

Transforming Diagrams' Semantics to Text for Visually Impaired

Charlie Cross, Deniz Cetinkaya^[0000–0002–1047–0685], and Huseyin Dogan^[0000–0002–9138–9319]

Bournemouth University, Department of Computing and Informatics
BH12 5BB, Poole, United Kingdom
<http://hci.bournemouth.ac.uk/>
{i7432987, dcetinkaya, hdogan}@bournemouth.ac.uk

Abstract. Using models and diagrams is a very useful and effective tool for representing information and systems in a graphical form to communicate and understand them better. On the other hand, graphical representations bring extra cognitive load and the process for understanding the diagrams is long and tedious in most cases for the visually impaired. To solve this problem, semantics of the diagrams should be converted to a different format that is both human and machine readable as well as communicable for the visually impaired. Most existing diagramming tools are not easily usable for the visually impaired as a tool for creating and using diagrams. In this paper, we propose an online system for defining specific diagrams and converting their semantics to text which can have a speech output for the visually impaired. We present analysis and design of this online system as well as a proof of concept prototype implementation. The prototype system provides create, save, load and transform features and tested with participants to recreate the diagrams using the automatically generated text output. Our case study showed that the results are very promising and the proposed solution can provide a way to correctly and accurately represent the information in diagrams textually.

Keywords: Assistive technology · Modelling for visually impaired · Diagrams to text.

1 Introduction

Visual impairment and sight loss are affecting millions of people's lives in the world. In the UK, there are around 2 million people living with sight loss whereas around 360,000 are registered as blind or partially sighted [13]. Recent technological improvements and research in assistive technologies can help to support these people. For example, there are a number of ways to support reading and writing such as braille, text-to-speech, making text much larger, etc.

One of the challenging tasks for the visually impaired, especially in education and business life, is working with diagrams or graphical representations. Using models and diagrams is a very useful and effective tool for representing

information and systems in a graphical form to communicate and understand them better. Diagrams are commonly used in most presentations and often make understanding a system or technology easier. Especially while teaching a new subject or introducing a new concept, diagrams, models or graphical representations are frequently used. However, visually impaired do not get the same benefits from the use of diagrams as the normal sighted do. Through research and interviews we identified that little to no solutions exist for supporting conversion of diagrams to a readable format for the visually impaired [2, 4]. Related work in the literature does not provide a practical and implemented solution so understanding the diagrams is still a difficult process for visually impaired people. So, the aim of this research is to provide the visually impaired with a system that will not only support them to easily understand diagrams but also provide a platform that is efficient and easy to use.

During our interviews, we explored that the current process is as follows: first a tactile version of the diagram is printed and then the labels are separated from the objects. Then, labels are printed out in braille as well as a separate diagram that has braille labels in the objects. This whole process is lengthy and other alternatives lack context when revealing information about the diagram.

A common tool used by the visually impaired is text-to-speech tools. There are various text-to-speech programs for example a program called NVDA [16], which allows blind people to access and interact with the Windows operating system and some other third-party applications provides such a feature. On the other hand, text-to-speech software can only read out what is available in the text and actually they cannot explain diagrams, pictures, etc. unless text is there to describe them.

To solve this issue of there being no easy way for the visually impaired to understand diagrams, we propose the idea of having an online system that converts a diagram's semantics into text. When the system converts the diagram into text, the text will be formatted in a way that explains the diagram's semantics and what it means sentence by sentence. As a quick example, for a given flowchart the text will be generated something like "This diagram is a flow chart. Object number one is labelled as 'start' and leads onto object number two".

The goal is to make the automatically generated text being as natural as possible for the user so they can fully understand each part of the diagram and they do not need to use other methods like printing out tactile versions of the diagrams. The system will be an accessible web application to create diagrams, save them in a predefined format that can be easily read by the system and converted into a text format for the visually impaired.

In this paper, we present an online system for defining specific diagrams and converting their semantics to text which can be read out for visually impaired people. We explain analysis and design of this online system as well as demonstrate a proof of concept prototype implementation. The prototype system provides create, save, load and transform features and tested with participants to recreate the diagrams using the automatically created text output. This research will have both academic and social impact. Having a tool that can be used with

little effort while working with graphical representations and diagrams would greatly help with saving time and effort of the visually impaired especially students at different levels.

The rest of the paper is organised as follows: next section provides the background information; section 3 explains the requirements analysis and design of the system; section 4 presents the implementation details and testing; finally section 5 discusses the evaluation of the study, concludes the paper and draws the future agenda.

2 Background Information

When designing and building a program with a heavy focus on user experience, it's always important to consider how we can make the application as usable and accessible as possible. Hence, during software design and development, we prioritised the usability and accessibility of the software [12].

Usability can be broken down into five different components as learnability, efficiency, memorability, errors and satisfaction [5, 15]. Using these five different components gives a clear focus on how to build a usable system. To build a successful system there are three main principles described by Nielsen: firstly having an early focus on users, then conducting empirical usability studies and finally using an iterative design process [14].

Therefore, we analysed different technologies used by the visually impaired to understand the available methods and tools as well as to identify the areas for potential improvement. During the interviews with the visually impaired, it has been recognized that one of the most common technologies used by visually impaired is text to speech tools.

Text to speech is an incredibly powerful tool that is constantly growing in terms of usability and accessibility. NVDA [16] is an application for the visually impaired to help them access and navigate their computer. On top of software that reads out everything that is on a desktop, there are now libraries such as Google's Text-to-Speech API [7] that can be integrated into websites to help read out information for users in mostly any language and in an accent, which is nearly natural.

Another commonly used technology by the visually impaired is tactile diagrams, this is done by with a heat fuser machine that uses heat to raise swell paper to add texture [8]. Hughes [9] showcases the technology and describes the main difference for visually impaired when reading diagrams through tactile is that they have to first look at all the details of the diagram to build an understanding of it.

There are many software for creating diagrams such as Microsoft Visio, Powerepoint, Lucidchart, draw.io, SimpleDiagrams, etc. Most of this software supports different types of diagramming techniques and modelling languages. Besides, they generally provide functionality to save the diagrams in different formats including text or XML. However, none of these programs are easily usable for the visually impaired as a tool for creating diagrams, that's where programs

like PlantText may help [17]. PlantText allows users to create diagrams using a strict word language, this way the visually impaired only need to write to be able to create diagrams. Figure 1 shows an example screenshot of PlantText.

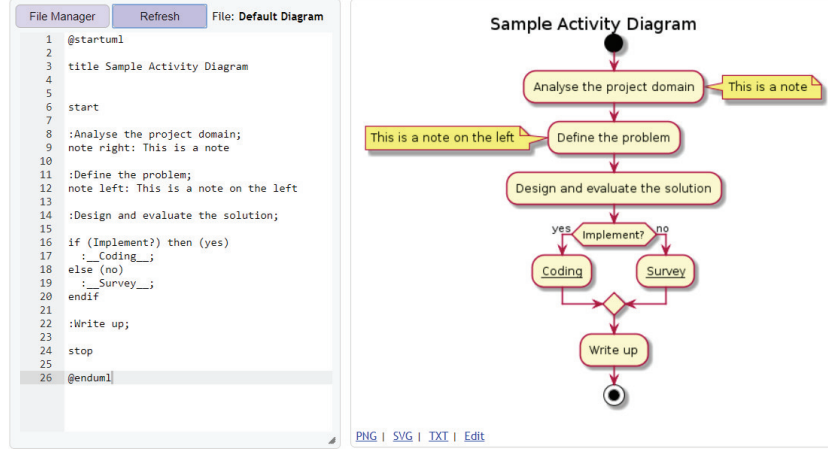


Fig. 1. PlantText example online edited [17].

In addition, refreshable braille displays are an extremely useful tool that can convert text from a digital format to braille on a mechanical output. This has been mentioned as a helpful method to easily check what has been written during the interviews.

Our research focuses on transforming diagrams' semantics to text for visually impaired people. We present an online system for defining specific diagrams and converting their semantics to text which can be read out for visually impaired people. Although related work exists in the literature, to the best of our knowledge there is no practical solution for this problem yet. Some early attempts are evident e.g. the TeDUB project to give the visually impaired a way to access diagrams [19, 10]. TeDUB was a European Union-funded project running from 2001 to 2005. It was intended to deliver a way for blind people to access technical diagrams and drawings. In addition, recent studies proposed methods to convert abstract meaning representation graphs to text [18] and raised some interesting points about how converting AMR nodes to text phrases can be far from literal due to the information in the graph and this can make generating clear sentences difficult.

3 System Analysis and Design

During the analysis of the functional requirements interview and walkthrough review methods are used [1]. We made interviews with a severely visually impaired student from our department to analyse the problem domain. During the

walkthrough reviews we made preplanned weekly meetings during the design and development stage to review the work done so far and reach a consensus [3]. MoSCoW method is used to prioritise the requirements. An agile methodology with regular meetings and an incremental software development approach was used during implementation.

3.1 Requirements

The requirements are gathered at the early stages and minor amendments are done in later stages. A prototyping method is utilised for requirements validation. The intended end users for the system are the visually impaired as well as anyone who wants to define models and/or transform them into text format with diagram's semantics. Major functional requirements are listed as follows:

- System must provide an editor to create specific type of diagrams.
- System must have a Save function to save created diagrams.
- System must have a Load function to load and edit saved diagrams.
- System must have a Generate function to generate textual output describing the diagram by converting diagrams' semantics into text.
- System must have a Save Output function to save textual output that describes the diagram.
- System must have a Check function to check if the file has the correct format.
- System must have a Save As function to convert diagrams into different formats such as XML, JSON or CSV.
- System could have integrated text-to-speech software.

Non-functional requirements are considered as well.

- System must be reliable and error free.
- System must be accessible and can be used on popular browsers.
- System must be efficient and respond quickly to actions.
- System must have a responsive design for different devices.
- System must be easy to use and navigate.
- System should conform to accessibility guidelines focusing primarily on visually impaired.

3.2 Design

Use case diagrams are used to give an understanding of the complete sequence of events from a user perspective [6]. An important consideration for the page layout is that it must accommodate the visually impaired, this means that navigation must be simplistic and easy to navigate through, avoiding elements like dropdowns, burger menus, etc. There was careful consideration when planning to avoid additional features like logins that would add extra complexity to the system with little reward for the purpose of the visually impaired. Each page was iteratively designed using whiteboard wireframes to quickly be able develop a design for the whole system that could then be implemented. When designing

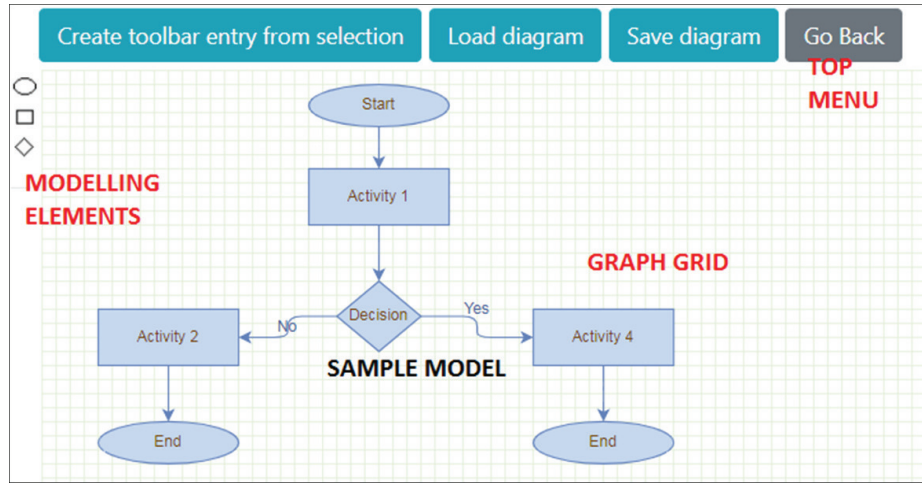


Fig. 2. Design for creating a diagram.

these pages, minimal inputs were placed so that the visually impaired can easily tab between controls in the proper order. Figure 2 shows the user interface design for creating diagrams functionality.

The designs have gone through a few iterations as new features were added and some features were moved to popup modals to make the page simpler look-out.

4 Online System for Converting Diagrams into Text

We developed an online system as a prototype with JavaScript and PHP. We used Bootstrap as CSS framework. We used an open source library MxGraph which is a client side JavaScript library for 2D diagrams [11]. Other well-known libraries are: Rappid, statejs, Draw2D, GoJS, etc. Most of these libraries have a well-developed set of tools creating, loading and saving diagrams to meet the user's needs.

We chose MXGraph because it supports most popular browsers, it is open source, it does not require any third-party plug-ins and it provides custom diagram layout. It is also easy to implement due its large range of examples provided in the download package and detailed documentation of its features. Figure 3 shows the overall system design and main file structure.

To implement the diagram container the main required libraries are called for the use of necessary functions. Changing the layout and user interface can be done through the JavaScript code by changing the styling of certain objects or through changing the linked images that are in the images folder. As seen in Figure 4, containers are defined first before the graph is initialised into one of the defined containers.

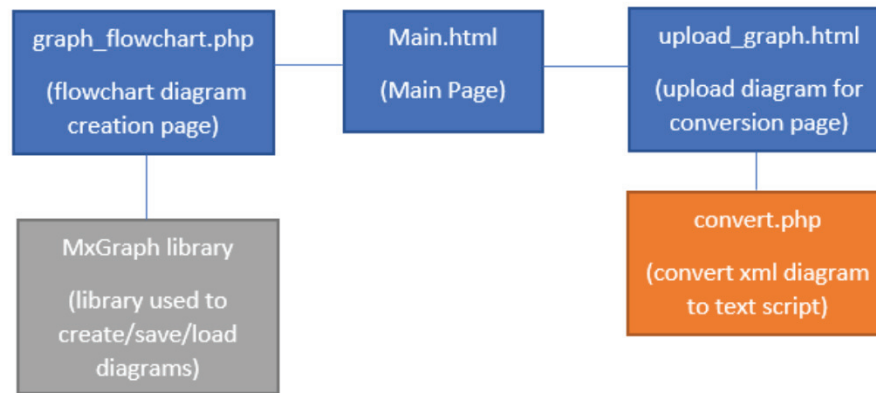


Fig. 3. System design overview.

```

// Creates the model and the graph inside the container
// using the fastest rendering available on the browser
var model = new mxGraphModel();
graph = new mxGraph(container, model);
graph.dropEnabled = true;
  
```

Fig. 4. Graph initialisation.

MxGraph has an impressive number of features that can be added and customised to a user's specific diagram, so when it came to select what features to add, only the ones that would be essential for helping create diagrams and increase ease of use were included.

Essential features included:

- Naming cells
- Deleting cells
- Creating connector lines
- Snap connectors
- Scrolling the page

Ease of use features included:

- Copying a cell
- 90 degrees connector lines
- Creating custom toolbar entries
- Tool tips

There are many other features that increase the ease of use and are essential for diagram creation, however they are included with the base graph e.g. resizing cells and selecting groups of cells. Most features that were added were completed

```
// Enables new connections in the graph
graph.setConnectable(true);
graph.setMultigraph(false);
// Enables tooltips and panning
graph.setPanning(true);
graph.setTooltips(true);
```

Fig. 5. Enabling available features.

by simply enabling them as can be seen in Figure 5. However, other features required custom code which was specific to its function.

The diagrams are saved into an XML format, which is completed by using multiple function calls that encode the data and then gets the XML from the encoded information. This information is then set as a JavaScript element which will be downloaded to the user's local storage. Generate function takes the XML file and converts it to the designed text format. The screenshot of the main page is shown in Figure 6.

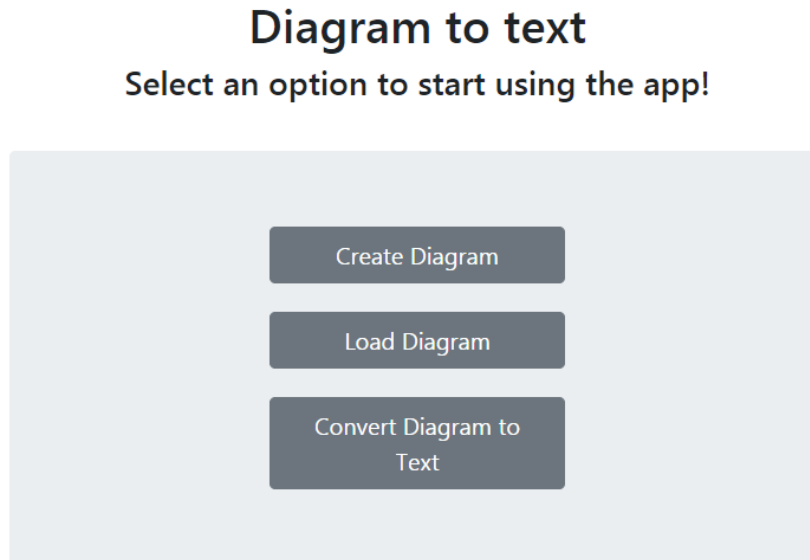


Fig. 6. Main page screenshot of the prototype.

Generating the output relies on natural flow through the diagram. This is challenging as the diagrams' semantics should be known and the algorithm should figure out the flow of the diagram. We implemented and added Flow

Chart diagrams as an example. Other diagramming techniques can be added into the system but the generate function should be defined for each technique.

For Flow Charts, the start of a flow and end of a flow can be identified easily as the beginning cell does not have line targeting it and the ending cell does not have a line coming from it. Activity cells lead onto one cell and never any more, so they can be identified easily. The difficult part however is decision cells because a decision cell can split the main flow into many paths. Each new path must properly be labelled so that no confusion happens when the text is converted into speech. Figure 7 and Figure 8 show a diagram example and associated text output respectively.

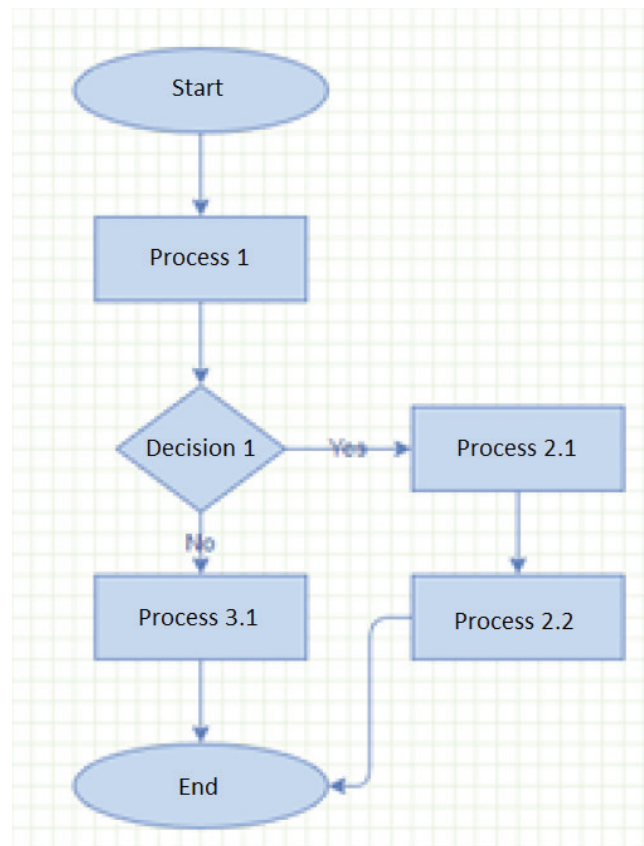
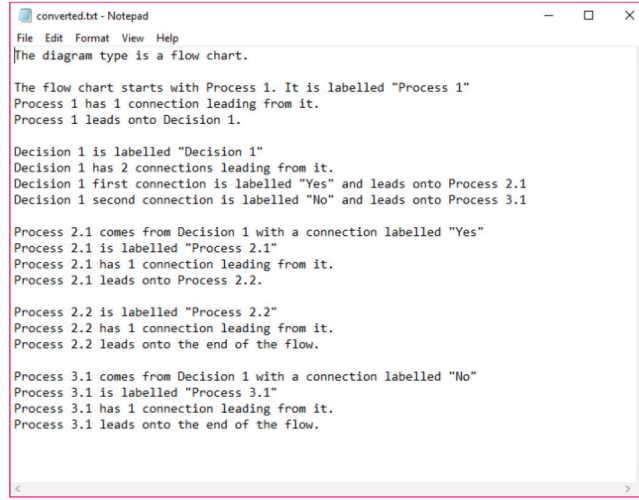


Fig. 7. A diagram example.

The information for the cells is stored in an array in the correct path order, and hence the algorithm iterates through the array and prints the related information with added custom text depending on the element type.



```

converted.txt - Notepad
File Edit Format View Help
The diagram type is a flow chart.

The flow chart starts with Process 1. It is labelled "Process 1"
Process 1 has 1 connection leading from it.
Process 1 leads onto Decision 1.

Decision 1 is labelled "Decision 1"
Decision 1 has 2 connections leading from it.
Decision 1 first connection is labelled "Yes" and leads onto Process 2.1
Decision 1 second connection is labelled "No" and leads onto Process 3.1

Process 2.1 comes from Decision 1 with a connection labelled "Yes"
Process 2.1 is labelled "Process 2.1"
Process 2.1 has 1 connection leading from it.
Process 2.1 leads onto Process 2.2.

Process 2.2 is labelled "Process 2.2"
Process 2.2 has 1 connection leading from it.
Process 2.2 leads onto the end of the flow.

Process 3.1 comes from Decision 1 with a connection labelled "No"
Process 3.1 is labelled "Process 3.1"
Process 3.1 has 1 connection leading from it.
Process 3.1 leads onto the end of the flow.

```

Fig. 8. Text output for the given example.

5 Discussion

5.1 Evaluation

We tested the system with various models and measured the accuracy of the outputs. Simple models had around 10 modelling elements while more complex ones had around 30 elements. The results of the tests are given in Table 1 and they show that the application is fault free, whereas the failed tests were marked, corrected and retested.

Table 1. System test results for black box testing.

Test result	Number of tests	Percentage
Passed	32	97%
Failed	1	3%

To evaluate the study, we designed acceptance tests and run them by five participants who are students at our university. Participants had basic information about flow chart diagrams. Participants did not see the diagrams but they are only provided the text output generated by the prototype system and this was read to the participants. The participants are then asked to recreate the diagram. Each participant attempted to recreate the same diagram separately. The test is repeated for three different diagrams with different complexity.

An average of the diagram's accuracy was calculated by comparing the participants created diagrams to the original one. For each cell or information that was not in the diagram or wrongly placed, a percentage was taken away. For

instance, if there were ten cells and the participant missed one cell, then the percentage accuracy would be 90%.

Table 2. Acceptance test results.

Diagram tested	Accuracy
Diagram 1	100%
Diagram 2	96,4%
Diagram 3	96%

Results were promising as all results were over 95% as shown in Table 2. The more complex the diagrams the less accurate the participants were when recreating the diagrams. Having a percentage loss of only 4% on the most complex diagram shows good promise for the system as the most complex diagram is not realistic and purposely complex in design.

The survey with the participants showed that they were only confused by multiple decision subpaths, however the overall feedback was very positive. Participants said that the information given for describing a diagram helped them to easily visualise the diagram.

5.2 Conclusion and Future Work

This paper presented a research study to transform diagrams' semantics to text for visually impaired people. An online system was created to define diagrams and then to convert them automatically into text as a way for the visually impaired to understand the diagrams better. This system was built as a prototype for the purpose of analysing whether a system like this could be effective and useful for the visually impaired.

The background study and literature review showed that there is work going on in other domains to help the visually impaired understand pictures, graphs, etc. However, little work has been done to give the visually impaired a way to understand domain specific diagrams like UML and flow charts.

As a future work, we would like to extend our work with adding other diagramming techniques. Currently, we are using manual integration to text-to-speech software. A fully integrated solution will increase the impact and usage of the prototype. Future development of this web application would likely find great value in supporting formats for braille and having tactile versions of diagrams, as this seems to be a popular method for the visually impaired to understand diagrams.

Acknowledgments

We would like to thank the participants and interviewees for their valuable input and comments.

References

1. Alvarez, R., Urla, J.: Tell me a good story: Using narrative analysis to examine information requirements interviews during an erp implementation. *SIGMIS Database* **33**(1), 38–52 (Feb 2002), <https://doi.org/10.1145/504350.504357>
2. Cohen, R.F., Meacham, A., Skaff, J.: Teaching graphs to visually impaired students using an active auditory interface. *SIGCSE Bulletin* **38**(1), 279–282 (2006). <https://doi.org/10.1145/1124706.1121428>, <https://doi.org/10.1145/1124706.1121428>
3. Cross, C.: Online system for creating and converting diagrams to text. Final Year Project Dissertation, Bournemouth University (2019)
4. Doherty, H., Cheng, B.: UML modeling for visually-impaired persons. In: *International Proceedings on HuFaMo@MoDELS (International Workshop on Human Factors in Modeling co-located with 18th International Conference on Model Driven Engineering Languages and Systems (MoDELS'15))*. pp. 4–10. ACM/IEEE (2015)
5. Ferré, X., Juristo, N., Windl, H., Constantine, L.: Usability basics for software developers. *Software, IEEE* **18**, 22 – 29 (02 2001). <https://doi.org/10.1109/52.903160>
6. Gemino, A., Parker, D.: Use case diagrams in support of use case modeling: Deriving understanding from the picture. *Journal of Database Management* **20**, 1–24 (01 2009)
7. Google-API: Cloud text-to-speech. Retrieved from <https://cloud.google.com/text-to-speech/>. Last accessed 30 Jan 2020.
8. Gupta, R., Balakrishnan, M., Rao, P.V.M.: Tactile diagrams for the visually impaired. *IEEE Potentials* **36**(1), 14–18 (2017). <https://doi.org/10.1109/MPOT.2016.2614754>
9. Hughes, S.: How to create tactile graphics. Royal Blind Learning Hub. Retrieved from <https://learninghub.royalblind.org/mod/page/view.php?id=370>. (2017)
10. King, A., Blenkhorn, P., Crombie, D., Dijkstra, S., Evans, G., Wood, J.: Presenting uml software engineering diagrams to blind people. In: Miesenberger, K., Klaus, J., Zagler, W.L., Burger, D. (eds.) *Computers Helping People with Special Needs*. pp. 522–529. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
11. MxGraph: mxgraph 4.0.0. website and tutorial. Retrieved from <https://jgraph.github.io/mxgraph/>. Last accessed 2 Jan 2020.
12. Nganji, J., Nggada, S.: Disability-aware software engineering for improved system accessibility and usability. *International Journal of Software Engineering and Its Applications* **5**, 47–62 (07 2011)
13. NHS: Blindness and vision loss. Retrieved from NHS Website: <https://www.nhs.uk/conditions/vision-loss/>. Last accessed 2 Jan 2020.
14. Nielsen, J.: 25 years in usability. Nielsen Norman Group (2008)
15. Nielsen, J.: Usability 101: Introduction to usability. Nielsen Norman Group (2012)
16. NV-Access: Homepage. Retrieved from nvaccess: <https://www.nvaccess.org/>. Last accessed 2 Jan 2020.
17. PlantText: Uml editor - an online tool that generates images from text. Retrieved from <https://www.planttext.com/>. Last accessed 2 Jan 2020.
18. Song, L., Zhang, Y., Wang, Z., Gildea, D.: A graph-to-sequence model for amr-to-text generation. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*. pp. 1616–1626 (01 2018). <https://doi.org/10.18653/v1/P18-1150>
19. TeDUB-project: Technical drawings understanding for the blind. Retrieved from <https://cordis.europa.eu/project/rcn/60750/factsheet/en>. Last accessed 30 Jan 2020.