1. Problems and barriers a Students who are visually impaired has:

* **Blocks4All** <https://dl.acm.org/citation.cfm?id=3051525>
  1. Nature of Block language: (B4all)
     1. You really have to see the shape in order to match.
     2. Inaccessible to screen readers.
     3. Tactile and digital block programming are visual.
  2. Drag-n-drop is also not feasible, similar as Block programming (B4all)
* **Digital Systems Lab for VIS** <https://helvia.uco.es/xmlui/handle/10396/16554>
  1. CAD – computer aided design (circuits, OS…) (Digital Systems Lab)
     1. This is visual, and this leads to inaccessibility of labs.
  2. VLE – visual learning environment: useful for adapting practices (Digital Systems Lab)
     1. However, current plat form is not accessible to VISs.
  3. TUI – Tactile User Interfaces and AUI – Audio User Interfaces: Touch and Hearing (Digital Systems Lab)
     1. This leads to the problems that those images/circuits/graphs that are not accessible to AUIs and TUIs (e.g. you cannot touch a picture, and touching a voltage circuits might cause dangers).
  4. Incapable of plugging the cable into correct sockets properly. (Digital Systems Lab)
  5. Trouble with debugging (Digital Systems Lab)
     1. Wrong insertions of cables – inefficiency
* **StructJumper** <https://dl.acm.org/citation.cfm?id=2702589>
  1. Reading code (StructJumper)
     1. Skim a big chunk of code – Screen reader, but can we do better?
  2. Visual Aids (StructJumper)
     1. Colours in IDE, such as Eclipse/Intellij/Netbeans
     2. This lead to the problem of debugging, but this time it is more about the code.
     3. Yes, we can use screen reader, but what if we have a big chunk of code, say 500 lines?
  3. Underused of tools of an IDE, go underused by blind programmers (StructJumper) pp.3044
  4. Overview of the code (StructJumper) pp.3044
     1. Code structure – can be read by screen reader, but what about a general structure?
     2. Therefore they rely on API docs and header files – but what if code not well documented?
* **Blind Learners Programming Through Audio** <https://dl.acm.org/citation.cfm?id=1057018>
  1. Programming Language Problem
     1. These commands must be correctly written and well defined
     2. Most of these programming languages are heavily based on visual interfaces.
     3. Two major problems:
        1. The issue of verifying the program consistency and the correct reading of command lines. — Pure language
        2. If we provide current tools to support program construction (audio I think?), they deal with graphical user interfaces.
* A Comparison of Program Comprehension Strategies by Blind and Sighted Programmers <https://ieeexplore.ieee.org/document/7987041>
  1. Syntax
     1. This line-by-line strategy is feasible, but imposes a high cognitive load on the programmer—syntax details such as whitespace and block braces are often very difficult to follow,
     2. The result is that programmers who are blind have utterly different mechanisms for reading source code as compared to their sighted counterparts. — Our results suggest that blind programmers read and comprehend code much like their sighted counterparts.
     3. Certain programming languages and coding styles are less suited for introducing blind students to programming because the syntax creates an otherwise-avoidable distrac- tion. C-family languages such as Java make extensive use of parentheses and semicolons which are always spoken by the screen reader and sound unnatural to the blind novice.
  2. Tools designed for sighted and blind people are different
     1. If blind programmers read different areas of code than sighted programmers, then tools designed for sighted programmers might not be sufficient for the blind community.
  3. Lack of studies of comparing program comprehension by blind programmers to comprehension by sighted programmers.
* Eliciting Programming Challenges Faced by Developers with Visual Impairments: Exploratory Study <https://ieeexplore.ieee.org/document/7809491>
  1. Tools in IDEs are underused by blind programmers
     + 1. Not clear if these people don’t know or tools are unaccessible to them.

1. What are ways that blind programmers use for navigating spatial interface?

* Blocks4All <https://dl.acm.org/citation.cfm?id=3051525>
  1. Non-visual techniques
     1. Blocky — allowance for touch interactions
     2. audio — speech and sound
  2. Techniques that allow children to easily create or move the blocks
     1. Touchscreen — similar to A.
     2. replace drag and drop with “select select drop”
     3. using sound to place blocks together
  3. Techniques that allow children to create rich non-visual programs that are fully accessible
     1. Interactions for understanding the structure of the program and the state of the program
* Digital for VIS <https://pdfs.semanticscholar.org/cc84c6f10ac5cf5c73918f0ff002fff28426f532.pdf>
  1. Touch is best for orientations in schematics
  2. Real devices
     1. Sometimes pins are too close to each other
  3. Embossed cardboard of the chip
     1. A cardboard with the design of every chip
  4. A human being like TA

1. What are ways that blind programmers use for navigating text-based interface?

* Digital for VIS <https://pdfs.semanticscholar.org/cc84/c6f10ac5cf5c73918f0ff002fff28426f532.pdf>
  1. Interfaces that are non-visual
     1. TUIs — Tactile User Interfaces
     2. AUIs — Audio User Interfaces
     3. Other audio interfaces
  2. Translation to Braille
     1. Note that in Spain, blind students taking comp org is unprecedented (according to the author, it has never happened)
     2. Braill4e is used for both blind and sighted students
* StructJumper <https://dl.acm.org/citation.cfm?id=2702589>
  1. Smith et al. created a tool to allow blind programmers to navigate the hierarchical structure of the program in Eclipse.
  2. StructJumper is expanded on this tool mentioned above.
     1. Based on a hierarchical tree of the nesting structure of a program
  3. Audio based programming
     1. Sodbeans created by Steiff et al. pp.3044
        1. How it works

It relies on audio cues to convey information

Sodbeans’ auditory cues are built on three principles (They are also applied to Structjumper)

* + - 1. Advantages

They are short (cues)

Browsable

Important information comes first

It does include audio debugging ability

* + 1. Similar tools including Microsoft Visual Audio IDE focusing on debugging
       1. Sonification
    2. Eclipse plugin wrote by Smith et al.
  1. Navigation Aids for Screen Readers
     1. Minimal research on this
     2. Some are able to read webpages
     3. Many can skip lines
  2. Code navigation
     1. Make more information visible like providing them fisheye view of the code
     2. Help programmers navigate quicker
     3. Problems
        1. Not a lot of researches on this
        2. They do not help programmers navigate within the code
* A Comparison of Program Comprehension Strategies by Blind and Sighted Programmers <https://ieeexplore.ieee.org/document/7987041>
  1. Two classic theories for blind programmers to understand code structures.
     1. Top down: People form a new hypothesis about the structure when they encounter a different line
     2. Bottom up: They divide codes into different chunks which are small lines that are easy to be understood.
     3. People do not completely understand of the large code, but rather to familiarize how specific concepts are implemented.

This often happens when a software is being developed, and any understanding could become invalidated in weeks.

* 1. Categories of tools that help blind programmers understand code
     1. specialized programming languages or environments
     2. specialized audio or tactile representations of programming “space”
     3. audio or tactile means of improving computing accessibility generally
  2. Connection between spatial and textual code navigation
     1. other researchers have noted that while reading code, programmers form mental models of the code which can be treated as spatial information <https://cs.gmu.edu/~tlatoza/papers/icse2006.pdf>
     2. This spatial information can be represented and navigated non-visually [29], [30], [31], for example, via sounds which represent different window action connec- tions in Visual Basic [31] or a tree view that represents the hierarchical structure of a source file [32]
     3. [29]: <https://www.sciencedirect.com/science/article/pii/S1084804598900601>
     4. [30]: <https://dl.acm.org/citation.cfm?id=2502047>
     5. [31]: <https://dl.acm.org/citation.cfm?id=1169035>
     6. [32]: <https://dl.acm.org/citation.cfm?id=2702589> (StructJumper)
     7. Above are the links that this paper has cited
* Designing a Scripting Language to Help the Blind Program Visually <https://dl.acm.org/citation.cfm?id=1169035>
  1. Visual Basics
     + Form: Text format
       - This might be time consuming and hard for both sighted and blind programmers to create a form by writing such specifications without constantly checking to see if the form looks the way that the creator intended it to look.
     + GUI development tools: container class (VB or JFrames)
       - Objects are placed sequentially
       - The size options are limited to small, medium or large widths and heights specified by the number of lines of text
       - TBC

1. How do you make drag-n-drop interfaces more accessible

Since I have already provided tools that are useful for blind programmers in question 2 and 3, here I am going to reiterate those ways and conclude the advantages and drawbacks of them.

* 1. Some non-visual techniques that allow programmers or children to conduct draw-n-drop moves (B4All) <https://dl.acm.org/citation.cfm?id=3051525>
  + pros:
    - Non-visual means more accessible to VIS
    - More direct and easy to manipulate
    - Good for block programming
  + cons:
    - Scope is very narrow: it works well with block programming, but what about text-based programming?
    - This is too slow: this might work fine with a small chunk of code but what about a huge chunk of code? Inefficient
    - Applicability: this might work fine for children or people without any experience, but might be suitable for higher education.
  1. Touch Screen (embossed cardboard of the chip) <https://pdfs.semanticscholar.org/cc84/c6f10ac5cf5c73918f0ff002fff28426f532.pdf>
  + pros:
    - Accessible to VIS when it comes to circuits and block programming
  + cons:
    - Inefficiency: when it comes to large and complex structure, touch screen might not work very well
    - Scope of use: the scope of use is narrow because touch screen might not be useful when it comes to text-based programming. When we have large chunks of code, touching might not help students learn the general structure well

1. Real device/human help like TAs <https://pdfs.semanticscholar.org/cc84/c6f10ac5cf5c73918f0ff002fff28426f532.pdf>
   * pros:
     + Interesting
     + Programmers will have a better understanding with real device like circuits and quick help with TAs
   * cons:
     + Inefficiency: large and complex circuits may take students more time to understand the structures of the circuits
     + Scope of applicability: it might work for block, drag-n-drop, or circuit design, but it might not work well with text-base design
     + Safety: working with real device might result in dangerous acts. A blind student might plug the wrong cable into wrong outlets thus resulting in electrical shock. (In the paper, however, the author states that security will be ensured since the device has been proved to be safe)
2. Interfaces like TUI/AUI and screenreader <https://dl.acm.org/citation.cfm?id=2702589>
   * pros:
     + It helps with text-based programming
     + When implemented appropriately, this could be used efficiently to skip some lines.
   * cons:
     + Scope: unlike previous tools, this could only be used for text-based programming
     + Inefficiency: When a programmer is facing a large chunk of code, it could get inefficient if the screen reader reads every line.
     + Understanding of the code: Screenreader can read every line of the code, but it might not help programmer with the general structure of the code.
3. Code navigation/other navigation aids <https://dl.acm.org/citation.cfm?id=2702589>
   * pros:
     + This makes navigation an easier task by helping programmers navigate their code quicker
     + They are also efficient since they can skip lines
   * cons:
     + Applicability: there are not a lot of researches on these navigation tools
     + Understanding code structure: They do not help programmers navigate within the code. (This is mentioned in StructJumper for code navigation tools, but I do not understand why)
4. What new models do?
   1. StructJumper - hierarchical trees that gives programmers choice to go to a specific command step by step
   2. Audio Programming Language
   * Models
     + APL has two main layers: Audio Interface and Programming Logic — AI & PL
       - AI:
         * Circular Command — CC:

A command list to be chosen according to the program state such as cycle, condition, input, output, and variables

Query is used to define variable names and values, and input/output audio. Text is optional.

* + - * PL:
        + Run program,
        + End loop or condition,
        + Delete last command, and
        + Save command and Verify the next step.
  + Development
    - Java 1.4.1 and FreeTTS Java speech synthesizer.
    - Modules: DataBase, Integrity, Kernel, and CHI
    - APL has a circular command list introduced to the user to select the desired command.
    - Command navigation and command semantic — CN & CS
      * CN: A reduced command list to improve the navigation through the list and to optimize the programming time.
  + Drawback of Audio Programming Language
    - The Audio Programming Language has been shown to be an effective teaching tool, but is not intended as a production language. The lack of a visual representation makes it difficult to interface with sighted programmers
    - From <https://ieeexplore.ieee.org/document/7987041>

1. Sodbeans: <https://dl.acm.org/citation.cfm?id=1953323>
   * + - Screen Reader:
         * Similar to screen reader but designed to be compatible with programming languages.

“Push” accessibility model.

With the help of APIs — but those APIs are often poorly maintained

* + Components:
    - custom virtual machine
    - compiler
    - debugger
  + Auditory cues
    - “repeat” over “for” or “while”
    - Auditory cues has been strictly analyzed
    - Design:
      * short
      * “browsable”: like the third
      * most important info comes first
  1. ***GFK: Graph Sketching: its general idea is to create an interface where users can have computational equivalence.*** 
     + - ***Connection view***

***Adding, editing, and removing nodes:***

***Creation: double-clicking on an empty space on the graph panel***

***Edition: ouble- clicking on the node or by selecting the node and pressing Enter, both of which bring up the Node Properties dialog box.***

***Removal: selecting it and pressing the Delete key, selecting Remove from the Graph menu, or pushing the Remove button on the toolbar.***

***Whenever a node is removed, its edges are removed as well.***

***Adding, editing, and removing edges:***

***Creation: pushing the Add Edge button on the toolbar, selecting Add Edge... from the Graph menu, or entering CTRL+E.***

***Edition: double- clicking on the edge or by selecting the edge and pressing Enter, both of which bring up the Edge Properties dialog box.***

***Removal: selecting it and pressing the Delete key, selecting Remove from the Graph menu, or pushing the Remove button on the toolbar.***

***Graph navigation***

***How to select a node?***

***by clicking on it,***

***selecting Jump to Node... from the Graph menu,***

***or by entering CTRL+J.***

***The latter two bring up Jump to Node dialog***

***How to select edges?***

***pressing up arrow when an edge is selected, moves focus to (selects) the opposite endpoint; pressing Escape returns focus back to the original endpoint.***

***How to know status?***

***information about the selected node/edge is displayed in the status bar and announced by the screen reader, if one is in use.***

* + - * ***Grid view: rovides blind (and other) users with the ability to spatially lay out a graph***

***Adding editing and removing nodes:***

***Creation: double-clicking on an empty square or by pressing Enter when focus is on an empty square.***

***Edition: double- clicking on the square containing the node or by pressing Enter when focus is on the node’s square.***

***Move: changing its X and/or Y Location via the Node Properties dialog or by cutting and pasting it in a different square using the standard CTRL+X, CTRL+V sequence.***

***Removal: selecting it and pressing the Delete key, selecting Remove from the Graph menu, or pushing the Remove button on the toolbar.***

***Whenever a node is removed, its edges are removed as well.***

***Adding editing and removing edges:***

***Same as using connecting view***

***Editing and removing needs connection view***