them had a working *link* that was related to the tool or tools present in the paper; Two of the tools did not have a *link* in the

article, but they could easily be found with a quick internet search for the name of the tool on the main search engines (Google and Bing); One of them had a material or snippets related to the tool in the paper; Six had *links*, but the *links* did not work, which could be a dead *link* or a *link* that redirected to the wrong page; The other 31 tools did not have any *link* at work and searches on search engines only directed to the article itself. In total, only nine out of forty-seven tools were available online when this research took place.

Table 4.5: Distribution of tools by availability

Availability	**ID**
Yes	
- *Mentioned in the paper*	S06, S17, S22, S31, S34, S42, S43
- *Not mentioned in the paper*	S02, S05
No	
- *Has material related to it*	S08
- *Link does not work*	S18, S25, S29, S39, S40, S44
- *Not mentioned in the paper*	S01, S03, S04, S07, S09, S10, S11, S12, S13, S14, S15, S16, S19, S20, S21, S23, S24, S26, S27, S28, S30, S32, S33, S35, S36, S37, S38, S41, S45, S46, S47

To showcase the availability of the primary works shown in Figure 4.4, we created Figure 4.5 where the available works for each area are shown by color.

RQ.1 Summary:
We have listed above all the relevant tools found in the primary works. Some fall into multiple categories due to having various tools within a single primary work. Based on our findings, 6 out of 20 programming tools, 1 out of 14 diagramming tools, 1 out of 3 computer network tools, and 1 out of 2 robotics tools are currently available. None of the tools in the other categories are now available. The category with the lowest percentage of available tools aside from the categories that had none is diagramming where only approximately seven percent of the total tools can be used.

4.2.2 Evaluation Methods (RQ2)

To answer this question, we verified in each primary work which was the research methods utilized to evaluate the tools present in the works, as shown in Figure 4.6. The classification used for the research methods comes from the work of Lazar et al. (2017) according to the book's chapters. The primary studies were classified into six categories according to the work's description of the evaluation process and our input on what best fits the evaluation process mentioned in the work.

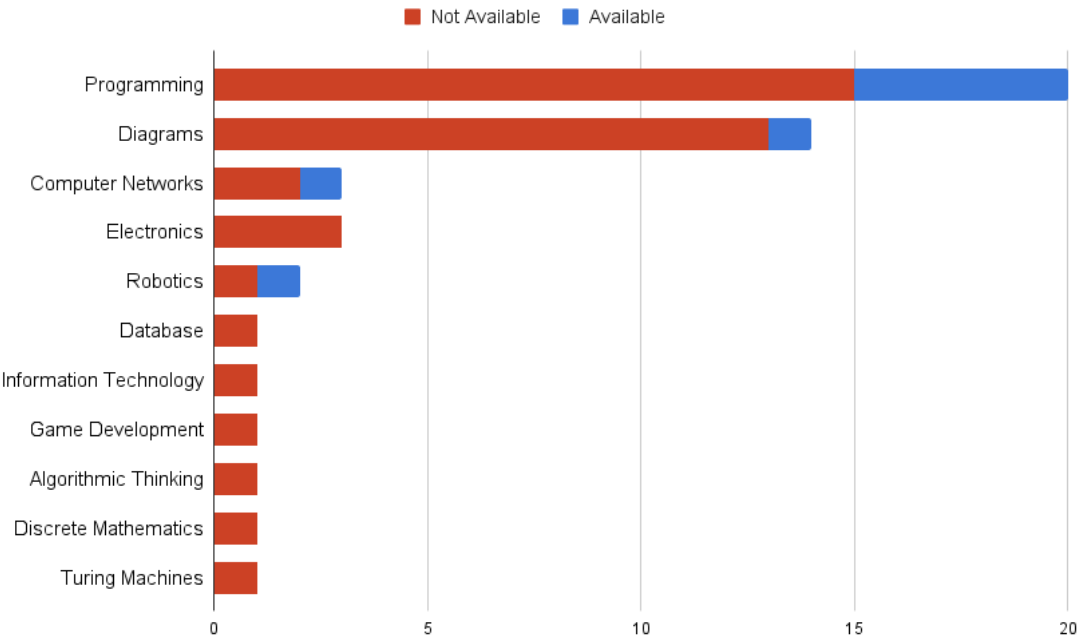

Figure 4.5: Availability of the tools

Table 4.6 complements Figure 4.6 showing which of the works were classified in which one of the categories.

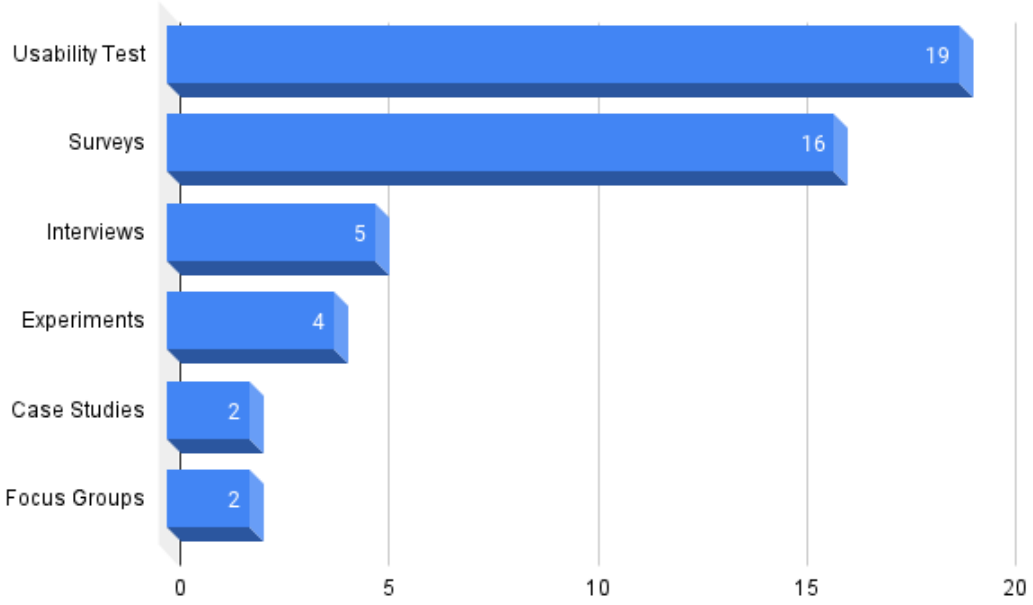

Figure 4.6: Research methods used in the primary works

Table 4.6: Research methods by primary work

Research Method	**ID**
Usability Test	S05, S08, S09, S10, S12, S14, S16, S22, S23, S25, S28, S29, S30, S32, S33, S37, S44, S45, S47
Survey	S03, S04, S07, S09, S14, S17, S23, S26, S30, S31, S32, S35, S38, S42, S43, S45
Interview	S03, S21, S33, S37, S43
Experiment	S05, S06, S13, S34
Case Study	S15, S29
Focus Group	S01, S16

Looking at Figure 4.6, it is shown that the most used methods to evaluate the tools were Usability Tests and Surveys. However, many of the primary works did not have documentation about any evaluation process during the development of the tool as we can see in Figure 4.7 that shows the number of evaluation methods that each one of the 47 primary works used in their evaluations. Through the graphic we can see that 12 of those works did not go through any evaluation, 22 of them used only one method of evaluation, and 13 of them used two or more of the methods of evaluation present in Figure 4.6.

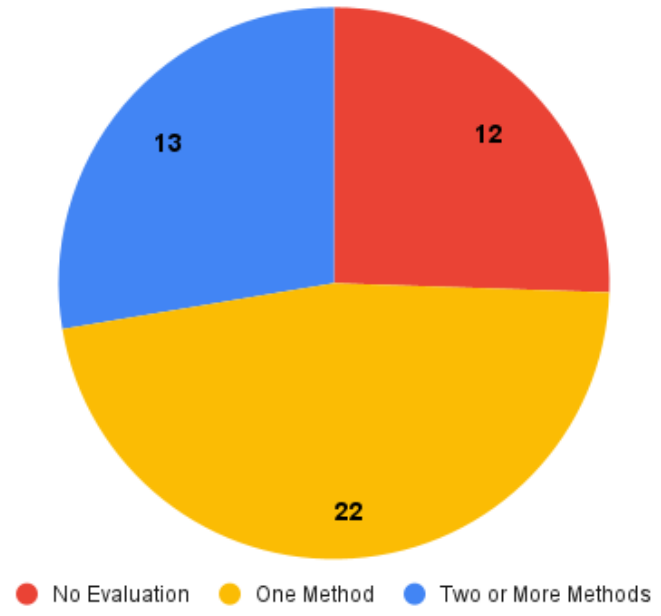

Figure 4.7: Quantity of methods used in the primary works

It was also observed that there is limited participation of individuals with VI in the studies. Figure 4.8 illustrates the number of participants who are visually impaired

or blind involved in the evaluation process of each primary study. Only works that specified that the participants were blind or visually impaired were counted for the graph. Participants who were sighted, simulated blindness with the use of blindfolds or experts in the area did not count for the graph in Figure 4.8.

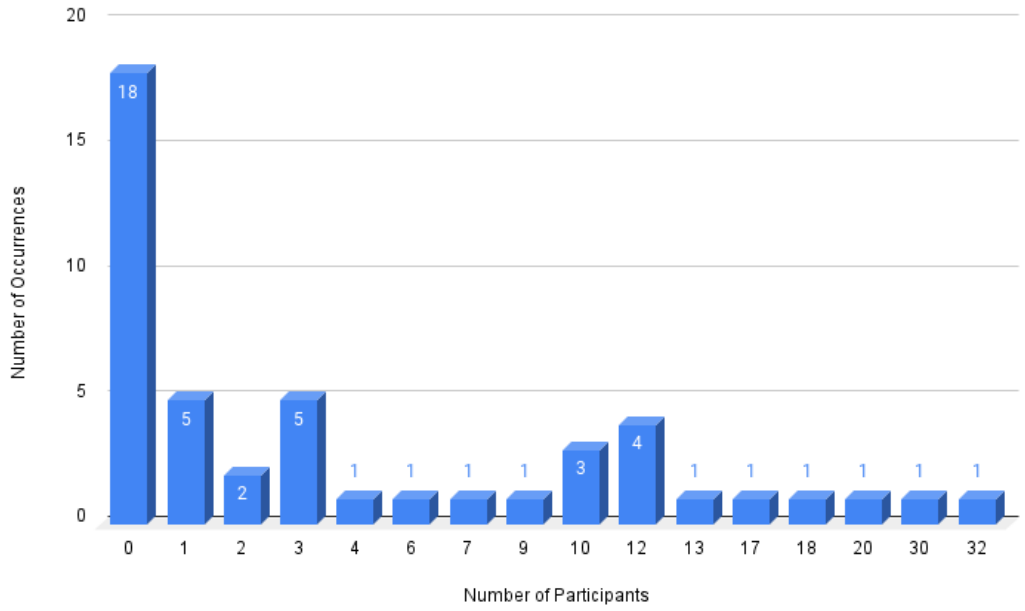

Figure 4.8: Quantity of visually impaired and blind participants that participated in the evaluation of the primary works

In total, we had 18 studies that had no involvement of SVI. Studies that had no evaluation (12 of them according to Figure 4.7) were counted as studies that had zero participants.

RQ.2 Summary:
The evaluation methods presented show that many works have informal evaluation processes or are not evaluated at all. There is a small amount of VI participants in these evaluations and in some cases, the tools are evaluated without SVI at all.

4.2.3 Technology Readiness Levels (RQ3)

For this question, we used the Technology Readiness Levels (TRL), a metric present in the work done by Mankins et al. (1995).

The TRL scale is a metric system that assesses the maturity level of a particular technology used on and off in NASA space technology planning for many years (Mankins et al., 1995). Introduced in the 1970s by NASA to be a consistent measure

of technology maturity, the TRL scale is used for technology assessment in multiple industries such as power systems and consumer electronics serving a broader goal than intended (Olechowski et al., 2015).

According to Bergamini (2020), the TRL metric is used in Brazil by the Brazilian Company of Research and Industrial Innovation (Embrapii) and the Brazilian Agricultural Research Corporation (Embrapa) to assess the maturity levels of technologies in projects.

The levels start with a certain technology in a very basic scientific form and progress until the technology has proven to work in an operating environment allowing the technology assessed on the TRL scale to easily represent its level of readiness towards eventual operation (Olechowski et al., 2015).

To assess the TRL of the tools described in the primary studies, we manually reviewed the works and applied a categorization method based on the software evaluation criteria used by the Brazilian Agricultural Research Corporation (Embrapa)[2] and seen in the work done by Bergamini (2020).

To ensure the works were categorized accurately, we used the questions present in the TRL calculator created by the Brazilian Ministry of Science, Technology, and Innovation (MCTI)[3] to classify the primary works in their corresponding TRL.

The questions we used from the calculator and the sheet containing how each of the primary works was ranked are available in Portuguese in Appendix B. For a quick overview, Table 4.7 shows how the primary works were classified in each TRL category.

Table 4.7: Distribution of primary studies by their Technology Readiness Level

TRL	**ID**
TRL 1	S27
TRL 2	S19, S36, S46
TRL 3	S02, S11, S18, S24, S39, S40, S41
TRL 4	S01, S04, S07, S09, S12, S15, S20, S21, S26, S28, S30, S32, S33, S45
TRL 5	S03, S05, S06, S08, S10, S13, S14, S16, S22, S23, S25, S29, S31, S35, S37, S38, S42, S44, S47
TRL 6	S17
TRL 7	S34
TRL 8	S43

None of the works found are at TRL 9, due to none being at a continuous production stage. As all the tools were at least at the TRL 1, they had their concept and

[2]https://www.embrapa.br/escala-dos-niveis-de-maturidade-tecnologica-trl-mrl
[3]https://formularios.mctic.gov.br/index.php/117963

application formulated. Only eleven of the tools were in the first three TRL levels which according to Bergamini (2020) are said to be the ones where there is only an idea of a system without prototypes and such.

Most tools are at TRL 4 and 5, corresponding to the optimization and prototyping stages, respectively. This indicates that while many tools have been developed, their progress has largely stalled at these levels.

Only three of the tools were above TRL 5 showing that most are still not in their last stages of development and have not been getting updates frequently. Figure 4.9 shows a graph of the TRL level of the tools.

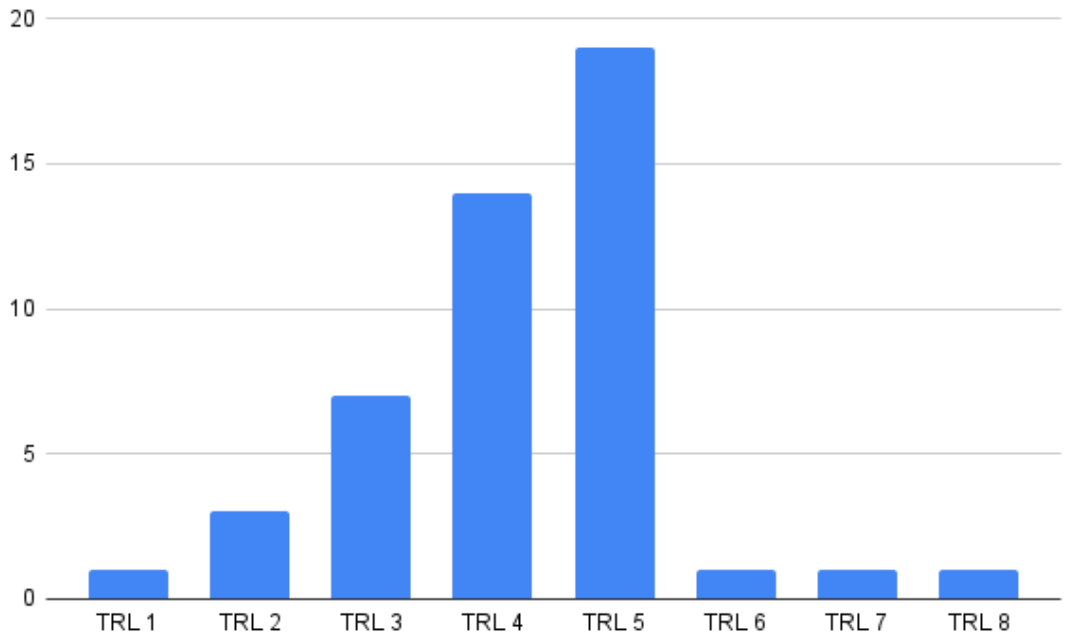

Figure 4.9: TRL of the tools

RQ.3 Summary:
We found that most of the primary works selected stopped at the prototypes and evaluation stages of the tool by using the TRL scale to measure them. Most of the tools are concentrated in the TRL 4 and TRL 5 group while only a few managed to be at higher maturity levels.

4.2.4 Target Audience (RQ4)

To answer this research question the papers were analyzed regarding the audience that the tool proposed in the paper aims to help. All the tools found have the objective of helping people with VI, but some of the tools found do not have people

who are blind (visual acuity worse than 3/60) as their target audience or do not mention them in the paper.

As Table 4.8 and Figure 4.10 show, of the 47 works, 42 are aimed at both blind students and students with other types of visual impairments, and only 5 of them are aimed only at students who are not blind.

Table 4.8: Distribution of primary studies by visual impairments types

VI Classification	**ID**
Blind and Visually Impaired Users	S01, S02, S03, S05, S06, S07, S08, S09, S10, S12, S13, S14, S15, S16, S17, S18, S19, S20, S21, S22, S24, S25, S26, S27, S28, S29, S30, S32, S33, S34, S35, S36, S37, S39, S40, S41, S42, S43, S44, S45, S46, S47
Only Visually Impaired Users	S04, S11, S23, S31, S38

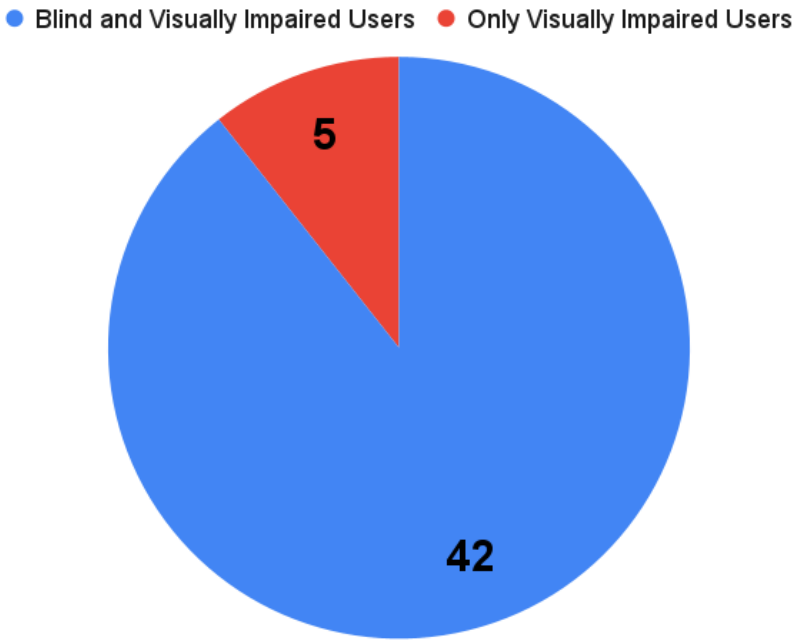

Figure 4.10: Classification of the target VI audience of the primary works

As for which field was the target audience of the studied primary works overall, we divided it into four categories: 'Academia', if the work focused on schools, colleges, and universities with a focus on students and teachers; 'Industry', if the work focuses

on companies and the work exerted by programmers; 'Both' in case both apply and 'Not mentioned' in case both are not mentioned in the article.

Table 4.9: Distribution of primary studies by their target audience

Target Audience	**ID**
Academia	S01, S02, S03, S04, S07, S08, S11, S15, S16, S18, S19, S21, S25, S27, S28, S29, S30, S31, S32, S33, S34, S36, S38, S39, S40, S41, S43, S44, S45, S47
Industry	S13, S22, S23, S26, S35, S37
Both	S05, S06, S09, S10, S12, S14, S17, S20, S42
Not Mentioned	S24, S46

This division was done to compare the results of this table with the other previous results and see if tools tailored for either the Industry or Academia are more available and better evaluated. It was found that out of the nine tools that were found to be available, most of them focused on the Academia (S02, S31, S34, and S43) or were classified as both (S05, S06, S17, and S42) with only one being focused solely on the Industry (S22) as seen in Table 4.5 and Figure 4.11.

On the other hand, all six studies focusing on the Industry (S13, S22, S23, S26, S35, S37) were evaluated with seven or more SVI showing a more thorough process of evaluation when a tool is designed for the Industry.

RQ.4.1.1 Summary:
The tables showed that most studies had the Academia exclusively as their target audience and also that the tools presented in the primary works usually include not only those with mild, moderate, and severe visual impairment but those who are blind as well. The comparison between tables also revealed that studies focused on industry tend to have a more rigorous evaluation process, it was also shown that the current tools available either cater to both academia and industry or primarily focus on academia.

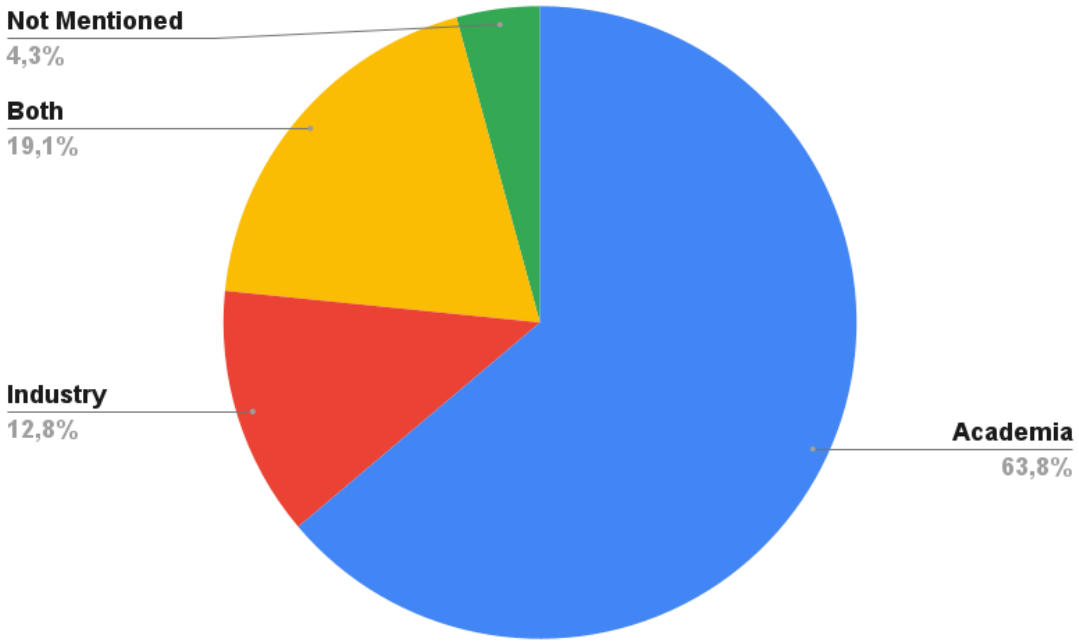

Figure 4.11: Classification of the target field of the primary works

4.3 Systematic Review Discussion

This section discusses the main findings of the systematic review done in this study.

Availability of the developed tools

When analyzing the selected studies, we could observe that most of them had either no longer available links or no links to the tools that were developed as shown in Table 4.5. This shows that most of the tools that get published in research papers either stop getting maintenance, do not get finished by the authors in the case of prototypes, or are not made available to the community that would benefit from them.

Only seven (S06, S17, S22, S31, S34, S42, S43) of the forty-seven primary studies analyzed mentioned in the paper provide a link where you could access the tool and that link was still available by the time this research was done. Two other tools (S02, and S05) were found with a simple search in web search engines even though they were not mentioned in the paper.

This makes a total of approximately 80% of the studies that are not currently available to the public showing a staggering amount of research papers where the tools that were prototyped or created were not made available or are no longer available to be used by their target audience.

Most of the available tools are related to programming, 6 out of 20 programming

tools, 1 out of 14 diagramming tools, 1 out of 3 computer network tools, and 1 out of 2 robotics tools are currently available. All the tools about the areas composing the Others category were unavailable. Even though diagramming has the second highest number of tools found in this study, only a single one of them was available to be used.

This highlights the need for tools specifically designed for SVI that are accessible and readily available for their use. It also underscores how, despite a considerable amount of research on accessibility tools, SVI still face significant challenges in computer courses (Alves et al., 2022).

Distribution of the tools by area

The distribution of the tools by area was notably more focused on two major areas, programming and diagramming. It was notable that the other areas such as computer networks, electronics, databases, and robotics were lacking in tools that can give SVI support in these courses.

Aside from the areas that had only a few tools to support it, some other areas had no tools at all that aided in their courses such as 3D modeling and mobile application development.

During the process of selection of works, we noticed a high amount of hardware-based tools for SVI in the area of robotics but an incredibly low amount of software tools for that same target audience, this may be due to the area being very hardware-based, but as much of it is still needs to be coded in software for the machines to operate, it is still a gap worth pointing out.

This shows that software-based tools are very uneven when it comes to their distribution by area in computing courses, evidencing the lack of support from assistive tools in many other areas that are not related to programming or diagramming.

Evaluation of the tools

Out of the forty-seven works, eleven of the tools had no or near to no evaluation process in the research paper published describing them, another thing of note is that some of the evaluation methods seen in the primary works lacked enough formality to be a formal evaluation test. Most of the evaluation processes used different methods to evaluate their tools depending on the tool use showcasing a lack of uniformity in the methods to evaluate assistive technologies for SVI.

This brings up questions about the effectiveness of those tools in helping SVI such as: Is the shortage of tools due to the lack of tools in general or due to the lack of high-quality tools that are useful enough to be used by SVI? What are the best ways to evaluate tools for SVI?

It is known that SVI still struggle because of the lack of tools that can help them with their activities in computed-related courses (Alves et al., 2022). This problem

could be due to the low quality of the tools developed to aid those students.

A notable aspect of the evaluation of the tools present in those primary works is the low quantity of people with VI evaluating them. Some tools try to circumvent this by using experts or simulating visual impairment by covering the vision of the participants in their evaluation. Still, it is not quite the same because people with VI use the computer mainly with their keyboards and not much with their computer mouse.

Due to the aforementioned difference in how sighted users and users with VI use the computer, evaluations without people with VI can have uncertain results about how well the tool works for their target audience.

Many works also have a very low amount of participants increasing the amount of bias about the user's experience coding and programming on the evaluation of the tool. Some works try to reduce this bias by saying that the participant in their study is either a novice or a professional, but most do not, leaving the aspect of bias uncertain for the readers. Figure 4.8 shows that many works have less than twenty participants for the primary works studied, with many of them having three or fewer participants per study showcasing even more the lack of SVI in the process of evaluation of the tools.

Technology Readiness Levels of the tools

Regarding the technology readiness levels of the tools, most of them are in the middle levels of technology readiness, with only a few of them reaching the highest levels (TRL 7, 8, and 9). This means that many tools get created, tested with a relevant public, and evaluated but they do not get released and maintained for the target audience to use after the scientific paper is done.

Some of the works also present tools that never got any indication of surpassing the lower levels of the TRL metrics. Most of those with TRL 4 or lower were not evaluated or the evaluation of the tool was done in a very informal way and not documented properly warranting them their position on the lower levels.

The TRL alongside the information on how many works are still available and the information about how their evaluation was done showcases the current state of assistive technologies for SVI in computing courses in higher education where many of the tools that would be useful for that target audience are currently unavailable for use even with a decent amount of academic works that focus on this subject.

It is necessary to not only ensure that the developed tool is of high quality but also to maintain those tools so that they can continue giving continuous support to those who would greatly benefit from their use in their academic careers.

Target audience of the tools

During the study, it was found that most of the tools are tailored to suit both the needs of completely blind and low-vision students, usually through the means of audio feedback. This shows that most of the tools developed for this target audience are inclusive regardless of the user's visual acuity.

Most of the tools developed present in the primary studies analyzed also have a focus on academia focusing primarily on students who are going through computing-related courses. This is highly likely due to the search string utilized to find the papers involving keywords that favor works related to academia.

Challenges

We identified multiple challenges in the research on tools for SVI in higher education that help point out possible gaps in the studies about ATs for VIs in the computing field.

1. *Scarcity of working tools.* During the study it was seen that many of the fields in computing courses are scarce when it comes to working ATs that are available to be used to learn how to code or create diagrams. This scarcity is even stronger when it comes to diagramming seeing how many of the tools that are created for this area are unavailable to be used creating a big gap when it comes to the diagramming field.
2. *Lack of thoroughness in the primary studies.* Many of the primary studies found were excluded by the exclusion criteria because they had less than five pages or could not provide meaningful results for the research due to the lack of thorough explanation about the process of development of the tool, its functionality, and the lack of an evaluation process.
3. *Lack of scientific vigor in the tools' evaluation.* Ten of the primary works we investigated for this systematic review had no sections explaining the process of evaluation of the tool created, and many others utilized informal methods to evaluate the tools that they created.
4. *Evaluation without the target audience.* Another big issue we saw in the process of evaluation is that many of the works had little to no participation of people with VI during the evaluation of the tools.

4.4 Systematic Review Related Work

Some studies were found about tools or methods that help in the education of SVI. The focus of these works is diverse, some with a more general scope focusing on accessible education itself, others more specific, focusing on one of the areas of computing. Of the works found, two are systematic mappings, three are bibliographical

reviews and five are systematic reviews. Table 4.10 shows the titles, types, areas of focus, and references of those papers.

Table 4.10: Related works

ID	Title	Type	Reference
RW01	Technologies in Education for Visually Impaired People: A Literature Review	Bibliographic Review	López Flores and González Lara (2023)
RW02	Approaches for diagrams accessibility for blind people: a systematic review	Systematic Review	Torres and Barwaldt (2019)
RW03	Teaching robot programming activities for visually impaired students: A systematic review	Systematic Review	Damasio Oliveira et al. (2017)
RW04	Addressing Accessibility Barriers in Programming for People with Visual Impairments: A Literature Review	Bibliographic Review	Mountapmbeme et al. (2022b)
RW05	A Tecnologia Assistiva e a Inclusão de Educandos com Deficiência Visual: um Mapeamento Sistemático da Literatura	Systematic Mapping	de Melo et al. (2019)
RW06	Um Mapeamento Sistemático sobre o Ensino de Programação para Pessoas com Deficiência	Systematic Mapping	Araújo and Andrade (2020)
RW07	Teaching programming for blinds: A review	Systematic Review	Al-Ratta and Al-Khalifa (2013)
RW08	Making Programming Accessible to Learners with Visual Impairments: A Literature Review	Bibliographic Review	Hadwen-Bennett et al. (2018)
RW09	Assistive Technology to Assist the Visually Impaired in the Use of ICTs: A Systematic Literature Review	Systematic Review	Zen et al. (2022)
RW10	A Review of Computer-Assisted Instruction for Students With Visual Impairment	Systematic Review	Tuttle and Carter (2022)

Bibliographic Reviews.

López Flores and González Lara (2023) did a bibliographic review about tools with a general scope that helps the education of people with VI, including e-mail services and urban mobility. The authors also state that many technologies were created to

support SVI and classify their results into two categories, tools focused on mobility and educational tools.

The literature review by Mountapmbeme et al. (2022b) focuses on identifying the barriers and bringing possible solutions to the problems SVI encounter in programming learning. Several barriers that disrupted the learning process of SVI were identified in this study, such as code navigation and comprehension challenges; debugging; *skimming*; and code editing. The article also lists the proposed tools found to break these barriers, these tools fit into the following categories: accessible programming tools, accessible programming languages, and accessible programming *toolkits*.

Hadwen-Bennett et al. (2018) carried out a bibliographical review on the strategies that are adopted for teaching computing focused on students in high school and higher education. The most common strategies were auditory and tactile *feedback*, accessible text-based languages, accessible block-based languages, and physical artifacts.

Systematic Mappings.

The work done by de Melo et al. (2019) brings a systematic mapping of assistive technologies and the inclusion of SVI in education with a focus on primary works written in Portuguese. The mapping contains 30 primary studies and focuses on answering questions regarding the increase in the number of assistive technologies, the distribution of the primary works concerning the levels of education and the region of Brazil in which the primary work was done, the types of tools found, and the benefits and problems encountered with the use of these technologies.

The systematic mapping written by Araújo and Andrade (2020) focuses on finding approaches that help to teach programming to people who have some type of disability. The mapping found 29 works that met the inclusion and exclusion criteria, the work focuses on answering research questions that involve: The approaches of the works found, the target audience of the works, the technologies used in these approaches, and how SVI are assessed at schools and universities.

Systematic Reviews.

Torres and Barwaldt (2019) carried out a systematic literature review with a much more specific scope of approaches that help to make science, technology, engineering, and mathematics (STEM) diagrams more accessible to people who are blind. In the systematic research carried out, they point out that audible interface approaches are predominant along with speech output approaches, which they point out to be a consequence of modern operating systems and applications being accessed usually with the help of some screen reader. Also during the study, it is notable that there is an uprising of an alternative to audible interfaces, tactile graphics.

The primary work done by Damasio Oliveira et al. (2017) was a systematic review that focuses on the use of robotics for the education of SVI to discover which methodological procedures are being used in the teaching of programming with robots for SVI. Al-Ratta and Al-Khalifa (2013) in their brief systematic literature review focusing on teaching programming for blinds, brings details about the quantitative details, publication topics, programming languages, and assistive devices found in their research.

Zen et al. (2022) wrote a systematic review of resources and tools related to software development that also focused on programming logic and computational thinking. In their search, they found 12 articles that met their criteria and are related to learning computer programming and the use of software development tools by people with VI, as for the work of Tuttle and Carter (2022), they did a systematic review to examine eight CAI (Computer-assisted Instruction) intervention studies implemented with 92 U.S. school-age children with visual impairments to make a comprehensive examination of the studies addressing those interventions.

To show the areas on which each of the works is focused, Table 4.11 divides the works into four different categories: education in general (education in any field without a focus on the computing area), programming, diagrams, and robotics.

Table 4.11: Distribution of related works by their area

Area	**ID**
Education in General	RW01, RW05, RW09, RW10
Programming	RW04, RW06, RW07, RW08
Diagrams	RW02
Robotics	RW03

As we can see through Table 4.11, these works are more focused on only one of the areas that are present in the curriculum of higher education in computing, such as programming, leaving aside other important courses such as computer networks, diagramming, robotics, and data structure. These works also do not focus much on the evaluation of the articles and are only concerned with listing and briefly explaining the proposed tools without verifying whether these articles certify that the proposed ATs are effective and fulfill their objectives.

With the lack of analysis and evaluation of how the authors of these primary works validate and evaluate their papers and what characteristics and metrics are taken into account for this validation, there is still information to be discovered in this area that may help to uncover the reason for the existence of barriers in the education of computing for SVI even with a plethora of proposed tools. This is one of the gaps left in this area that is going to be addressed with this study.

With the systematic review, a comprehensive analysis of the assistive technologies developed for SVI in ICT courses is being done including all the areas of focus instead

of only one, the main objective is to extract both general and in-depth information from those studies such as their availability, evaluation methods, Technology Readiness Level (TRL), and target audience, which is not present in the other systematic reviews mentioned above.

4.5 Threats to Validity

There are some threats to the validity of this systematic review. They are discussed in the following subsections:

- **Vocabulary:** The search was guided by the search strings written in both English and Portuguese that contain many keywords often seen when addressing works about accessibility towards SVI. However, some relevant studies may have been missed by not utilizing those specific keywords in their titles or abstracts.
- **Research Questions:** The research questions are general to get a better overview of the current situation in the field, and more specialized research questions would allow further research.
- **Publication bias:** This study is limited to the digital libraries mentioned in Table 4.1, there may be more tools that have not been published in research papers and therefore have not been selected for the systematic review. Our study is also limited to works published in Portuguese or English, limiting the scope of research.
- **Subjectivity bias:** The selection process where the inclusion and exclusion criteria were applied was done by only the first author, with the other authors aiding in the last stage of the process where the extraction of relevant information to the research questions happened. The classification of the primary works by their technology readiness levels was done solely by the main author of this article. This bias was reduced by the use of the methodology of evaluation of another work to classify the works in their correct category according to the metric.

4.6 Chapter Summary

In this chapter, we discussed the research questions, review strategy, and the results of the Systematic Review. To find works that addressed tools that help SVI in higher education courses, we found 9,066 works by the search string we created, and out of those works we selected 47 for this systematic review after carefully examining the works.

The results of the systematic review were divided by research question. The first research question had the objective of listing and verifying the availability of the

tools created to assist SVI in higher education. Only 9 of the 47 works had tools that were currently available to use by the time the research took place, this issue is even worse in the diagramming field where only 1 out of 14 tools were available.

The second research question had the goal to find the methods used for the process of evaluations of the tools. Most of the tools were evaluated through usability tests and surveys, with less than one-third of the works using other methods such as interviews, experiments, case studies, and focus groups. Another important finding is that many works had no process of evaluation of their tools whatsoever and some of them were evaluated without the involvement of participants with VI.

The third question verified the maturity level of the tools according to the TRL metric, it was found that most of the tools are concentrated in the levels TRL 4 and 5. As for the fourth question, we verified the target audience of the tool, most of the tools found have their focus mostly on academia.

Chapter 5

CraftPy

According to the results found in the Systematic Review performed and the discussion about them in Chapter 4.3, it was shown that one of the biggest gaps in the field of assistive tools for SVI is the high quantity of unavailable diagramming tools that we presently have. Only one of the fourteen tools found was available online during the systematic review.

To mitigate this problem, we created the `CraftPy` tool to assist ICT students who are blind or visually impaired in building diagrams through the Python programming language to ensure that the users would not need to learn a new specific programming language to learn how to use the tool. By the time this study took place, `CraftPy` allows users to create class diagrams, use case diagrams, and entity-relationship diagrams, commonly seen in software engineering and database courses in universities. The `CraftPy` tool is available at the following link: https://lucaskart.github.io/craftpy/

We also performed an ad-hoc search to verify if the existing tools are compatible with screen readers. Performing tests with the StarUml[1], Draw.IO[2], Astah[3], and Umple[4] tools using the NVDA[5] screen reader, we found that none of them work well with screen readers, being hard and complex to navigate and having unreadable content.

According to the TIOBE[6] index for April 2024, Python is currently the most popular language with more than 6% ahead of its competitors. The TIOBE index is an indicator of the popularity of programming languages updated every month using famous websites such as Google, Bing, and Wikipedia to calculate their ratings. For that reason, Python was chosen to be the language that is used to generate the UML

[1]https://staruml.io/
[2]https://draw.io/
[3]https://astah.net/
[4]https://cruise.umple.org/
[5]https://www.nvaccess.org/about-nvda/
[6]Avaliable on: (https://www.tiobe.com/tiobe-index/)

diagrams. Therefore, it is not necessary to learn a new language that would be only used inside our tool.

This chapter is divided into seven sections.

Section 5.1 presents the initial details of the development of the `CraftPy` tool;

Section 5.2 presents the technologies utilized by the `CraftPy` tool;

Section 5.3 gives details about the steps and processes of the `CraftPy` tool and a comparison of the `CraftPy` tool with the other tools found in the systematic review;

Section 5.4 shows examples of Python codes to create diagrams with the `CraftPy` tool;

Section 5.5 demonstrates `CraftPy` through a usage example;

Section 5.6 explains the evaluation that took place to assess the `CraftPy` tool;

Section 5.7 gives details about the limitations of the `CraftPy` tool;

Section 5.8 shows the threats of validity of the evaluation of the `CraftPy` tool;

Section 5.9 is a summary of the chapter.

5.1 Tool Development

The development of our tool was inspired by a prototype presented by Verde et al. (2023). A prototype aimed at assisting SVI in building UML diagrams through the Python programming language. By the time this work took place, Py2UML was unavailable to the general public due to it not being available online.

Their tool, which generates a single diagram, has several limitations and is not publicly available. We believe that the most important aspect of a tool aimed at SVI is to make it accessible to users. Therefore, we completely redesigned the tool's code using different technologies to address these issues.

This work changed the technologies used by the tool to allow it to be hosted online, expanded the tool to enable the creation of more types of diagrams, and created more examples and documentation to help with the use of the tool. The development of `CraftPy` took into consideration the Web Content Accessibility Guidelines (WCAG) to ensure that the tool would be accessible to its target audience.

The WCAG are guidelines that contain a wide range of recommendations for accessibility in websites to ensure that the content is accessible for people with disabilities (Caldwell et al., 2008).

The WCAG follows four principles shown in the work made by Caldwell et al. (2007) that claims content in a web page needs to be:

- **Perceivable:** The information and components in the web page must be presented in a way the user can perceive it with at least one of their senses.
- **Operable:** The navigation and components in the web page must be operable, and thus cannot require interaction that a user cannot perform.
- **Understandable:** The information and operations of the web page must be understandable, therefore it cannot be beyond the understanding of a user.
- **Robust:** The web page content must be interpreted reliably by a multitude of user agents including ATs.

In its current state, the tool can create three types of diagrams and draw multiple items unique to those diagrams, the following diagrams and concepts can be created with `CraftPy`:

- **Class Diagrams**: `CraftPy`'s class diagrams can create classes, attributes, visibility markers, multiplicity, associations, aggregations, compositions, specialization, and dependency.
- **Use Case Diagrams:** `CraftPy`'s use case diagrams can create actors, use cases, inclusion between use cases, and extension between use cases.
- **Entity-Relationship Diagrams:** `CraftPy`'s entity-relationship diagrams can create entities, attributes, relationships, primary keys, foreign keys, weak entities, identifying relationships, multiplicity, and multivalued attributes.

A more technical overview of the technologies used to create `CraftPy` can be seen in the next section.

5.2 Tool Architecture and Technologies

`CraftPy` is a web application developed using the Vite[7] development server, the React[8] library, and TypeScript[9]. Vite serves as a fast and lightweight bundler for the front end, while React provides a robust framework for building components that encapsulate logic and user interface into independent, reusable units. TypeScript enhances the code's security and reliability with its static typing features. Also, `CraftPy`'s architecture follows the principles of Single-Page Application (SPA) development, where most of the processing logic is handled on the client side, resulting in a more fluid and responsive user experience.

Accessibility and responsiveness are crucial for ensuring that the web application is inclusive and adaptable to different devices and users, especially since the tool's target audience includes people with visual impairments. To achieve this, we use

[7]https://vitejs.dev/
[8]https://react.dev/
[9]https://www.typescriptlang.org/

the RadixUI[10] component library and the TailwindCSS[11] framework. TailwindCSS enables responsive styling of React components, allowing them to adapt seamlessly to various screen sizes and devices. Radix UI provides accessible components out of the box, ensuring an inclusive experience from the outset of development.

Additionally, we evaluated `CraftPy` with Google PageSpeed Insights (PSI)[12]. PSI reports on a page's user experience and offers suggestions for improvements. For `CraftPy`, PSI revealed a maximum score in accessibility for desktop devices, confirming that all components are accessible to screen readers and support keyboard navigation.

No JavaScript libraries were found for analyzing Python code to extract class, attribute, and method information. Instead, regular expressions were implemented, disregarding Python's mandatory indentation conventions and using reserved words as delimiters. For instance, 'def' marked the start of a method, with its end indicated by another 'def' or 'class' declaration. If a class was improperly formed, its code block was discarded, retaining only valid parts for diagram representation.

At the end of the analysis, a JavaScript object is generated containing all the information separately, enabling the construction of any type of diagram. This means that the code analysis is unique, and each diagram is built according to the established rules, providing the necessary context for each analyzed element. Subsequently, for visualization of the diagrams, a DOT[13] file was generated and rendered using the d3-graphviz[14] library.

`CraftPy` is hosted and deployed through GitHub Pages[15]. The application is deployed using continuous integration and continuous delivery (CI/CD) pipelines, which automate the build, testing, and deployment process. This ensures that new versions are made available quickly and securely, keeping the application always up-to-date and stable for end users. `CraftPy` tool is licensed under the GNU General Public License and all source code is available on GitHub.

5.3 Tool Features

The `CraftPy` tool has multiple features in its system. The main function is the creation of multiple kinds of diagrams, such as class, use case, and entity-relationship diagrams. Figure 5.1 shows the main interface of the system. At the top, we have the navigation bar of the system with buttons redirecting to the other tabs, such as the examples and help tabs and the source code of the tool hosted on GitHub.

[10] https://www.radix-ui.com/
[11] https://tailwindcss.com/
[12] https://pagespeed.web.dev/
[13] https://graphviz.org/
[14] https://www.npmjs.com/package/d3-graphviz
[15] https://pages.github.com/

Below the navigation bar, there are seven buttons lined horizontally. The first button is a select button called **"Code Examples"** that allows the user to test many premade examples to help understand how the tool works and show what a completed diagram looks like. The second button **"Save Code"** allows users to save their written Python code in a file. The third to fifth buttons are buttons that allow the user to select whether to create a class, use case or entity-relationship diagram. The last two buttons allow the user to save the generated diagram: The first one **"Save .dot"** saves the diagram in a .DOT file, an extension used for Graphviz graphs, and the last one **"Save Diagram"** allows the user to save the diagram as an image.

The lower parts of the page are divided in two: The first is the code editor where the user inserts the Python code that is going to create the diagram; and in the second section, the generated diagram is presented with XML elements.

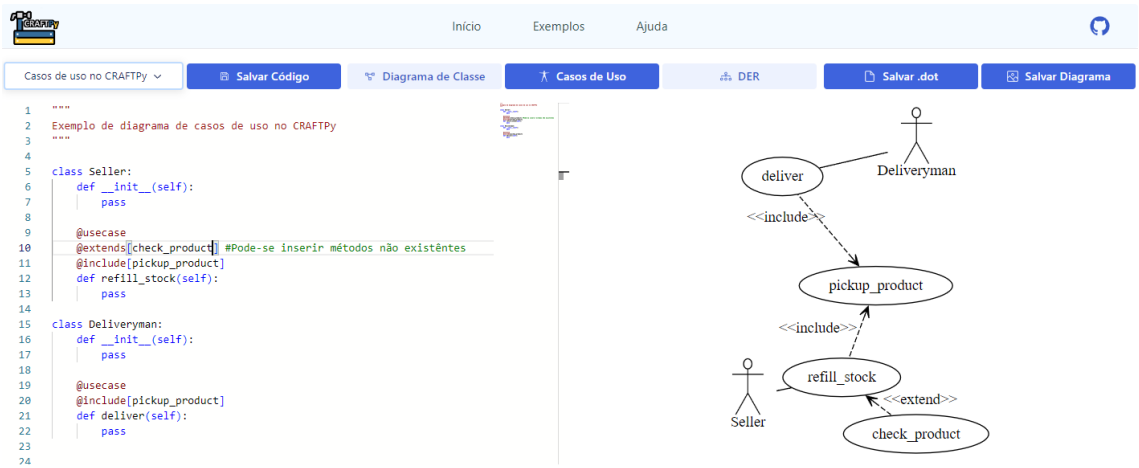

Figure 5.1: System interface.

The `CraftPy` tool is a UML diagram generator that allows the creation of multiple kinds of diagrams, such as class and use case diagrams, for Python programmers. The target audience of the tool is VI students who cannot use the visual aspects of most of the main UML tools. `CraftPy` allows users to write, edit, and execute Python code online in their browsers without installing any tool to generate diagrams based on the Python code. As it uses Python code to operate, `CraftPy` removes the learning curve that other tools have of learning a specific programming language to use them.

The process flow in `CraftPy` is divided into two main parts, the first focuses on the user, and `CraftPy` executes the second. First, the user writes the Python code and selects the type of diagram they want to generate; both actions can be done interchangeably through the interface. In the meantime, `CraftPy` captures the Python code, translating it into the DOT language thus generating a visual diagram. This flow is shown in Figure 5.2.

`CraftPy` offers the user multiple examples of code that can be compiled and turned

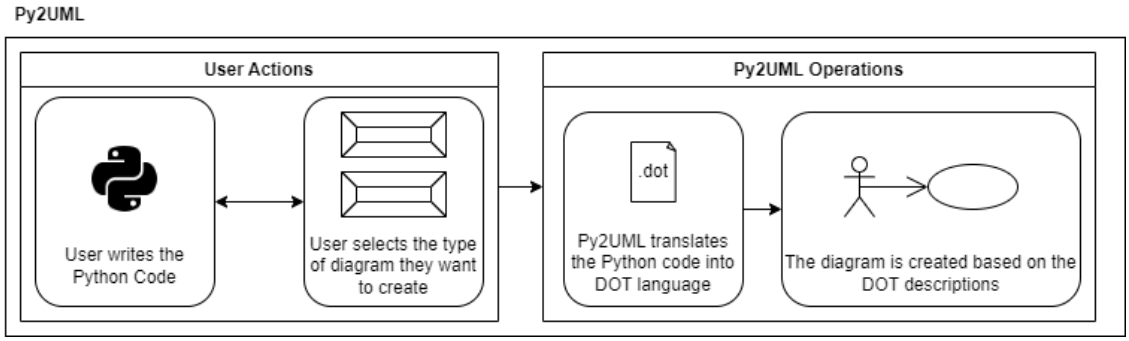

Figure 5.2: CraftPy's process flow

into diagrams to help the users develop their diagrams through Python code. Both the Python code and the diagram can be downloaded through the website.

`CraftPy`'s interface was created to allow VI students to easily navigate through the page and create diagrams without the need for visual elements. It has compatibility with screen readers such as NVDA allowing the elements of the page to be read out loud for VI students.

5.3.1 Comparison with other tools

Compared to the other tools found on the Systematic Review, `CraftPy`'s advantages lie in the fact that out of the tools that can create diagrams, `CraftPy` is one of the most versatile when it comes to the variety of diagrams it can create. Table 5.1 compares the features of the tools of the primary works of the systematic review.

Some of the primary works do not specify what kind of diagram can be created by their tool, thus we used a "-" symbol to signal that the type of diagram is unclear.

While the **ModelByVoice** prototype (S30) does not have restrictions when it comes to the type of diagrams it can create, there was a lack of working links to access the prototype created and the way the diagrams are created is vastly different, following a speech-to-text format to create them.

`CraftPy`'s functionalities allow creating diagrams without the need for other software and installing anything on the user's computer as it is a web application. It also allows users to create different sorts of diagrams at the same time, a feature that most of the tools found in our study do not have.

5.4 Tool Diagrams

`CraftPy` allows users to create three kinds of diagrams: class diagram, use case diagram, and entity-relationship diagram. This section summarizes the instructions in Python code to generate the diagrams in the `CraftPy` tool.

Table 5.1: Table of diagramming tool features

	Avaliable Online	Focus on UML	Can Create Diagrams	Types of Diagrams Created by the Tool
S01	No	No	Yes	Flowchart
S11	No	No	Yes	Entity-Relationship
S12	No	No	No	-
S14	No	Yes	No	-
S15	No	Yes	Yes	Use Case
S20	No	Yes	Yes	Class
S29	No	Yes	Yes	Use Case
S30	No	No	Yes	Not Specified/Any
S35	No	Yes	Yes	Use Case
S39	No	Yes	No	-
S41	No	Yes	No	-
S42	Yes	Yes	No	-
S44	No	No	Yes	N^2 charts
S47	No	No	No	-

5.4.1 Class Diagram

To create a class in the diagram, it is needed to write a Python class. To add attributes to the class, it is necessary to create a constructor for the class using the *'__init__'* method. To add functions to a class, it is necessary to define the functions within the scope of the class. It is possible to create private attributes and methods through two subtraces. The following example shows a *Car* class with a private attribute (*brand*) and a given method (increaseKM):

```
class Car:
  def __init__(self, brand:str):
    self.brand = brand

    def increaseKM(self):
        pass
```

A class can inherit from another by using parentheses when it is instantiated. It is also possible to create an aggregation between classes when one class uses objects of another class in its constructor. In the following example, the *windows* list is a list of Window's type:

```
class Automobile:
    pass

class Car(Automobile):
    pass
```

To create an association between classes in the diagram, it is necessary to instantiate an object within a class that belongs to another class outside the constructor to avoid dependency relationships. Multiplicity can be achieved through the use of lists. Example:

```
class Car:
  __tire: list[Tire] = list()
```

The example above also creates a visibility marker for the tire attribute. To make an attribute private to a class you insert two underlines (__) before the name of the variable.

Aggregation and composition are two types of class associations in object-oriented programming. Aggregation involves one class using objects of another class as part of its structure, often by passing instances of one class as arguments to the constructor of another. In contrast, composition entails a closer relationship, where one class directly creates an object of another class within its constructor. For example, in aggregation, a class may have a member variable referencing objects of another class, whereas, in composition, one class owns or manages the lifecycle of another class. Example:

```
class Car(Auto):
    def __init__(self, brand:str, windows:list[Window]):
        self.brand = brand
        self.windows:list[Window] = windows # Aggregation
        self.owner = Owner(name="Marcos") # Composition
```

5.4.2 Use Case Diagram

To represent actors in a use case diagram using Python, each actor is modeled as a class. To create an actor in the diagram, you need to define a new Python class. To implement inheritance between two or more actors, simply create inheritance between their classes. For example:

```
class Actor:
class Admin(Actor):
class User(Actor):
```

To create a use case in the diagram, you need to define a function with the @usecase decorator before the function declaration. For an 'include' relationship between two use cases, define an additional @include decorator with the related use case name in brackets. Similarly, for an 'extend' relationship, use an additional @extends decorator with the related use case name in brackets. For example:

```
@usecase
@include[verify_captcha]
@extends[two_factor_authentication]
def login(self):
```

```
    pass

@usecase
def verify_captcha(self):
    pass

@usecase
def two_factor_authentication(user):
    pass
```

5.4.3 Entity-Relationship Diagram

To create an entity in an Entity-Relationship diagram, it is necessary to start by defining a Python class. To add attributes to the entity, it is needed to create a constructor for the class. Attributes that are lists, tuples, and dictionaries are automatically considered as multivalued attributes. To define primary keys, it is necessary to declare them as class attributes outside the constructor. In the case of foreign keys, the variable name must begin with two underscores. An entity is automatically identified as a weak entity if it does not have a primary key. An example is shown below.

```
class Entity:
    primaryKey = 3  # Primary key
    __foreignKey = 2  # Foreign key
    def __init__(self, name, attributes):
        self.name = name
        self.attributes = attributes
        self.multivaluedAtt = (5,6) # Multivalued attribute
```

To establish a relationship between two entities in the diagram, it is necessary to define a decorator @relationship with the name of the related entity in brackets. To specify the relationship's multiplicity or cardinality, it is necessary to use the @multiplicity decorator with the multiplicities of both parts separated by a colon.

```
class Project():
    pass

class Employee():
    pass

    @relationship[Project]
    @multiplicity[n:m]
    def workOn():
        pass
```

For identifying relationships between two entities, it should be used the @identifyingrelationship decorator with the related entity's name in brackets, along with the @multiplicity decorator to specify the relationship's multiplicity or cardinality, separating the multiplicities of both parts with a colon.

```
class Order():
    pass

class Customer():
    pass

    @identifyingrelationship[Order]
    @multiplicity[1:1]
    def makeOrder():
        pass
```

5.5 Usage Example

In this section, we present three different examples of the tool that is used to create three different kinds of diagrams. Figure 5.3 shows a class diagram created by using `CraftPy` adapting an example from Fowler (2018) book. The adapted diagram has five classes in total, starting with the "Customer" class, which contains information about a customer who orders a good from an automatic system. The classes "PersonalCustomer" and "CorporateCustomer" inherit from "Customer", symbolized by the arrow present in the diagram. The "Order" class represents an order in the system, "Customer" and "Order" have an aggregation relationship with each other, where customers can make multiple orders but an order can only have a single customer. Similarly, the "OrderList" class that represents the list of orders in the system shares a similar relationship with the "Order" class.

For the use case diagram, we adapted another example from the (Fowler, 2018) book. In this diagram, we have four actors: "Trader", "Salesperson", "TradingManager" and "AccountingSystem", each with their own use cases. Both the "AnalyzeRisk" and "PriceDeal" use cases include another use case called "ValueDeal". This diagram can be seen in Figure 5.4.

The last of the three diagrams shown in Figure 5.5 is the entity-relationship diagram, often seen to represent databases of systems. Based on an example of the Takai et al. (2005) book, the tool created a diagram with three entities, each with its own primary key, and the relationship between each of the three entities. The relationships have an 'm..n' multiplicity signaling, for example, that a supplier can supply multiple parts and a part can be supplied by multiple suppliers.

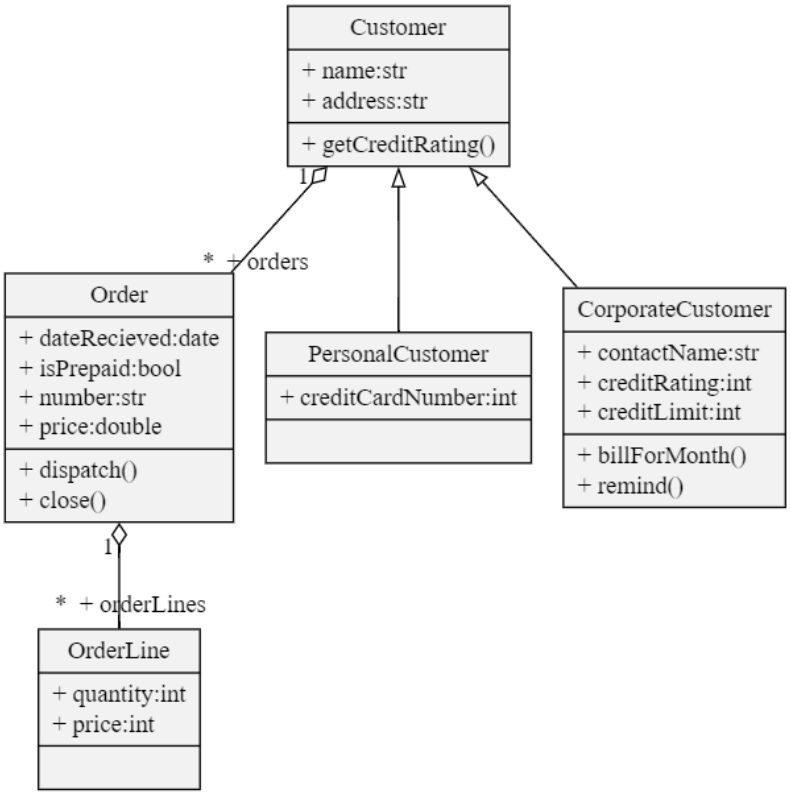

Figure 5.3: Class diagram.

5.6 Tool Evaluation

We conducted a preliminary evaluation of the `CraftPy` tool. Thus, we sent email invitations to 45 people with visual impairments and received 8 responses over 15 days, even sending some reminders. The results of the participants' forms provide insights into the effectiveness of `CraftPy` in meeting the needs of users with VI. Feedback was used to identify strengths and areas for improvement in tool design and functionality. The analysis focuses on usability, accessibility, and the overall user experience.

5.6.1 Evaluation Design

This section describes the design of our preliminary tool's evaluation.

The participants were selected from a study conducted by Alves et al. (2022). In that study, 45 students with visual impairments were surveyed to investigate their profiles, perceptions of teaching content focused on accessibility and assistive technology (AT), and areas where accessibility could be improved to promote their inclusion. The students agreed to share their emails for participation in future studies on similar topics.

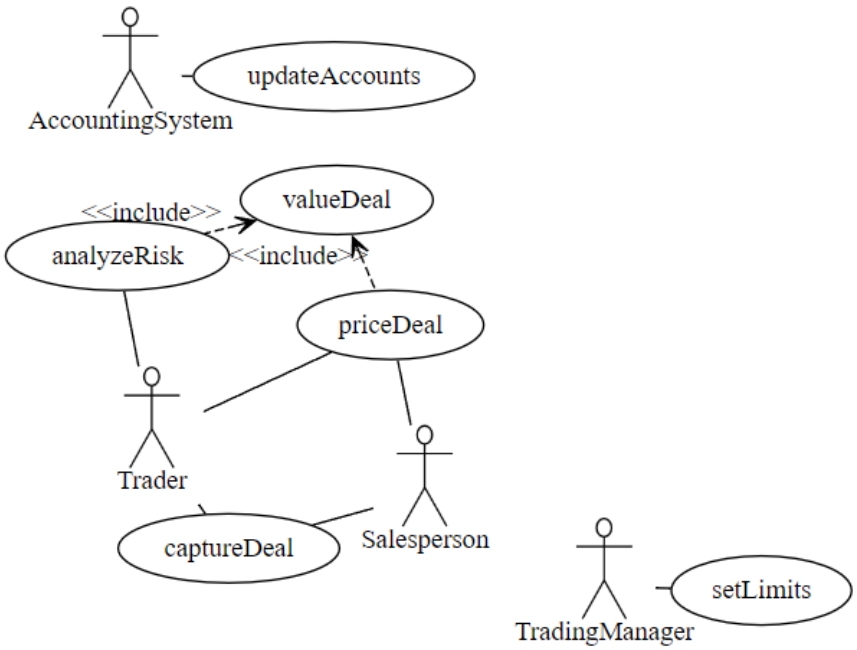

Figure 5.4: Use Case diagram.

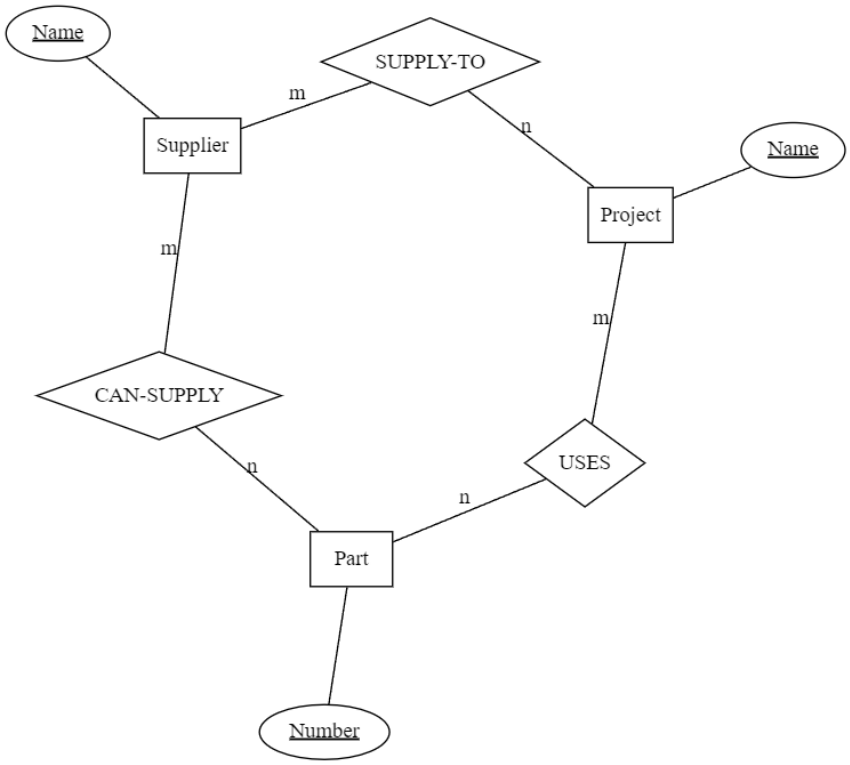

Figure 5.5: Entity-Relationship diagram.

To participate in the tool's evaluation, participants needed to meet the following criteria: have studied or are currently studying any ICT course and have basic Python knowledge. Thus, participants received an email inviting them to participate in the experiment over a period of 15 days. The email included all the necessary things for the experiment such as links to the background survey, detailed instructions for the experiment, and the final survey for feedback.

The surveys were created in Google Forms and were sent in email to the participants containing questions regarding their background in the area and about the experiment itself, the content of those surveys can be seen in Section 5.6.2 and Section 5.6.5 respectively.

Due to the low number of people with VI in the ICT field and the requirement to have basic Python knowledge, finding participants for the study was a difficult task. Some emails were no longer being used, and some reported being unable to complete the experiment in 15 days due to technical issues or lack of Python knowledge. However, as seen in 4.8, the number of participants is still higher than approximately 70% of the works found in the systematic review.

A Google Meet call was held with two of the participants who were completely blind to observe how they completed the activities present in the experiment. The experiment involved the following steps:

1. **Background Survey**: Participants completed a background survey to provide demographic information and details about their IT experience and visual impairment.
2. **Introduction to Diagrams**: Participants were provided with explanations and examples of class diagrams, use case diagrams, and entity-relationship diagrams.
3. **Experiment Tasks**: Participants were assigned three tasks, one for each type of diagram, involving minor edits to provided codes:
    - Class Diagram Task: create two new classes inheriting from another existing one and add two new attributes.
    - Use Case Diagram Task: create a new function in an existing class and mark it as a use case using the respective decorator.
    - Entity-Relationship Diagram Task: create a given key for a given class and a new function in another class, and establish a relationship with a class using the @relationship decorator.
4. **Final Survey**: After completing the tasks, participants filled out a final survey to provide feedback on their experience with `CraftPy`. The survey included questions about the usability, accessibility, and any issues encountered while using the tool.

5.6.2 Background Survey

To gather relevant data from participants before they completed the experiment, we created a Background Survey with four questions to collect demographic information, including the degree of visual impairment and their educational background in ICT.

The survey included the following four questions:

1. What is your degree of visual impairment?* (Selection Box)

- Blindness
- Near Blindness
- Severe Visual Impairment
- Moderate Visual Impairment
- Monocular Vision
- Other

2. Which technology course are you studying or have you studied?* (Selection Box)

- Computer Science
- Computer Engineering
- Information Systems
- Software Engineering
- Computer Networks
- Other

3. How old are you?* (Open Question)

4. Do you already work in the ICT field? If yes, what is your job?* (Open Question)

5.6.3 Introduction to Diagrams

To assist participants unfamiliar with the diagrams used in software modeling, we attached a document to the email that explained the different types of diagrams and two code examples in Python to be used in the `CraftPy` tool.

The explanation of diagrams and examples is presented as follows.

Class Diagram:

The class diagram is one of the most important and widely used diagrams in UML. Its main focus is on allowing the visualization of the classes that will compose the system, with their respective attributes and methods, and demonstrating how the

classes in the diagram relate, complement, and transmit information to each other. This diagram provides a static view of how the classes are organized, focusing on how to define their logical structure. (Guedes, 2018)

Use Case Diagram:

The use case diagram aims to present an overall external view of the functionalities that the system should offer to users, without focusing too much on how these functionalities will be implemented. It is mainly used in the phases of system requirements elicitation and analysis, although it is consulted throughout the modeling process and can serve as a basis for various other diagrams. It seeks to present a simple and easily understandable language so that users can get a general idea of how the system will behave. It aims to identify the actors (users, other systems, or even special hardware) that will use the software in some way, as well as the services, or functionalities, that the system will provide to these actors, known in this diagram as use cases. (Guedes, 2018)

Entity-Relationship Diagram:

The entity-relationship diagram is one of the most widely used diagrams in conceptual modeling, specifically entity-relationship modeling. It represents a description of the database in a way that is independent of implementation in a DBMS (Database Management System). The conceptual model records what data can appear in the database, but does not record how these data are stored at the DBMS level. (Heuser, 2009)

CraftPy Examples:

The first example is a ticket sales system for a movie theater, it contains six classes: Movie, Session, Client, Ticket, TicketSalesSystem, and MovieTheater. Both the Client and TicketSalesSystem classes have functions that work as use cases for use case diagrams.

The second example is a bookstore system, which is composed of four classes: Book, Customer, Sale, and BookstoreSalesSystem. The Customer and BookstoreSalesSystem have use cases and some of those use cases include other use cases as is the case with the 'Buy Book' function.

Both of those codes are compatible with `CraftPy` and can be used to see the diagram of a complete system in the tool.

Ticket Sales System for a Movie Theater:

```
class Movie:
    def __init__(self, title: str, duration: int, genre: str):
        self.title = title  # Movie title
        self.duration = duration  # Duration in minutes
        self.genre = genre  # Movie genre

class Session:
```

```
    def __init__(self, movie: Movie, time: str, room: str, price:
                                        float, available: int):
        self.movie = movie  # Movie being shown
        self.time = time  # Start time of the session
        self.room = room  # Room where the session will take place
        self.price = price  # Ticket price
        self.available = available  # Quantity of available tickets
class Client:
    def __init__(self, name: str, cpf: str, age: int):
        self.name = name  # Client's name
        self.cpf = cpf  # Client's CPF
        self.age = age  # Client's age

    @usecase
    def buy_ticket(self, session: Session, quantity: int):
        pass


class Ticket:
    def __init__(self, session: Session, client: Client, quantity:
                                        int):
        self.session = session  # Session for which the ticket was
                                            purchased
        self.client = client  # Client who purchased the ticket
        self.quantity = quantity  # Quantity of tickets purchased


class TicketSalesSystem:
    sessions: list[Session] = []  # List of available sessions
    def __init__(self):
        pass


    @usecase
    def add_session(self, session: Session):
        self.sessions.append(session)


    @usecase
    def remove_session(self, session: Session):
        self.sessions.remove(session)


    @usecase
    def find_sessions_by_movie(self, movie: Movie):
        pass


    @usecase
    def find_sessions_by_time(self, time: str):
        pass
```

```
class MovieTheater:
    def __init__(self):
        self.ticket_sales_system = TicketSalesSystem()
```

Bookstore System:

```
class Book:
    def __init__(self, title: str, author: str, genre: str, price:
                                        float, stock: int):
        self.title = title  # Book title
        self.author = author  # Book author
        self.genre = genre  # Book genre
        self.price = price  # Book price
        self.stock = stock  # Quantity available in stock


class Customer:
    def __init__(self, name: str, cpf: str, email: str):
        self.name = name  # Customer's name
        self.cpf = cpf  # Customer's CPF
        self.email = email  # Customer's email


    @usecase
    def find_books_by_author(self, author: str):
        return [book for book in self.books if book.author ==
                                            author]


    @usecase
    def find_books_by_genre(self, genre: str):
        return [book for book in self.books if book.genre == genre]


    @usecase
    @include[find_books_by_author]
    @include[find_books_by_genre]
    def buy_book(self, book: Book, quantity: int):
        pass


class Sale:
    def __init__(self, book: Book, customer: Customer, quantity:
                                        int, total: float):
        self.book = book  # Sold book
        self.customer = customer  # Customer who made the purchase
        self.quantity = quantity  # Quantity of books sold
        self.total = total  # Total purchase amount


class BookstoreSalesSystem:
    def __init__(self):
```

```
        self.books: list[Book] = []  # List of available books


    @usecase
    def add_book(self, book: Book):
        self.books.append(book)


    @usecase
    def remove_book(self, book: Book):
        self.books.remove(book)


    @usecase
    def find_books_by_author(self, author: str):
        return [book for book in self.books if book.author ==
                                            author]


    @usecase
    def find_books_by_genre(self, genre: str):
        return [book for book in self.books if book.genre == genre]


    @usecase
    def buy_book(self, book: Book, customer: Customer, quantity:
                                        int):
        pass
```

5.6.4 Experiment Tasks

For the experiment, we aimed to verify whether the tool could be effectively used by participants with VI. Thus, we formulate a total of three activities that participants should carry out. A document detailing three Python code examples was prepared to describe how these codes serve as models for the activity and how they translate into the developed diagram. After the explanation of the three codes, the activity was proposed using the concepts aforementioned in the explanation of the codes.

Each code example and task were designed to test a different type of diagram. Task 1 is related to class diagrams, Task 2 is related to use case diagrams, and Task 3 is related to entity-relationship diagrams.

The experiment protocols and tasks were created in Portuguese, and the results obtained from the participants were also written in Portuguese. The experiment included the following protocol and activities translated into English.

Experiment Protocol:

Three codes are provided for three diagrams. The text below contains an explanation of the three provided codes. After presenting the three codes, there are three activities, one for each diagram.

In order to conduct the experiment, we added the 'Code Examples' functionality in `CraftPy`'s dropdown menu, the three codes used in the tasks were named Code 1, Code 2, and Code 3, respectively "Código 1", "Código 2" e "Código 3" in Portuguese, as Figure 5.6 shows. The user can choose one of them to work on or alternate among them.

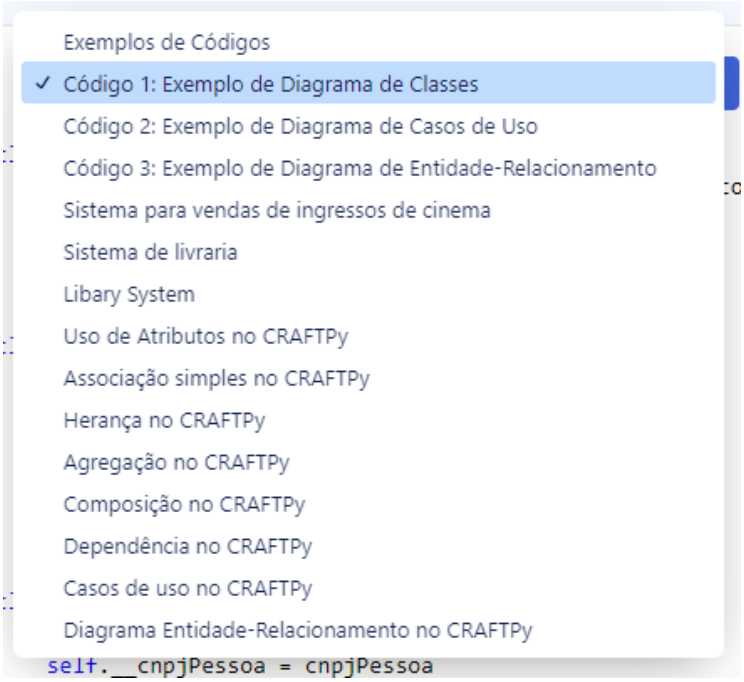

Figure 5.6: CraftPy's dropdown menu

These codes shown in Figure 5.6 are detailed in the following.

Code 1: Class Diagram.

There are five classes:

- Person
- IndividualPerson
- LegalEntityPerson
- CommonAccount
- Transaction

The class Person has the following attributes:

- personName of type string.
- personIncome of type integer.
- personAccount of type list of accounts.

The classes IndividualPerson and LegalEntityPerson inherit from the class Person.

The class IndividualPerson has the following attributes:

- personCPF of type integer.

The class IndividualPerson has the following method:

- validateCPF

The class LegalEntityPerson has the following attributes:

- personCNPJ of type integer.

The class LegalEntityPerson has the following method:

- validateCNPJ

The class CommonAccount has the following attributes:

- accountNumber of type integer.
- accountPassword of type integer.
- accountTransactions of type list of transactions.

The class CommonAccount has the following methods:

- depositAmount
- validatePassword
- withdrawAmount

The class Transaction has the following attributes:

- transactionType of type integer.
- transactionAmount of type double.

The classes Person and CommonAccount have a one-to-many aggregation relationship because the class Person has a list of CommonAccount objects as one of its attributes.

The classes CommonAccount and Transaction also have a one-to-many aggregation relationship because the class CommonAccount has a list of Transaction objects as one of its attributes.

To generate the class diagram, the user of the `CraftPy` tool needs to select the Class Diagram button in the interface or use the shortcut Alt + 2. The diagram generated with Code 1 is shown in Figure 5.7.

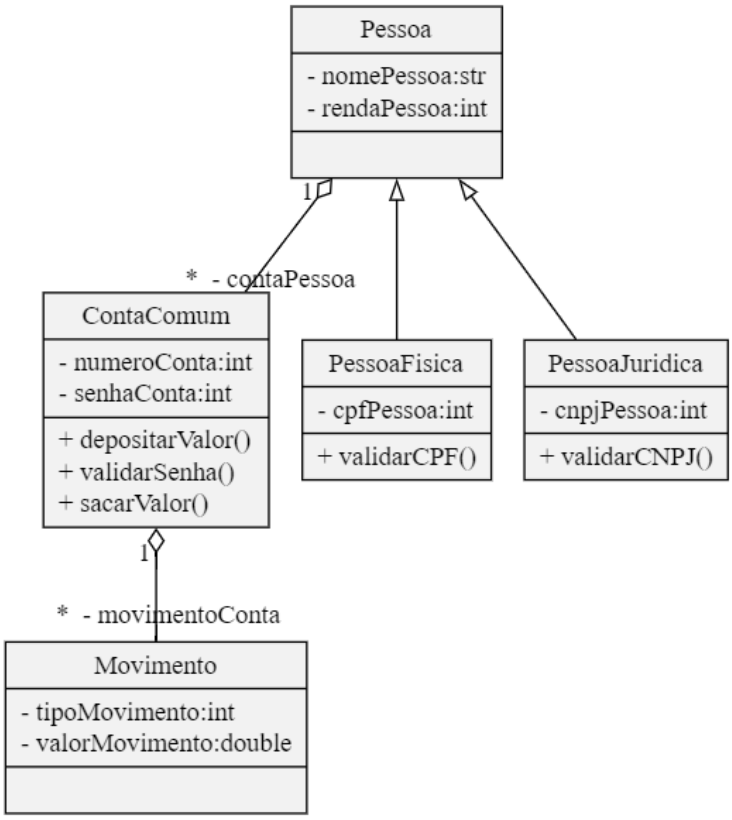

Figure 5.7: Code 1 generated diagram

Code 2: Use Case Diagram.

For the use case diagram, one more function has been added to the classes IndividualPerson and LegalEntityPerson. These functions are named consultCPF and consultCNPJ, respectively.

A decorator @usecase (at usecase) was used on the lines preceding the following functions to indicate that they are use cases:

- validateCPF
- consultCPF
- validateCNPJ
- consultCNPJ

For generating the use case diagram, the user of the tool needs to select the Use Case Diagram button in the interface or use the shortcut Alt + 3. The diagram generated with Code 2 is shown in Figure 5.8.

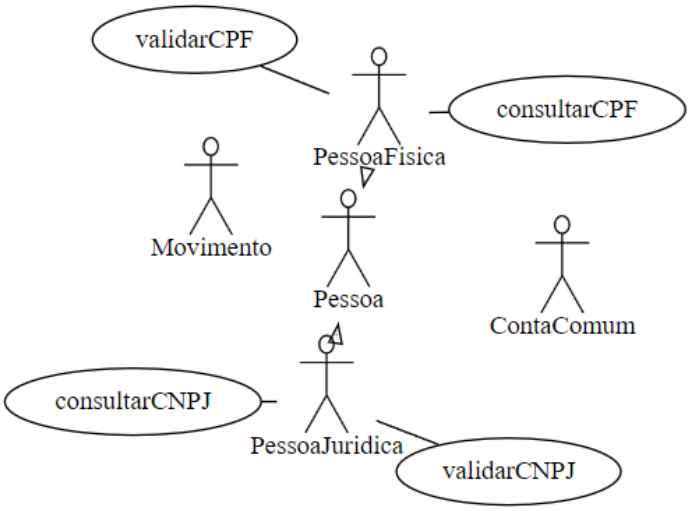

Figure 5.8: Code 2 generated diagram

Code 3: Entity-Relationship Diagram. For the entity-relationship diagram, two new attributes were created at the beginning of the class, outside the constructor, to mark them as key attributes:

- accountId in the class CommonAccount as an integer.
- personId in the class Person as an integer.
- validateCNPJ
- consultCNPJ

Two decorators @relationship (at relationship) were used to create a relationship between the classes CommonAccount and Transaction.

The decorators were added to the methods depositAmount and withdrawAmount, and they receive as a parameter, in brackets after the @relationship decorator, the name of the class they relate to. In this case, the class Transaction was added within the brackets to create the relationship between the classes.

To generate the entity-relationship diagram, select the Entity-Relationship Diagram button in the interface or use the shortcut Alt + 4. The diagram generated with Code 3 is shown in Figure 5.9.

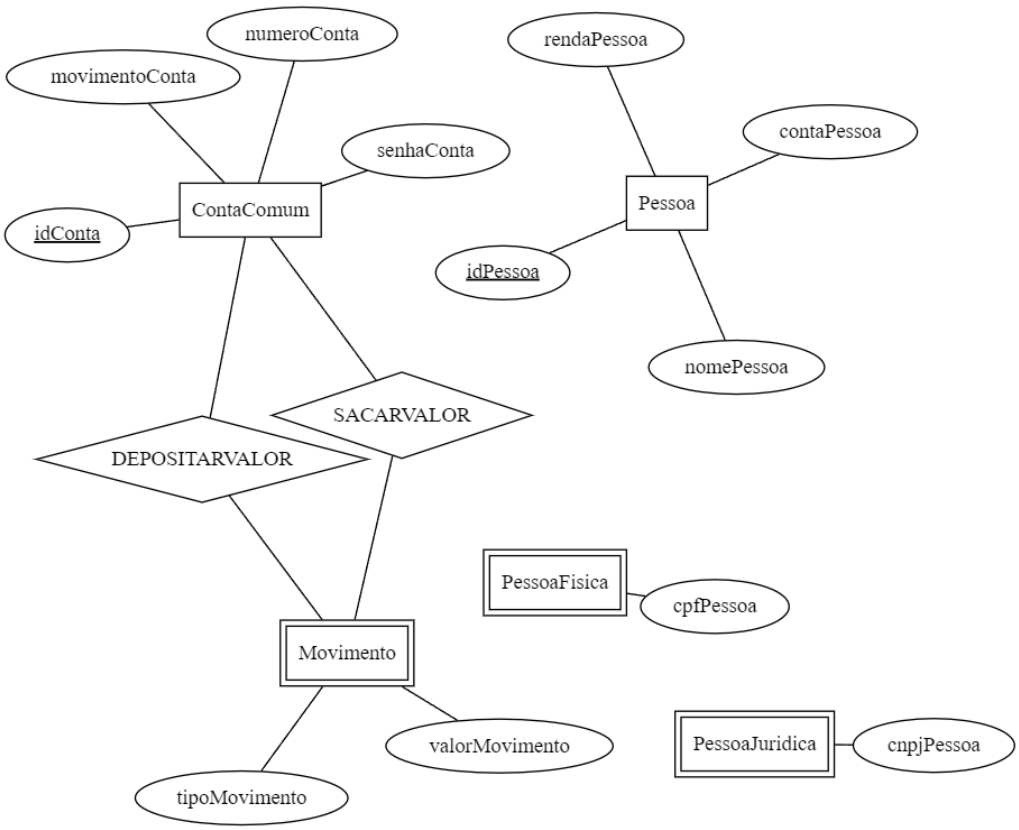

Figure 5.9: Code 3 generated diagram

Activities:

For the experiment, the participants should perform three activities that should be done using Code 3 as a base. At the end of the activity, we asked them to save the code and place it in the form provided.

Activity 1: Class Diagram

For this activity, the participant needs to:

- Create two new classes, SpecialAccount and SavingsAccount, that inherit from the CommonAccount class.
- Add the following attribute to SpecialAccount: accountLimit
- Add the following attribute to SavingsAccount: accountAnniversary

The diagram resulting from Activity 1 should resemble the one shown in Figure 5.10.

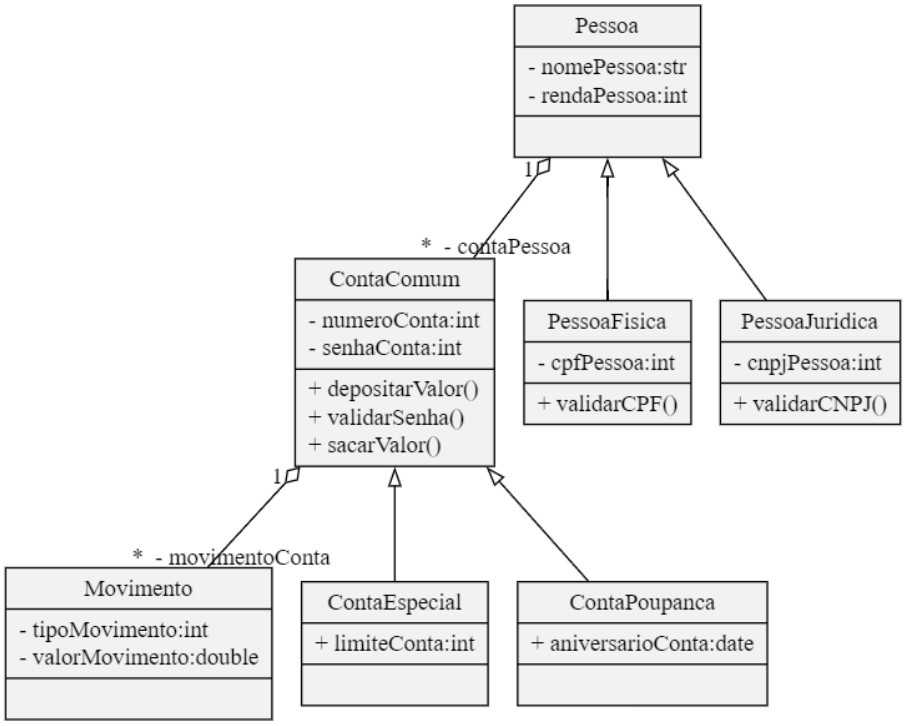

Figure 5.10: Code 1 activity answer

Activity 2: Use Case Diagram

For this activity, the participant needs to:

- Create a new function in the CommonAccount class called updatePassword.
- Mark this new function as a use case using the @usecase (at usecase) decorator.

The diagram resulting from Activity 2 should resemble that shown in Figure 5.11.

Activity 3: Entity-Relationship Diagram

For this activity, the participant needs to:

- Create a primary key for the Transaction class called transactionId.
- Create a new function in the Person class called accessAccount.
- Create a relationship in the accessAccount function using the @relationship (at relationship) decorator for the CommonAccount class.

The diagram resulting from Activity 3 should resemble the one shown in Figure 5.12.

After completing the three activities, the participant would answer the form presented in the Final Survey.

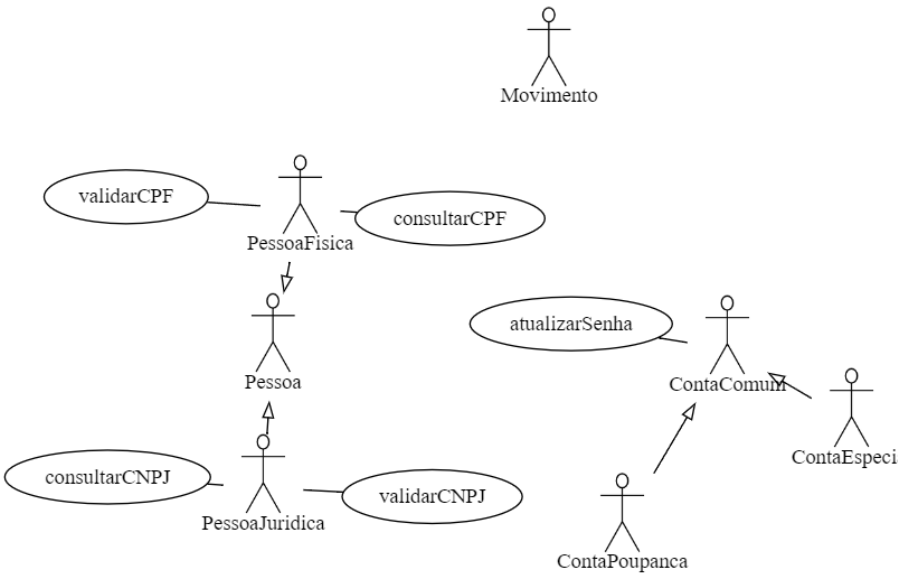

Figure 5.11: Code 2 activity answer

5.6.5 Final Survey

The final survey collected information about the experiment and feedback on the tool. The first question allowed participants to submit their answers to the experiment's questions while the others allowed the participants to give feedback on the strong and weak points of the tool.

The final survey had the following questions:

1. **Paste the code of the completed activity in this question.*** (Open Question)
2. **Were you already familiar with the 3 diagrams used?*** (Open Question)
3. **Which screen reader did you use for the experiment?** (Open Question)
4. **What are the positive aspects of the tool?** (Open Question)
5. **What are the aspects of the tool that could be improved?** (Open Question)
6. **Do you have any additional comments?** (Open Question)

5.6.6 Evaluation Results

The results were gathered from two different surveys: A background survey to provide demographic information about the participants shown in Section 5.6.2; and a final survey to input the answers for the tasks assigned and to provide feedback about their experience using `CraftPy` shown in Section 5.6.5.

Background Survey results: In the background survey, we collected information about the eight participants. Table 5.2 summarizes their details.

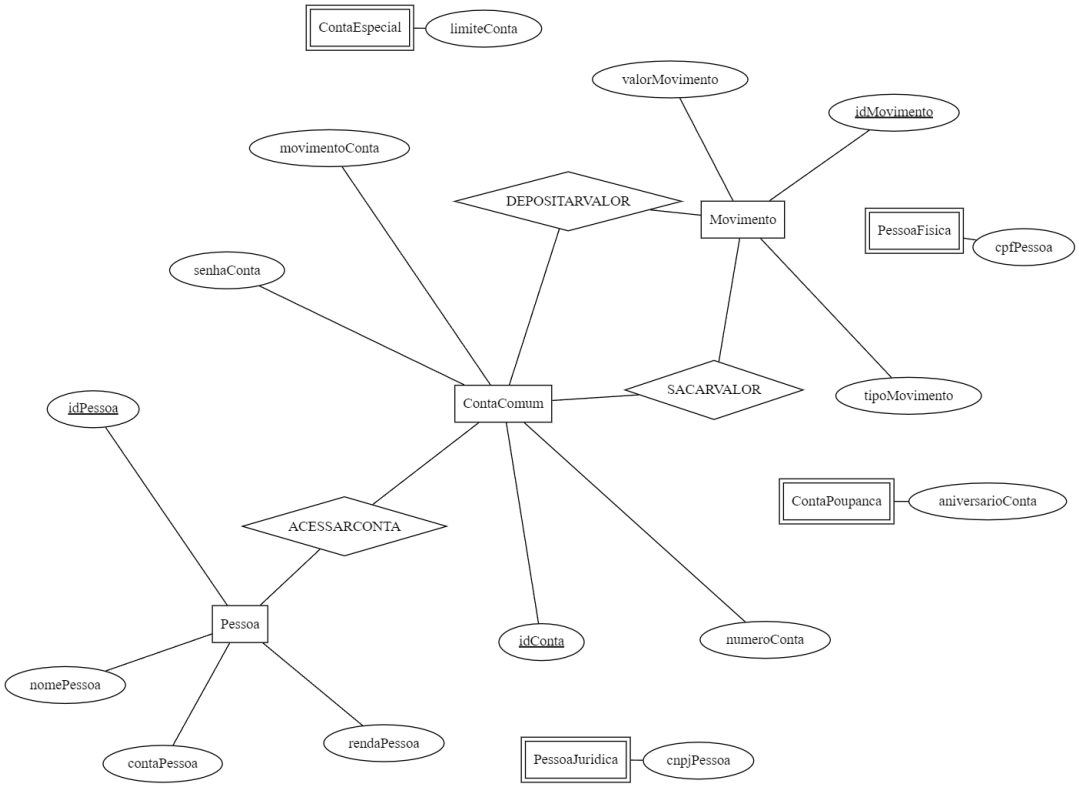

Figure 5.12: Code 3 activity answer

The ages of the participants vary between 21 and 35 years old, with most of them being under 30 years old. Five of the participants are in the 21-29 age range, and the last three are in the 30-35 age range.

Six of the eight participants (P1, P3, P4, P5, P6, and P7) already work in the ICT field, while two (P2 and P8) are not employed. The jobs found in the results were: System Analyst, IT Analyst, Quality Analyst, Software Engineer, Software Developer, and Process and Information Architect.

When asked which technology course they are currently studying or have completed, two participants (P1 and P4) selected two courses: Computer Science and Computer Engineering; and Computer Science and Software Engineering, respectively. As for the participants that selected one course: Three of them selected Information Systems, two selected Computer Science, two selected Computer Engineering, one selected Software Engineering, one selected Information Technology and one selected Internet Systems.

Out of the participants, three of them were blind (P1, P6, and P8), two of them had monocular vision (P5 and P7), one had a severe visual impairment (P4), one had a moderate visual impairment (P2) and one had multiple colobomas (P3). As the degrees of visual impairments are very different from one another, further ahead we classified them based on whether they used a screen reader to complete the tasks.

Table 5.2: Summary of the participants

#P	Age	Course	Condition	Job Occupied	Completed Tasks
P1	20-29	Comp. Science and Comp. Engineering	Blindness	Software Developer	T1;T2;T3
P2	20-29	Comp. Engineering	Moderate VI	None	T1;T2;T3
P3	20-29	Inform. Systems	Multiple Colobomas	Process and Information Architect	None
P4	30-39	Comp. Science and Soft. Engineering	Severe VI	System Analyst	T1;T2;T3
P5	30-39	Information Technology	Monocular Vision	IT Analyst	T1;T2;T3
P6	30-39	Inform. Systems	Blindness	Quality Analyst	T1;T2;T3
P7	20-29	Inform. Systems	Monocular Vision	Software Engineer	T1;T2
P8	20-29	Internet Systems	Blindness	None	T1;T2

Final Survey results. In the final survey, we asked participants to submit their answers and provide feedback. Since half of the participants were completely blind or had severe visual impairments, they used a screen reader to navigate the tool's content. Four used NVDA (P1, P6, P7, and P8), two used VoiceOver (P3 and P4), and the other two (P2 and P5) didn't use any screen reader for the experiment.

Only one out of the eight participants (P2) reported not knowing the three kinds of diagrams before completing the experiment and one (P6) claimed only to know some of them. The participant (P2) who did not know about these software modeling diagrams was able to answer all the tasks, whereas the one (P6) who only knew them partially was not able to complete the last task.

The statistics for completing the activities are as follows: Seven participants (P1, P2, P4, P5, P6, P7, and P8) completed both Task 1 and Task 2 about class and use case diagrams. However, only five participants (P1, P2, P4, P5, and P6) were able to answer Task 3 correctly which was about entity-relationship diagrams. The tasks can be found in Section 5.6.4. One participant (P3) was unable to complete the experiment due to a system problem that made them unable to complete any of the tasks. This problem was addressed after this participant had answered the surveys and reported the bug they found during the experiment.

Two blind participants (P1 and P6) who completed the tasks shared their progress via a Google Meet call. During the call, the experiment protocol and tasks were read aloud to them as they worked through the tasks independently.

We created two groups to separate the results from those who were able to complete the tasks and use a screen reader and those who did not use a screen reader. Group 1 (P1, P4, P6, P7, and P8) with those who used a screen reader, and Group 2 (P2

and P5) with those who did not use a screen reader.

All of the participants in Group 2 were able to complete all three tasks and submit a correct code in the Final Survey. Furthermore, the participants in Group 1 struggled with the third task in particular, with only three out of the five submitting the correct answer.

Feedback from the final survey was divided into two sections: one focusing on the positive aspects of the tool and the other on areas for improvement.

Positive aspects of the tool. Overall, the participants pointed out that the strong points of the tool are its ease of use for those who understand Python, its accessibility for those who use screen readers, and how the tool represents diagrams textually using code.

As the `CraftPy` tool is designed for accessibility for SVI and ease of use through Python, the positive feedback from participants aligns with our expectations for the tool and ensures it is satisfactory in those regards to the target audience. Positive comments are presented as follows.

- "Very good from a programmer's perspective, because, instead of spending time learning a new system, the person can model the classes they want to show through code, in any text editor. For those who already know the Python language, the experience is even better. I also found the interface very interesting, and very easy to use. In my opinion, even better because you don't have to install anything to start using the tool. I also found the information in the diagrams within the tool page very accessible and easy to understand." (P4)
- "Well, I really liked it, because, for those who are learning, this is a very good way of representing relationships, since UML is very visual." (P8)
- "Using type hint and dynamically assembling diagrams, which help to validate the code; generating the diagrams in text, not images, so they can be read by screen readers." (P5)
- "The ease of creating relationships and showing output in real-time." (P1)
- "The tool is accessible, allowing a better view of the screen components." (P2)

Points for improvement:

- "I wish I could able to resize the width of code screens and generated diagrams. Mainly because I use a larger font (125% zoom). In the interface, there could be an option to enable/disable line wrapping for the code, and when line wrapping is disabled, a horizontal scroll bar should be displayed just for the code. In the class diagram, the attributes only appeared after being initialized in init. They should appear just because they are declared in the class. In the Entity-relationship diagram, the classes that inherit from the superclass mistakenly don't have a relationship that their superclass has." (P5)

- "I tested your page in three different browsers (Firefox, Safari, Chrome). In all three browsers, I couldn't even finish writing the first new class because all the interface elements disappeared from it as soon as I typed '()'. Using VoiceOver, the reader tells me: 'Content is empty'. So effectively the content was no longer being rendered."[16] (P3)
- "Unfortunately, the 'Save Diagram' button didn't work." (P4)
- "Due to having enlarged components, some do not appear on the screen depending on the display used." (P2)
- "Responsiveness, when reducing the screen or accessing via cell phone it was very difficult to read in accessibility mode or move on to other topics." (P7)
- "The output. The final graph is not understandable using screen readers."[17] (P1)

The most frequently mentioned area for improvement was the tool's responsiveness. Due to the enlarged elements of the `CraftPy` tool, designed for better accessibility for low-vision users, it was not very compatible with smaller screens such as small monitors, tablets, or smartphones. Another issue, reported by one participant (P3), was caused by a system error, which was fixed immediately after the experiment to ensure the other participants could complete the experiment.

The participants' complete feedback, without translation, is available in Portuguese in Appendix A.

The responsiveness of the tool has also been improved after the feedback we received, working better even on smaller screens such as smartphone screens while keeping the accessibility to screen readers the same.

Overall, the participants found the tool easy to use and praised its accessibility and compatibility with the screen readers. This experiment is critical for refining `CraftPy` and ensuring it serves its target audience effectively. By involving users with VI in the testing process, we aim to create a more inclusive and user-friendly tool for creating diagrams.

5.7 Tool Limitations

`CraftPy` contains some limitations when it comes to diagram creation due to it being a recently developed tool. However many of them can be addressed with further development of the tool. Those limitations include: (i) **Diagram type limitation:** So far `CraftPy` can only generate three kinds of diagrams: Class Diagrams, Use Case

[16] This issue was addressed right after it was reported to allow the other participants to complete the experiment.

[17] The participant could not understand how the screen reader read the XML elements in the graph instead of the graph being unable to be read by the screen reader.

Diagrams and Entity-Relationship Diagram; and (ii) **Python language restrictions:** Due to the Python language not needing certain formalities, some restrictions come with the use of it when generating diagrams, such as specifying the capacity of an array, multiplicity in Class Diagrams are either 1:1 or 1:n.

5.8 Threats to Validity

In this section we discuss the threats to the validity of the evaluation of the tool, they are divided as follows:

- **Selection of Subjects:** As the tool requires the participants to know the Python programming language and enough knowledge about software modeling and diagrams, we only selected participants with knowledge of Python and made available material to explain to the participants the experiment what each of the diagrams represents to mitigate this threat to validity.
- **Sample Size:** The evaluation process involved eight participants, but one was unable to complete the tasks due to a tool error, which was fixed before other participants encountered the issue. As a result, only seven participants completed the experiment, making the small sample size a notable threat to validity. However, given the niche participant criteria, this number is relatively high compared to similar studies on tools for SVI.

5.9 Chapter Summary

In this chapter, we discussed the development process and preliminary evaluation of `CraftPy`. Designed to assist SVI in higher education ICT courses, `CraftPy` translates Python code into visual diagrams using screen readers. The tool's primary focus is on accessibility and responsiveness.

We conducted a preliminary evaluation through an experiment involving eight participants and three activities. Seven out of the eight participants completed most of the activities and generated the required diagrams.

Chapter 6

Conclusions

To discover the gaps that currently exist in the field of assistive technologies for SVI in ICT courses, we performed a systematic review to investigate and analyze the tools that help SVI in university-level computing courses. Through the use of search strings, inclusion, and exclusion criteria, and snowballing to filter the high amount of results in the databases, we selected 47 primary works for the systematic review. The forty-seven studies were classified based on their use, area, conference, journal, year of publishing, and availability **(RQ1)**; their evaluation methods and quantity of participants **(RQ2)**; their Technology Readiness Levels (TRL) **(RQ3)**; and their target audience **(RQ4)**. The systematic review can also be used by other teachers in the ICT field to find works to support SVI in computing.

Around 80% of the primary works found had no means to access the developed tools, showing a significant lack of availability of the software-based tools created with a research paper about them. The distribution of the tools among the computing areas was mainly focused on programming and diagramming courses, showcasing a deficit of software-based aid for other areas such as databases, networks, robotics, electronics, and 3D modeling.

The TRL metric and research about the tool availability show that many works about software tools for SVI in computing courses have tools that are in the middle levels of technology readiness but never get released to the public, or get released for a short time and due to the lack of maintenance those tools are made unavailable for their target audience.

More than 23% of the primary studies analyzed had no or near to no evaluation process in the research paper published about them and the informality of the evaluation processes used to validate those tools was a problem that we noticed during the analysis of the articles investigated.

Another issue found about the works is that the process of evaluation of these tools often leaves much to be desired, the problems range from the total lack of documented evaluation to things such as a very low quantity of participants with VI

evaluating the tools, the highly informal process of evaluation of the tool, tests done with sighted users instead of the target audience, and unclear methods of evaluation of the tool.

Only nine of the tools were available to be used, seven of them having links to their works and two of them that could be found with a quick search in search engines. The category that had the most available tools was programming and the category that had the least percentage excluding those that had none was diagramming with only one tool available out of fourteen tools presented in the primary works selected.

This work shows that the lack of tools for SVI in computing courses in higher education is not just because no researchers are working in this field, but because many of the software tools that are developed by researchers rarely are made available for the target audience to use staying only on paper or disappearing due to the lack of maintenance. It is needed to ensure the quality of the tools that are developed for SVI and to ensure that those tools are made available to them and given proper maintenance for continuous use.

In this study, we also developed the `CraftPy` tool to reduce the gap found by the systematic review. The `CraftPy` tool is a tool designed to assist SVI in higher education ICT courses in creating diagrams with an accessible interface fully compatible with screen readers. `CraftPy` is a web application that allows users to create three types of diagrams using Python code directly in the browser, eliminating the need to install software on their machines. As `CraftPy` works through Python code, it reduces the learning curve of the users to use the tool by not creating a new tool-specific programming language.

The `CraftPy` tool is versatile when it comes to the creation of diagrams for SVI, being able to create three different kinds of diagrams with ease while most of the other tools found in the systematic review were specialized in only one kind of diagram.

We conducted a preliminary experiment with eight participants with VI to evaluate the tool's effectiveness in completing a set of activities. Results showed that 87.5% of participants completed two of the three assigned tasks, with the third task having a lower success rate. Based on the feedback provided by the participants, we aim to improve `CraftPy`.

6.1 Future Work

This section discusses improvements that can contribute to the evolution of the `CraftPy` tool and further research that wasn't covered by the systematic review due to the amount of work found.

- Increasing the number of diagram types that `CraftPy` can create to increase its versatility and allow SVI to create new types of diagrams.

- Optimizing the `CraftPy` interface for better compatibility with smaller screens to allow users to access and utilize the web application through smaller screens.
- Enhancing the diagrams that already exist in the `CraftPy` tool to allow for greater versatility and the addition of more elements that weren't implemented yet.
- Further evaluation of the tool with more participants to ensure its usability to SVI with more participants to ensure that the `CraftPy` tool.
- Doing a systematic review with the gray literature or tactile tools for SVI in computer education.

References

Adebayo, E. O. and Ayorinde, I. T. (2022). Efficacy of assistive technology for improved teaching and learning in computer science. *International Journal of Education and Management Engineering*, 12(5):9–17.

Al-Ratta, N. M. and Al-Khalifa, H. S. (2013). Teaching programming for blinds: A review. In *Fourth International Conference on Information and Communication Technology and Accessibility (ICTA)*, pages 1–5.

Alves, L., Rocha, L., Pereira, C., Machado, I., Viana, W., and Junior, N. A. (2022). Estudantes com deficiência visual em computação: participação, perspectivas e desafios enfrentados. In *Anais do II Simpósio Brasileiro de Educação em Computação*, pages 67–76, Porto Alegre, RS, Brasil. SBC.

Anderson, L., Barker, B., Reid, A., Lin, K., Khalajzadeh, H., and Grundy, J. (2022). Node-read: A visually accessible low-code software development extension. In *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, MODELS '22, page 808–815, New York, NY, USA. Association for Computing Machinery.

Araújo, E. and Andrade, W. (2020). Um mapeamento sistemático sobre o ensino de programação para pessoas com deficiência. In *Anais do XXXI Simpósio Brasileiro de Informática na Educação*, pages 1713–1722, Porto Alegre, RS, Brasil. SBC.

Armaly, A., Rodeghero, P., and McMillan, C. (2018). Audiohighlight: Code skimming for blind programmers. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 206–216.

Baker, C. M., Milne, L. R., and Ladner, R. E. (2015). Structjumper: A tool to help blind programmers navigate and understand the structure of code. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, page 3043–3052, New York, NY, USA. Association for Computing Machinery.

Balik, S. P., Mealin, S. P., Stallmann, M. F., Rodman, R. D., Glatz, M. L., and Sigler, V. J. (2014). Including blind people in computing through access to graphs.

In *Proceedings of the 16th International ACM SIGACCESS Conference on Computers & Accessibility*, ASSETS '14, page 91–98, New York, NY, USA. Association for Computing Machinery.

Bergamini, R. L. (2020). Avaliação do nível de maturidade de tecnologia (trl) nas instituições de ciência e tecnologia (icts) com o modelo adaptado da afrl–air force research laboratory. *Revista de Administração de Roraima-RARR*, 10(1):1–28.

Bersch, R. (2008). Introdução à tecnologia assistiva. *Porto Alegre: CEDI*, 21.

Blenkhorn, P. and Evans, D. (1998). Using speech and touch to enable blind people to access schematic diagrams. *Journal of Network and Computer Applications*, 21(1):17–29.

Caldwell, B., Cooper, M., Reid, L. G., and Vanderheiden, G. (2007). Understanding wcag 2.0.

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

Connelly, R. (2010). Lessons and tools from teaching a blind student. *Journal of Computing Sciences in Colleges*, 25(6):34–39.

Crescenzi, P., Rossi, L., and Apollaro, G. (2012). Making turing machines accessible to blind students. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, SIGCSE '12, page 167–172, New York, NY, USA. Association for Computing Machinery.

Damasio Oliveira, J., Campos, M. d. B., and Stangherlin Machado Paixão-Cortes, V. (2020). Usable and accessible robot programming system for people who are visually impaired. In Antona, M. and Stephanidis, C., editors, *Universal Access in Human-Computer Interaction. Design Approaches and Supporting Technologies*, pages 445–464, Cham. Springer International Publishing.

Damasio Oliveira, J., de Borba Campos, M., de Morais Amory, A., and Manssour, I. H. (2017). Teaching robot programming activities for visually impaired students: A systematic review. In Antona, M. and Stephanidis, C., editors, *Universal Access in Human–Computer Interaction. Human and Technological Environments*, pages 155–167, Cham. Springer International Publishing.

de Azevedo, P., Souza, F. C., and Souza, A. (2021). B-model uma ferramenta para auxiliar estudantes com deficiência visual na modelagem de sistemas. *Revista Eletrônica de Iniciação Científica em Computação*, 19(3).

de Melo, A. C., Souza, E. P., and Lima, J. V. (2019). A tecnologia assistiva e a inclusão de educandos com deficiência visual: um mapeamento sistemático da

literatura. In *Anais do IV Congresso sobre Tecnologias na Educação*, pages 279–288, Porto Alegre, RS, Brasil. SBC.

Doherty, B. and Cheng, B. H. (2015). Uml modeling for visually-impaired persons. In *HuFaMo@ MoDELS*, pages 4–10.

Edyburn, D. L. (2000). Assistive technology and students with mild disabilities. *Focus on exceptional children*, 32(9).

Ehtesham-Ul-Haque, M., Monsur, S. M., and Billah, S. M. (2022). Grid-coding: An accessible, efficient, and structured coding paradigm for blind and low-vision programmers. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*, UIST '22, New York, NY, USA. Association for Computing Machinery.

Fowler, M. (2018). *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional.

Guedes, G. T. (2018). *UML 2-Uma abordagem prática*. Novatec Editora.

Hadwen-Bennett, A., Sentance, S., and Morrison, C. (2018). Making programming accessible to learners with visual impairments: A literature review. *International Journal of Computer Science Education in Schools*, 2(2):3–13.

Hakobyan, L., Lumsden, J., O'Sullivan, D., and Bartlett, H. (2013). Mobile assistive technologies for the visually impaired. *Survey of Ophthalmology*, 58(6):513–528.

Heuser, C. A. (2009). *Projeto de banco de dados: Volume 4 da Série Livros didáticos informática UFRGS*, volume 4. Bookman Editora.

Hooshyar, D., Ahmad, R. B., and Nasir, M. H. N. M. (2014). A framework for automatic text-to-flowchart conversion: A novel teaching aid for novice programmers. In *2014 International Conference on Computer, Control, Informatics and Its Applications (IC3INA)*, pages 7–12.

Horstmann*, M., Lorenz, M., Watkowski, A., Ioannidis, G., Herzog, O., King, A., Evans, D. G., Hagen, C., Schlieder, C., Burn, A.-M., King, N., Petrie, H., Dijkstra, S., and Crombie, D. (2004). Automated interpretation and accessible presentation of technical diagrams for blind people. *New Review of Hypermedia and Multimedia*, 10(2):141–163.

Hudec, M. and Smutny, Z. (2022). Ambient intelligence system enabling people with blindness to develop electrotechnical components and their drivers. *IEEE Access*, 10:8539–8565.

Hutchinson, J. and Metatla, O. (2018). An initial investigation into non-visual code structure overview through speech, non-speech and spearcons. In *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems*,

CHI EA '18, page 1–6, New York, NY, USA. Association for Computing Machinery.

Kane, S. K., Koushik, V., and Muehlbradt, A. (2018). Bonk: Accessible programming for accessible audio games. In *Proceedings of the 17th ACM Conference on Interaction Design and Children*, IDC '18, page 132–142, New York, NY, USA. Association for Computing Machinery.

Kennel, A. R. (1996). Audiograf: A diagram-reader for the blind. In *Proceedings of the Second Annual ACM Conference on Assistive Technologies*, Assets '96, page 51–56, New York, NY, USA. Association for Computing Machinery.

Kishore, K. V. K. and Raghunath, A. (2015). A novel e-learning framework to learn it skills for visual impaired. In *2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)*, pages 594–598.

Kitchenham, B., Charters, S., et al. (2007). *Engineering*, 45(4ve):1051.

Konecki, M. (2014). Guidl as an aiding technology in programming education of visually impaired. *J. Comput.*, 9(12):2816–2821.

Konecki, M., Lovrenčić, S., and Jervis, K. (2016). The use of assistive technology in education of programming. In *2016 ICBTS International Academic Research Conference Proceedings, Boston, USA*, pages 1–11.

Kuriakose, B., Shrestha, R., and Sandnes, F. E. (2022). Tools and technologies for blind and visually impaired navigation support: a review. *IETE Technical Review*, 39(1):3–18.

Lazar, J., Feng, J. H., and Hochheiser, H. (2017). *Research methods in human-computer interaction*. Morgan Kaufmann.

Lewis, C. (2014). Work in progress report: Nonvisual visual programming. In *Proceedings of the 25th Psychology of Programming Annual Conference (PPIG 2014)*.

Lin, Y., Wang, K., Yi, W., and Lian, S. (2019). Deep learning based wearable assistive system for visually impaired people. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*.

Loitsch, C., Müller, K., Seifermann, S., Henß, J., Krach, S., Jaworek, G., and Stiefelhagen, R. (2018). Uml4all syntax – a textual notation for uml diagrams. In Miesenberger, K. and Kouroupetroglou, G., editors, *Computers Helping People with Special Needs*, pages 598–605, Cham. Springer International Publishing.

Lopes, J., Cambeiro, J., and Amaral, V. (2018). Modelbyvoice-towards a general purpose model editor for blind people. In *MoDELS (Workshops)*, pages 762–769.

López Flores, N. U. and González Lara, A. L. (2023). Technologies in education for visually impaired people: A literature review. In Torres-Guerrero, F., Neira-Tovar, L., and Bacca-Acosta, J., editors, *2nd EAI International Conference on Smart Technology*, pages 163–169, Cham. Springer International Publishing.

Ludi, S. and Jordan, S. (2015). A jbrick: Accessible robotics programming for visually impaired users. In Antona, M. and Stephanidis, C., editors, *Universal Access in Human-Computer Interaction. Access to Learning, Health and Well-Being*, pages 157–168, Cham. Springer International Publishing.

Luque, L., Brandão, L. O., Kira, E., and Brandão, A. A. (2018). On the inclusion of learners with visual impairment in computing education programs in brazil: practices of educators and perceptions of visually impaired learners. *Journal of the Brazilian Computer Society*, 24(1):1–12.

Luque, L., Santos, C., Cruz, D. O., Brandao, L. O., and Brandao, A. (2016). Model2gether: a tool to support cooperative modeling involving blind people. In *Brazilian Conference of Software*.

Magalhães, R. L. and Neto, M. M. (2010). Aprender: Ferramenta de apoio à construção de diagrama entidade relacionamento para deficientes visuais. In *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, volume 1.

Mankins, J. C. et al. (1995). Technology readiness levels. *White Paper, April*, 6(1995):1995.

Markus, N., Juhasz, Z., Bognar, G., and Arato, A. (2008). How can java be made blind-friendly. In Miesenberger, K., Klaus, J., Zagler, W., and Karshmer, A., editors, *Computers Helping People with Special Needs*, pages 526–533, Berlin, Heidelberg. Springer Berlin Heidelberg.

Masson, M. E. J. (1983). Conceptual processing of text during skimming and rapid sequential reading. *Memory amp; Cognition*, 11:262–274.

Miyauchi, H. (2020). A systematic review on inclusive education of students with visual impairment. *Education Sciences*, 10(11).

Mohammadi, N. and Murray, I. (2013). Developing methodologies for the presentation of graphical educational material in a non-visual form for use by people with vision impairment. In *Proceedings of 2013 IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE)*, pages 373–377.

Moresi, E. et al. (2003). Metodologia da pesquisa. *Brasília: Universidade Católica de Brasília*, 108(24):5.

Mountapmbeme, A., Okafor, O., and Ludi, S. (2022a). Accessible blockly: An accessible block-based programming library for people with visual impairments. In *Proceedings of the 24th International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '22, New York, NY, USA. Association for Computing Machinery.

Mountapmbeme, A., Okafor, O., and Ludi, S. (2022b). Addressing accessibility barriers in programming for people with visual impairments: A literature review. *ACM Trans. Access. Comput.*, 15(1).

Murray, I. and Armstrong, H. (2009). Remote laboratory access for students with vision impairment. In *2009 Fifth International Conference on Networking and Services*, pages 566–571.

Nations, U. (2023). Convention on the rights of persons with disabilities and optional protocol. *Department of Economic and Social Affairs Disability*.

Obaido, G., Ade-Ibijola, A., and Vadapalli, H. (2020). Talksql: A tool for the synthesis of sql queries from verbal specifications. In *2020 2nd International Multidisciplinary Information Technology and Engineering Conference (IMITEC)*, pages 1–10.

Olechowski, A., Eppinger, S. D., and Joglekar, N. (2015). Technology readiness levels at 40: A study of state-of-the-art use, challenges, and opportunities. In *2015 Portland International Conference on Management of Engineering and Technology (PICMET)*, pages 2084–2094.

Owen, C. B., Coburn, S., and Castor, J. (2014). Teaching modern object-oriented programming to the blind: an instructor and student experience. In *2014 ASEE Annual Conference & Exposition*, pages 24–1167.

Pansanato, L. T. E., Bandeira, A. L. M., Santos, L. G. d., and Pereira, D. d. P. (2012). Projeto d4all: Acesso e manipulação de diagramas por pessoas com deficiência visual. In *Proceedings of the 11th Brazilian Symposium on Human Factors in Computing Systems*, IHC '12, page 33–36, Porto Alegre, BRA. Brazilian Computer Society.

Pender, E. C. and Healy, J. J. (2022). Accessible circuit diagrams. In *2022 33rd Irish Signals and Systems Conference (ISSC)*, pages 1–6.

Potluri, V., Vaithilingam, P., Iyengar, S., Vidya, Y., Swaminathan, M., and Srinivasa, G. (2018). Codetalk: Improving programming environment accessibility for visually impaired developers. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, page 1–11, New York, NY, USA. Association for Computing Machinery.

Reis, L. and Silva, V. (2022). deficiencia.org: Relato sobre o emprego de ferramentas computacionais enquanto tecnologias assistivas no ensino/aprendizagem para pessoas com deficiência visual. In *Anais do XXX Workshop sobre Educação em Computação*, pages 463–474, Porto Alegre, RS, Brasil. SBC.

Salih, A. E., Elsherif, M., Ali, M., Vahdati, N., Yetisen, A. K., and Butt, H. (2020). Ophthalmic wearable devices for color blindness management. *Advanced Materials Technologies*, 5(8):1901134.

Sánchez, J. and Aguayo, F. (2006). Apl: Audio programming language for blind learners. In Miesenberger, K., Klaus, J., Zagler, W. L., and Karshmer, A. I., editors, *Computers Helping People with Special Needs*, pages 1334–1341, Berlin, Heidelberg. Springer Berlin Heidelberg.

Santos, C. P., Ajala, V., and da Silva, D. (2019). Ferramenta para mediação do processo de desenvolvimento do pensamento algorítmico contemplando preceitos de acessibilidade. In *Anais do Workshop de Informática na Escola*, volume 25, pages 1124–1128.

Sassaki, R. K. (2003). Inclusão no lazer e turismo: em busca da qualidade de vida. *São Paulo: Áurea*.

Schanzer, E., Bahram, S., and Krishnamurthi, S. (2019). Accessible ast-based programming for visually-impaired programmers. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, page 773–779, New York, NY, USA. Association for Computing Machinery.

Serin, F. and Romeo, K. (2022). Drawing and understanding diagrams: An accessible approach dedicated to blind people. In Antona, M. and Stephanidis, C., editors, *Universal Access in Human-Computer Interaction. User and Context Diversity*, pages 94–109, Cham. Springer International Publishing.

Siegfried, R. M., Diakoniarakis, D., and Obianyo-Agu, U. (2004). Teaching the blind to program visually. In *Proceedings of ISECON*, volume 2005.

Smith, A. C., Cook, J. S., Francioni, J. M., Hossain, A., Anwar, M., and Rahman, M. F. (2003). Nonvisual tool for navigating hierarchical structures. In *Proceedings of the 6th International ACM SIGACCESS Conference on Computers and Accessibility*, Assets '04, page 133–139, New York, NY, USA. Association for Computing Machinery.

Smith, A. C., Francioni, J. M., and Matzek, S. D. (2000). A java programming tool for students with visual disabilities. In *Proceedings of the Fourth International ACM Conference on Assistive Technologies*, Assets '00, page 142–148, New York, NY, USA. Association for Computing Machinery.

Stefik, A., Alexander, R., Patterson, R., and Brown, J. (2007). Wad: A feasibility study using the wicked audio debugger. In *15th IEEE International Conference on Program Comprehension (ICPC '07)*, pages 69–80.

Stefik, A. M., Hundhausen, C., and Smith, D. (2011). On the design of an educational infrastructure for the blind and visually impaired in computer science. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, SIGCSE '11, page 571–576, New York, NY, USA. Association for Computing Machinery.

Svaigen, A. R. and Martimiano, L. A. F. (2018). Netanimations mobile app: Improvement of accessibility and usability to computer network learning animations. *IEEE Latin America Transactions*, 16(1):272–278.

Takai, O. K., Italiano, I. C., and Ferreira, J. E. (2005). Introdução a banco de dados. *Departamento de Ciências da Computação. Instituto de Matemática e Estatística. Universidade de São Paulo. São Paulo.*

Torres, M. J. and Barwäldt, R. (2021). Workspace awareness acessível: estratégias de sonificação para projetar interfaces colaborativas acessíveis aos cegos. In *Anais do XXXII Simpósio Brasileiro de Informática na Educação*, pages 619–629, Porto Alegre, RS, Brasil. SBC.

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

Tuttle, M. and Carter, E. W. (2022). A review of computer-assisted instruction for students with visual impairment. *The Journal of Special Education*, 56(3):132–145.

Ungar, S., Blades, M., and Spencer, C. (1993). The role of tactile maps in mobility training. *British Journal of Visual Impairment*, 11(2):59–61.

Verde, A., Alves, L., Fraga, L., and Soares, L. (2023). Py2uml: a tool for visually impaired students to build uml diagrams from python coding. In *Proceedings of the XXXVII Brazilian Symposium on Software Engineering*, SBES '23, page 491–496, New York, NY, USA. Association for Computing Machinery.

WHO et al. (2019). World report on vision.

Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, pages 1–10.

Zapirain, B. G., Zorrilla, A. M., Ruiz, I., and Muro, A. (2010). Learning electronics using image processing techniques for describing circuits to blind students. In

The 10th IEEE International Symposium on Signal Processing and Information Technology, pages 156–160.

Zen, E., Siedler, M. d. S., da Costa, V. K., and Tavares, T. A. (2022). Assistive technology to assist the visually impaired in the use of icts: A systematic literature review. In *XVIII Brazilian Symposium on Information Systems*, SBSI, New York, NY, USA. Association for Computing Machinery.

Appendix A

Final Survey Feedback

In this appendix, we present the written feedback from the eight participants, in Portuguese, matching their responses to the final survey.

What are the positive aspects of the tool?

- P1: A facilidade de criar relações e mostrar a saída em tempo real.
- P2: Ela possui acessibilidade, permitindo uma melhor visualização dos componentes da tela.
- P3: A sua ferramenta me lembra Mermaid (https://mermaid.js.org). Eu imagino que qualquer pessoa habituada com Python conseguiria desenvolver os diagramas.
- P4: Muito boa na perspectiva de um programador, pois, ao invés do tempo gasto com a aprendizagem de um novo sistema, a pessoa pode modelar as classes que deseja expor através de código, em qualquer editor de texto. Para quem já conhece a linguagem Python, a experiência é melhor ainda. Também achei muito interessante a interface, muito tranquila de utilizar. Na minha opinião, melhor ainda por não ter de instalar nada para começar a usar a ferramenta. Também achei muito acessível e fácil de entender as informações dos diagramas dentro da página da ferramenta.
- P5: Usar o type hint; montar os diagramas dinamicamente, que ajudam a validar o código; gerar os diagramas em texto, e não imagem, para poderem ser lidos por leitores de tela.
- P6: A possibilidade de auto completar.
- P7: Achei a ferramenta bem intuitiva, acredito que precisa melhorar mais, e sei que isso vai acontecer.
- P8: Bom eu gostei muito, por que, para quem ta aprendendo, isso é uma forma muito boa de representar os relacionamentos, já que a UML é muito visual.

What are the aspects of the tool that could be improved?

- P1: A saída. O gráfico final não fica compreensível usando leitores de tela.
- P2: Por ter componentes ampliados, dependendo da tela utilizada, alguns não aparecem no visor.
- P3: Testei a sua página em três navegadores diferentes (Firefox, Safari, Chrome). Nos três navegadores, não pude concluir sequer a escrita da primeira nova classe porque todos os elementos da interface sumiram da dela assim que eu digitava "()". Usando VoiceOver, o leitor me diz: "Content is empty". Então efetivamente o conteúdo não estava mais sendo renderizado.
- P4: Infelizmente, o botão de "Salvar diagrama" não funcionou...
- P5: Senti falta de poder redimensionar a largura das telas de código e diagramas/casos de usos gerados/ER. Principalmente mais porque eu uso fonte maior (zoom de 125%). Na interface, poderia ter uma opção para ativar/desativar quebra de linha para o código. E quando a quebra de linha estiver desativada, um uma barra de rolagem horizontal deveria ser exibida só para o código. No diagrama de classes, os atributos só apareceram após serem inicializados no init. Eles deveriam aparecer só pelo fato de serem declarados na classe, não? No ER, a contaPoupanca e a contaEspecial não apresentaram relacionamento com a Movimento. Mas deveria aparecer, pois herdaram da ContaComum, não?
- P6: Ok.
- P7: Responsividade, ao diminuir a tela ou acessar pelo celular foi bem difícil ler no modo de acessibilidade ou passa para outros tópicos.
- P8: Eu usei muito pouco a mesma para dizer algo sobre melhorias, de forma geral eu gostei bastante.

Do you have any additional comments?

- P1: Parabéns pelo incrível trabalho! Necessitamos iniciativas como a de vocês para sermos incluídos na sociedade da maneira correta.
- P3: Como citei acima, pessoas com deficiência visual também usam outras tecnologias assistivas que não são leitores de tela!
- P4: Muito boa a ideia e a iniciativa! Parabéns! Na minha opinião, a ferramenta pode auxiliar muito a comunicação de ideias entre os membros de uma equipe de desenvolvimento, e também em situações onde é necessário uma visualização por parte de pessoas envolvidas que não são necessariamente desenvolvedores. Pretendo testá-la onde trabalho assim que surgir a oportunidade!
- P5: Boa sorte e sucesso com seu projeto!
- P6: Não.
- P7: Parabéns pelo trabalho, espero que evolua ainda mais, fazendo isso para outras linguagens de programação como Java, C++, etc.

Appendix B

Details about the Technology Readiness Levels Classification

In this appendix, we present the questions chosen as criteria to classify a tool in a certain TRL that were present in the TRL calculator created by the Brazilian Ministry of Science, Technology, and Innovation (MCTI)[1] and the sheet in which each Technology Readiness Level was classified.

The questions chosen as criteria were the following:

TRL 1:

- Foi feita a formulação de princípios de base?
- Problema foi identificado para obter os fundamentos do pensamento como abordagem?
- Antecedentes foram identificados?
- Premissas e hipóteses básicas da tecnologia (projeto) foram determinadas?
- Foram definidas características básicas?

TRL 2:

- O planejamento do design experimental foi feito (formulação de tópicos de pesquisa e hipóteses de aplicação)?
- Levantamento de requisitos da tecnologia foi realizado?
- Foi realizado o estudo da viabilidade da tecnologia (do conceito e das aplicações)?
- Potencial usuário/consumidor e aplicação da tecnologia foram identificados?

[1] https://formularios.mctic.gov.br/index.php/117963

- Existem exemplos de aplicações (restritos a estudos analíticos com dados simulados/calculados)?

TRL 3:

- A prova de conceito inclui estudos analíticos e experimentos laboratoriais?
- Os requisitos de desempenho da tecnologia estão estabelecidos?
- As evidências ou análise detalhada sobre o conceito e/ou aplicação para apoiar suposições feitas/realizadas são suficientes?
- A função crítica da tecnologia foram validadas através de modelagem e simulação?
- As características da tecnologia foram validadas através de modelagem e simulação?

TRL 4:

- Foram feitos testes em ambiente controlado (laboratório) de cada função e/ou componente separado?
- Foram iniciados estudos em baixa fidelidade sobre a logística da tecnologia (interoperabilidade, confiabilidade, facilidade de manutenção, escalabilidade e segurança)?
- Desempenho funcional do elemento e de seus componentes foi demonstrado em ambiente laboratorial?
- Resultados dos experimentos em ambiente controlado (laboratório) mostraram que os componentes estão operacionais?
- Os dados obtidos foram validados para garantir relevância?

TRL 5:

- O Ambiente relevante foi associado ao componente/tecnologia?
- Foram aferidos efeitos de escala em relação ao modelo/protótipo?
- As funções críticas e interfaces entre componentes do elemento foram identificadas?
- Os experimentos são desenvolvidos levando-se em conta os problemas reais?
- Modelo/Protótipo da tecnologia está pronto para testes em uma condição modificada para ambiente relevante com alta precisão e fidelidade?

TRL 6:

- Os resultados dos testes do protótipo/modelo indicam viabilidade de engenharia?

- Nível de qualidade, confiabilidade e segurança foi determinado e é viável?
- Existe Relatório Técnico Final?
- Resultados de produção de P&D foram gerados?
- Problemas de produção foram identificados?

TRL 7:

- Cada elemento foi testado individualmente sob condições de estresse?
- O desempenho foi demonstrado em ambiente operacional?
- Planejamento de produção foi validado?
- Modelo/Protótipo foi demonstrado/testado em ambiente operacional real com sucesso?
- Os principais problemas de produção foram resolvidos?

TRL 8:

- A documentação da tecnologia foi concluída?
- Tecnologia qualificada por meio de teste e avaliação em ambiente/utilização real?
- Tecnologia pronta para utilização?
- Tecnologia pronta para a produção em série?
- Objetivo da tecnologia alcançado no ambiente operacional real?

TRL 9:

- Tecnologia madura?
- A tecnologia está em operação?
- Tecnologia pronta para comercialização?
- Sistema real totalmente demonstrado?
- Sistema logístico implantado/implementado?

TRL	1 - Ideação	2 - Concepção	3 - Prova de conceito	4 - Otimização	5 - Prototipagem	6 - Escalonamento	7 - Demonstração em ambiente operacional	8 - Produção	9 - Produção continuada	Total
S01										TLR 4
S02										TLR 3
S03										TLR 5
S04										TLR 4
S05										TLR 5
S06										TLR 5
S07										TLR 4
S08										TLR 5
S09										TLR 4
S10										TLR 5
S11										TLR 3
S12										TLR 4
S13										TLR 5
S14										TLR 5
S15										TLR 4
S16										TLR 5
S17										TLR 6
S18										TLR 3
S19										TLR 2
S20										TLR 4
S21										TLR 4
S22										TLR 5
S23										TLR 5
S24										TLR 3
S25										TLR 5
S26										TLR 4
S27										TLR 1
S28										TLR 4
S29										TLR 5
S30										TLR 4
S31										TLR 5
S32										TLR 4
S33										TLR 4
S34										TLR 7
S35										TLR 5
S36										TLR 2
S37										TLR 5
S38										TLR 5
S39										TLR 3
S40										TLR 3
S41										TLR 3
S42										TLR 5
S43										TLR 8
S44										TLR 5
S45										TLR 4
S46										TLR 2
S47										TLR 5