CRAFTPy: allowing people with visual impairments to create diagrams

Lucas Lopes Fraga
UEFS, Bahia, Brazil
lucasfraga884@gmail.com

Rafael Tosta Santos
IFBA, Bahia, Brazil
rafael.tosta@ifba.edu.br

Larissa Rocha
UNEB, PGCC/UEFS, Bahia, Brazil
larissabastos@uneb.br

ABSTRACT

The inclusion of students with visual impairments (SVI) in higher education has greatly advanced with the advent of new technologies. Despite these strides, challenges remain, particularly in Information and Communication Technology (ICT) courses that heavily rely on visual elements. Screen readers facilitate SVI's access to digital content, but many educational tools are still incompatible with these readers. This incompatibility is especially problematic in tools requiring interaction with visual components, such as block-based programming, diagramming, and 3D modeling tools. This study introduces CraftPy, an accessible web tool fully compatible with screen readers. CraftPy enables SVI to create various types of diagrams using Python code, employing an object-oriented approach to design classes, actors, entities, attributes, and relationships. We also conducted a preliminary evaluation involving eight SVI participants to assess the tool's effectiveness. Overall, participants found the tool to be highly accessible with screen readers and user-friendly. They were able to complete the experiment tasks with minimal difficulties. However, improvements are needed, particularly in enhancing screen responsiveness for low-vision users who depend on screen magnifiers. By developing CraftPy, we aim to promote equity in higher education, offering SVI enhanced opportunities to succeed in ICT courses. Link to the video: https://youtu.be/NXu4xbOH8Q4

KEYWORDS

UML, Python, Diagrams, Tool

1 INTRODUCTION

In its latest world report on vision, the World Health Organization (WHO) estimated that over 2.2 billion people globally experience some level of visual impairment, from mild to complete blindness. This staggering figure is expected to rise in the coming decades due to population growth and lifestyle changes [17]. As a result, the number of students with visual impairments (SVI) entering higher education is also increasing, as many countries are making concerted efforts to improve accessibility in universities for individuals with disabilities [9].

To support the learning process for SVI, various assistive technologies have been developed to assist them in higher education courses [10, 13, 19]. However, significant gaps remain, especially in visually intensive fields such as Information and Communication Technology (ICT). Tools like screen readers, such as JAWS[1] and NVDA[2], assist SVI in navigating digital content, but many programming and development tools are not compatible with these screen readers due to their reliance on graphical content and complex interfaces [6].

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

A survey by Alves et al. [2] revealed that over 40% of SVI participants reported a lack of instrumental accessibility in higher education computing courses. The primary challenges identified include the inadequacy of devices and assistive technologies designed to help SVI understand graphical interfaces [14]. This issue is particularly pronounced in ICT courses, where visual content such as diagrams plays a critical role. For instance, Unified Modeling Language (UML) diagrams, essential in software engineering courses, use graphical notations to represent software systems [5].

Additionally, the tools used to create and edit these diagrams, such as StarUML, Astah, Draw.io, and Dia, often pose significant accessibility challenges. These tools typically have complex interfaces that are not navigable by screen readers and require users to manipulate graphical elements manually to construct diagrams, making them inaccessible to blind users.

In a previous study, we proposed a prototype [16] specifically designed to be compatible with screen readers, allowing SVI to generate class diagrams using Python code. However, that tool had several limitations, including being unavailable online and creating only class diagrams with many constraints, such as neither showing multiplicity nor the variables responsible for the associations.

To address this gap, we developed 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 visually impaired individuals 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.

2 BACKGROUND AND RELATED WORK

This section provides the background for our study, discussing related tools that share a concept similar to CraftPy.

Visual impairment. Vision impairments are multiple conditions that afflict billions of people in the entire world. Affecting the visual system and its functions, visual impairments have serious consequences for individuals reducing their ability to see the world around them [17].

Adults with visual impairments experience a significant reduction in their quality of life. They are generally less represented in the workforce and have lower productivity. In addition, they tend to have higher rates of depression and anxiety compared to those with normal vision [17].

In ICT courses, visual impairments significantly hinder students due to accessibility challenges in both programming languages and environments [13]. To overcome these barriers and ensure quality education in the computing field, there has been a rise in the development of assistive technologies.

Assistive technologies for SVI in ICT Education. As outlined by Alves et al. [2], the primary challenge faced by SVI in computing courses is related to instrumental difficulties. These issues, as defined by Sassaki [14], stem from challenges in adapting materials, devices, tools, and assistive technologies to help students comprehend both the theoretical concepts and practical applications presented in the classroom.

The tools created for SVI still lack accessibility and are ineffective for students who have severe visual impairments or are completely blind. Therefore measures need to be taken, such as asserting that those tools have quality and are easy to use, and training the teachers and students about the use of assistive technologies [1, 2]. Although there are tools designed to aid visually impaired students [4, 7, 11, 12, 16, 18], many of these tools are either unavailable to the general public or were never published. Additionally, some require users to learn a new programming language or undergo extensive training.

Py2UML [16] presents a prototype of a tool that allows the creation and edition of UML class diagrams through the use of the Python programming language. Nonetheless, the prototype is not available to the general public and only creates UML class diagrams lacking some features, such as multiplicity and some types of relationships between classes.

PlantUML[3] is an open-source tool that allows users to create diagrams through coding. However, it has a learning curve because users must learn the PlantUML notations to fully utilize the tool, necessitating user training to create diagrams effectively.

The B-Model tool [4] was created to help SVI create UML diagrams through a language called BLM (Blind Modeling Language) to standardize the generation of the diagrams. The process of creating diagrams is divided into the specification of functional requirements, the interpretation of functional requirements, and the generation of diagrams. To use this tool it is needed to learn a new programming language.

TeDUB (Technical Drawings Understanding for the Blind) [7] was made to make graphical information accessible through the analysis of graphical content using text recognition and identification of diagram elements. It consists of two different modules, one that analyses the drawings and another that presents the results of said analysis. However, the tool has not been made available online.

AprenDER Magalhães and Neto [12] focuses on the creation of entity-relationship diagrams by SVI. However, the work does not make the tool available to the general public and limits the number of entities and relationships that the diagrams can support.

Luque et al. [11] presents the Model2gether tool. It promotes cooperative modeling between a sighted user and a blind user to include SVI in the process of creating diagrams. The tool implements two interfaces, one with a screen reader-compatible interface and one with a graphical interface to allow collaboration between both parties. However, the tool requires work in pairs for the creation of the diagrams.

The CRAFTPy tool is compatible with modern browsers for screen reading tools and creates three different kinds of diagrams: Class diagrams, Use Case diagrams, and Entity-Relationship diagrams in a single tool, those three types of diagrams are commonly used for the modeling of systems for both software and databases. It does not require taxing training to the users due to the use of the Python programming language for the creation of the diagrams and does not require a second user to help with their generation.

Python coding. According to the TIOBE[4] 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.

Python is also known for being a beginner-friendly programming language and suitable as a first language for novices due to it minimizing the use of keywords, being close to simple mathematical thinking, and having a built-in help module [3, 8]. In an experiment carried out by Bogdanchikov et al. [3] there was an increase of about 16% in the performance of the students in an Algorithms & Programming course after switching from JAVA to Python as the programming language used. For the reasons already listed, Python was chosen as the language used to generate the UML diagrams.

3 CRAFTPY

The CraftPy tool was designed to assist blind and visually impaired ICT students in building diagrams through the Python programming language. The tool aims primarily to ensure accessibility and reduce the learning curve for users by utilizing the Python language. Currently, 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/

3.1 Tool's Architecture

The development of our tool was inspired by a prototype presented by [16]. Their tool, which generates only class diagrams, lacks certain properties such as multiplicity and visibility markers, and is not publicly available, due to the technologies it uses. 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.

CraftPy is a web application developed using the Vite[5] development server, the React[6] library, and TypeScript[7]. 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

[3]https://plantuml.com/

[4]Avaliable on: (https://www.tiobe.com/tiobe-index/)
[5]https://vitejs.dev/
[6]https://react.dev/
[7]https://www.typescriptlang.org/

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 the RadixUI[8] component library and the TailwindCSS[9] 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)[10]. 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[11] file was generated and rendered using the d3-graphviz[12] library.

CraftPy is hosted and deployed on GitHub Pages[13]. 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 the source code is available on GitHub.

3.2 Tool's Features

The CraftPy tool has multiple features in its system. The main function is the creation of multiple types of diagrams, such as class, use case, and entity-relationship diagrams. Figure 1 shows the main interface of the system. For the UML class diagrams, using Python coding, users can define classes, attributes, methods, and establish inheritance relationships between classes. Additionally, CraftPy enables the definition of use cases, identifying different actors and detailing their interactions with the system. The tool is also effective in database modeling, allowing the creation of entity-relationship diagrams that detail entities, their attributes, and the relationships among them. Furthermore, the CraftPy interface features a top navigation bar that facilitates access to other important sections, including usage examples, help sections, and the source code available on GitHub.

These core functionalities empower users to create essential diagrams for software engineering and database courses, promoting an inclusive and accessible educational experience. CraftPy is designed to be fully compatible with modern screen readers, ensuring that all components can be navigated and read via keyboard and screen reading.

3.3 Tool's Diagrams

CraftPy is currently allowing 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.

3.3.1 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. And 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 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 Auto:
    pass

class Car(Auto):
    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()
```

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

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

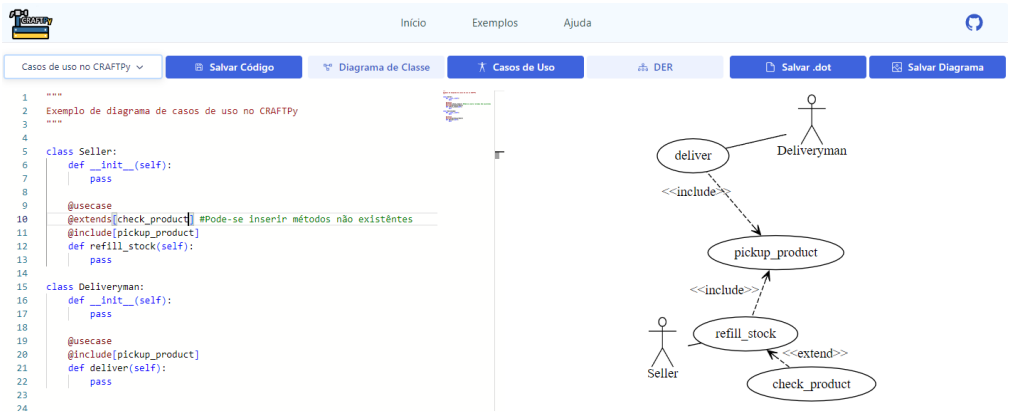

Figure 1: System Interface.

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
```

3.3.2 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
```

3.3.3 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)
```

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, it is necessary to use the @multiplicity decorator with the multiplicities of both parts separated by a colon. 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, separating the multiplicities of both parts with a colon.

```
class Order():
    pass

class Customer():
    pass

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

4 USAGE EXAMPLE

In this section, we present three different examples of the tool being used with the creation of three different kinds of diagrams. Figure 2 shows a class diagram created by using CraftPy adapting an example from Fowler [5] 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" both 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.

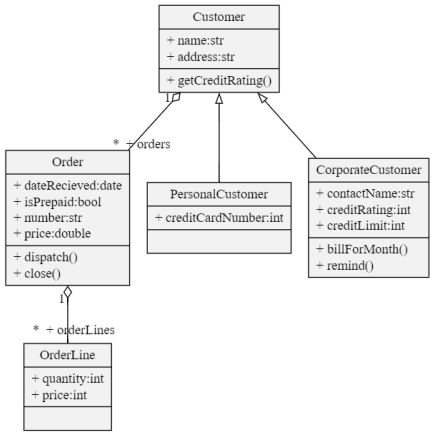

Figure 2: Class Diagram.

To the use case diagram, we adapted another example from the [5] 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".

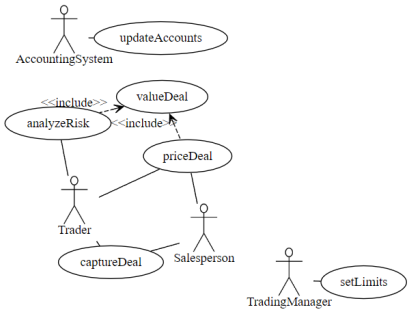

Figure 3: Use Case Diagram.

The last of the three diagrams is the entity-relationship diagram, often seen to represent databases of systems. Based on an example of the Takai et al. [15] 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 all have a 'm..n' multiplicity signaling for example that a supplier can supply multiple parts and a part can be supplied by multiple suppliers.

5 EVALUATION

We conducted a preliminary evaluation of the CraftPy tool. Thus, we sent email invitations to 45 people with visual impairments and

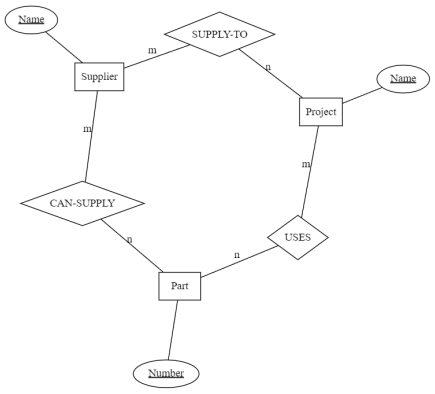

Figure 4: Entity-Relationship Diagram.

received 8 responses over a period of 15 days, even sending some reminders. The results of the participants' forms provide insights into the effectiveness of CraftPy in meeting the needs of visually impaired users. 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. All forms and evaluation documents are available in our replication package on Zenodo.

5.1 Evaluation Design

The participants were selected from a study conducted by Alves et al. [2]. 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.

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

5.2 Evaluation Results

Background Survey results. In the background survey, we collected information about the eight participants. Their age varies between 21 to 35 years old. Only one of the eight participants claimed to not know the three types of diagrams before doing the experiment. Six of the eight already work in the ICT field, while two of them are not employed.

When asked which technology course they are currently studying or have completed, two participants selected two courses (Computer Science and Computer Engineering; and Computer Science and Software Engineering). Three participants 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 totally blind, two of them had monocular vision, one had a severe visual impairment, one had a moderate visual impairment and one had multiple colobomas.

Final Survey results. In the final survey, we asked the participants to submit their answers to the questions and give their feedback. Since half of the participants who were part of the experiment were completely blind or had a severe visual impairment, they used a screen reader to understand the tool's content. Four of them used NVDA, two of them used VoiceOver and the last two didn't use any screen reader for the experiment.

The statistics for the completion of the activities are as follows: Seven participants were able to complete both Task 1 and Task 2. However, only five participants were able to answer Task 3 correctly. One participant was unable to complete the experiment due to a problem in the system that caused them to be unable to finish the activities.

Some feedbacks from the participants are presented in the following:

- "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, it's even better because you don't have to install anything to start using the tool."
- "I wish I could able to resize the width of the code screens and generated diagrams/use cases/ER. Mainly because I use a larger font (125%) zoom."

Some problems that caused system errors and were reported by the participants were addressed after the experiment. Some of those issues were compatibility with smaller screens for users that use amplifying tools and issues that caused the tool to stop responding. Overall, the participants found the tool easy to use and praised the accessibility of the tool and its compatibility with the screen readers used. This experiment is critical for refining CraftPy and ensuring it serves its target audience effectively. By involving visually impaired users in the testing process, we aim to create a more inclusive and user-friendly tool for creating diagrams.

6 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 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.

7 CONCLUSIONS

In this paper, we present CraftPy, a tool designed to assist SVI in higher education ICT courses 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. We conducted an experiment with eight visually impaired participants to evaluate the tool's effectiveness in completing a set of activities. Results showed that 87.5% of participants successfully completed two of the three assigned tasks, with the third task having a lower success rate. Future work includes: (i) increasing the number of diagrams types; (ii) optimizing the interface for better compatibility with smaller screens; and (iii) enhancing the existing diagrams to allow for greater versatility and the addition of more elements.

ARTEFACT AVAILABILITY

The supplementary material is available on Zenodo: https://doi.org/10.5281/zenodo.11432990

ACKNOWLEDGMENTS

This work was partially supported by UEFS-AUXPPG 2023 and CAPES-PROAP 2023 grants.

REFERENCES

[1] Enitan Olabisi Adebayo and Ibiyinka Temilola Ayorinde. 2022. Efficacy of assistive technology for improved teaching and learning in computer science. *International Journal of Education and Management Engineering* 12, 5 (2022), 9–17.

[2] Lais Alves, Larissa Rocha, Cláudia Pereira, Ivan Machado, Windson Viana, and Nailton Almeida Junior. 2022. Estudantes com Deficiência Visual em Computação: participação, perspectivas e desafios enfrentados. In *Anais do II Simpósio Brasileiro de Educação em Computação* (Online). SBC, Porto Alegre, RS, Brasil, 67–76. https://doi.org/10.5753/educomp.2022.19200

[3] A Bogdanchikov, M Zhaparov, and R Suliyev. 2013. Python to learn programming. *Journal of Physics: Conference Series* 423, 1 (apr 2013), 012027. https://doi.org/10.1088/1742-6596/423/1/012027

[4] Pedro de Azevedo, Francisco Carlos Souza, and Alinne Souza. 2021. B-Model Uma ferramenta para auxiliar estudantes com deficiência visual na modelagem

de sistemas. *Revista Eletrônica de Iniciação Científica em Computação* 19, 3 (set. 2021). https://sol.sbc.org.br/journals/index.php/reic/article/view/1886

[5] Martin Fowler. 2018. *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional.

[6] Alex Hadwen-Bennett, Sue Sentance, and Cecily Morrison. 2018. Making Programming Accessible to Learners with Visual Impairments: A Literature Review. *International Journal of Computer Science Education in Schools* 2, 2 (May 2018), 3–13. https://doi.org/10.21585/ijcses.v2i2.25

[7] M. Horstmann*, M. Lorenz, A. Watkowski, G. Ioannidis, O. Herzog, A. King, D. G. Evans, C. Hagen, C. Schlieder, A.-M. Burn, N. King, H. Petrie, S. Dijkstra, and D. Crombie. 2004. Automated interpretation and accessible presentation of technical diagrams for blind people. *New Review of Hypermedia and Multimedia* 10, 2 (2004), 141–163. https://doi.org/10.1080/13614560512331326017 arXiv:https://doi.org/10.1080/13614560512331326017

[8] Greg Lindstrom. 2005. Programming with python. *IT professional* 7, 5 (2005), 10–16.

[9] Ava Gibson Lise Bird Claiborne, Sue Cornforth and Alexandra Smith. 2011. Supporting students with impairments in higher education: social inclusion or cold comfort? *International Journal of Inclusive Education* 15, 5 (2011), 513–527. https://doi.org/10.1080/13603110903131747 arXiv:https://doi.org/10.1080/13603110903131747

[10] Néstor Ulises López Flores and Aída Lucina González Lara. 2023. Technologies in Education for Visually Impaired People: A Literature Review. In *2nd EAI International Conference on Smart Technology*, Francisco Torres-Guerrero, Leticia Neira-Tovar, and Jorge Bacca-Acosta (Eds.). Springer International Publishing, Cham, 163–169.

[11] Leandro Luque, CL Santos, Davi O Cruz, Leônidas O Brandao, and AA Brandao. 2016. Model2gether: a tool to support cooperative modeling involving blind people. In *Brazilian Conference of Software*.

[12] Rafael L Magalhães and Michelle MF Neto. 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)*, Vol. 1.

[13] Aboubakar Mountapmbeme, Obianuju Okafor, and Stephanie Ludi. 2022. Addressing Accessibility Barriers in Programming for People with Visual Impairments: A Literature Review. *ACM Trans. Access. Comput.* 15, 1, Article 7 (mar 2022), 26 pages. https://doi.org/10.1145/3507469

[14] Romeu Kazumi Sassaki. 2003. Inclusão no lazer e turismo: em busca da qualidade de vida. *São Paulo: Áurea* (2003).

[15] Osvaldo Kotaro Takai, Isabel Cristina Italiano, and João Eduardo Ferreira. 2005. Introdução a banco de dados. *Departamento de Ciências da Computação. Instituto de Matemática e Estatística. Universidade de São Paulo. São Paulo* (2005).

[16] Alisson Verde, Lais Alves, Lucas Fraga, and Larissa Soares. 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* (Campo Grande, Brazil) *(SBES '23)*. Association for Computing Machinery, New York, NY, USA, 491–496. https://doi.org/10.1145/3613372.3613418

[17] WHO et al. 2019. World report on vision. (2019).

[18] Ira Woodring, Charles Owen, and Samia Islam. 2024. A Method for Presenting UML Class Diagrams with Audio for Blind and Visually Impaired Students. In *Proceedings of the 17th International Conference on PErvasive Technologies Related to Assistive Environments* (Crete, Greece) *(PETRA '24)*. Association for Computing Machinery, New York, NY, USA, 15–20. https://doi.org/10.1145/3652037.3652056

[19] Eliana Zen, Marcelo da Silveira Siedler, Vinicius Kruger da Costa, and Tatiana Aires Tavares. 2022. Assistive Technology to Assist the Visually Impaired in the Use of ICTs: A Systematic Literature Review. In *XVIII Brazilian Symposium on Information Systems* (Curitiba, Brazil) *(SBSI)*. Association for Computing Machinery, New York, NY, USA, Article 18, 8 pages. https://doi.org/10.1145/3535511.3535529