Theoretical basis for code presentation: A case for cognitive load

NYAH SPEICHER and PRASHANT CHANDRASEKAR, University of Mary Washington, USA

Evidence supports that reducing cognitive load (CL) improves task performance for people of all abilities. This effect is specifically important for blind-and-low-vision (BLV) individuals because they cannot rely on many common methods of managing CL, which are frequently vision-based techniques. Current accessible "solutions" for BLV developers only sporadically consider CL in their design. There isn't a way to know whether CL is being alleviated by them. Neither do we know if alleviating CL is part of the mechanism behind why these solutions help BLV people. Using a strong foundation in psychological sciences, we identify aspects of CL that impact performance and learning in programming. These aspects are then examined when evaluating existing solutions for programming sub-tasks for BLV users. We propose an initial design "recommendations" for presentation of code which, when followed, will reduce cognitive load for BLV developers.

CCS Concepts: • **Human-centered computing → HCI theory, concepts and models**; **Accessibility theory, concepts and paradigms**; • **Applied computing → Psychology**.

ACM Reference Format:
Nyah Speicher and Prashant Chandrasekar. 2025. Theoretical basis for code presentation: A case for cognitive load. 1, 1 (November 2025), 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Writing computer code is a creative expression of problem solving. Significant development effort for projects in software, web development, machine learning, among others requires an integration of existing libraries or code repositories. Solutions are naturally built upon one another, re-packaged, extended, re-used, and creatively re-organized to solve different problems. This creative expression results in scenarios involving unnecessary challenges in programming tasks (such as code comprehension, editing, navigation, among others) for current and aspiring blind-and-low-vision (BLV) developers [34]. We believe that one of the underlying aspects that is core to many of the challenges (as studied extensively in literature) for BLV developers is the impact on cognitive load (CL).

For instance, one of the reasons that code debugging is challenging for BLV developers is because variable "watch windows" are inaccessible. As a result, BLV developers have to use their memory or external processes to keep track of dynamic program behavior and variable states. Similarly, BLV developers need to maintain an active memory of multi-level-nested code structures, cursor position, line location, among others, all of which put a strain on their memory, thereby impacting their task performance.

The main challenge is that the broader developer community is not incentivized to write code in "a manner" that might lower the cognitive load for the BLV community. This makes it continuously challenging for BLV developers to build feature-rich and well integrated solutions. Even if we could incentivize them, defining such coding "principles" or

Authors' Contact Information: Nyah Speicher, nspeiche@mail.umw.edu; Prashant Chandrasekar, pchandra@umw.edu, University of Mary Washington, Fredericksburg, Virginia, USA.

Manuscript submitted to ACM

"rubrics" that conceptually and practically lower cognitive load is a non-trivial task and needs to based on evaluating the application of psychology-grounded theories. If defined, these "theoretically-provable" considerations can be the foundation for design of future solutions that address accessibility barriers while programming.

1.1 Research Goal and Research Questions

The goal of this work is to examine the role that cognitive load plays in the design of various solutions that address accessibility barriers while programming. We,

- Identify and discuss the role/importance of CL in design of artifacts/solutions meant to support BLV individuals for *non-programming-related tasks*.
- Identify and **contrast** that against the role of CL in the design of artifacts or solutions meant to support BLV individuals for *programming-related* tasks.
- Thereby, identify gaps in current research on designing accessible solutions keeping CL as a criteria.
- Propose rules for code structure which are composed with consideration for CL and that have a significant impact on performance of downstream programming tasks for BLV individuals.

To achieve the goals, we formulated the following research questions:

- **RQ1:** How does cognitive load (CL) factor into the design of solutions meant to support BLV individuals?
- **RQ2:** What existing solutions/approaches meant to support programming tasks (in-)directly consider CL in their designs?
- **RQ3:** In what ways do solutions/designs for programming tasks which consider CL improve upon those that don't consider CL?

We first provide a background on aspects of cognitive load theory and the emphasis that psychologists put on its impact on task performance (Sections 2 and 3). We provide our review of the impact of cognitive load in the design of solutions meant to support BLV individuals for non-programming tasks (Section 5.1). We conduct an exhaustive evaluation of all existing solutions meant to support BLV developers in their programming tasks (Section 5.2). Specifically, we inspect the rationale behind the solution designs and check for basis in cognitive load theory. Using our expertise in Psychology, CS-Education, and Accessibility in Computing, we synthesize our findings and propose the creation of a "virtual code view" that is a projection of any code, refactored, to reduce the burden of cognitive load (Section 6).

2 Cognitive Load and Working Memory

Cognitive load (CL) encompasses the demand placed on an individual's mental resources by a particular task. It is a term to describe how those resources are stretched in order to achieve a result [11, 55]. Understanding the role of CL is important because it is associated with poorer performance on measures of learning and task execution [23, 55].

In order to understand CL, one must understand working memory, because the two concepts are inextricably linked. Working memory is a part of the short-term memory system. It serves as a sort of "workbench" for information that is immediately relevant to the task at hand. While information is in working memory, it can be manipulated and used to make sense of processes such as learning, comprehension, or reasoning [1, 10]. Information generally moves in and out of working memory quickly unless it is consciously maintained by the individual. Even with conscious maintenance, information in working memory is fleeting because the brain is driven by novel stimuli [9, 37]. Human attention is easily stolen away from one focus by a new object of interest in the environment. In this case, the information previously contained in working memory is quickly abandoned to make room for new information.

Importantly, working memory only has a limited capacity. Recent studies place this between just three and five "chunks" of information at a time [1, 10]. For this reason, it is a precious resource which the human brain has to manage deliberately. When participating in a goal-directed activity, there is a careful balance between allocating working memory resources to productive processes and being cautious not to overload the working memory and cause all of the individual's mental resources to be depleted. It has been demonstrated that overloading the brain's cognitive resources, aptly named **cognitive overload**, is associated with **poorer performance** and **learning outcomes** [23, 55]. Alternatively, there is evidence that maintaining a balance of the working memory and other cognitive resources contributes to improved performance [11].

A great example of how high CL impedes to processes of learning and task performance comes from a study in which the authors looked at the impact of different CL levels on the ability to evaluate and adjust to error feedback [23]. The procedure involved participants performing a time estimation task, and then receiving feedback about their performance on the task. Using that feedback, they would then attempt to improve their accuracy on the next trial. The authors found that participants were able to make more improvement between trials in the low CL condition compared to the low CL condition. This indicates that they were less effective at using error feedback to make adjustments in the high CL condition. In other words, they weren't able to learn as well from feedback during past trials. Although it only only looks at one type of learning, this study demonstrates how CL plays a key role in learning, memory, and performance.

3 Types of Cognitive Load

Cognitive load is made up of not one, but three interdependent components. Each of these types of CL contributes to the overall demand a person experiences, but they come from unique sources. It is important to understand the mechanisms behind these CL elements because they play different roles in the greater impact of CL. For example, some elements of CL can be modified by external means, while others are fixed entities. Knowing what part each of them plays in the context of learning and task performance enables them to be addressed effectively.

3.1 Intrinsic Cognitive Load

Intrinsic cognitive load (ICL) is the first of the three CL components. It comprises the mental effort expended by a person due to the difficulty or complexity of the task at hand [5, 16]. ICL cannot be modified by outside influences because it is inherent to the material. However, it is moderated by the expertise of the individual. A higher level of expertise comes with more experience and better strategies for processing information which don't rely as heavily on the working memory system. I.e., experts are better at coping with complex material efficiently, so they don't experience the same volume of ICL.

3.2 Extraneous Cognitive Load

Extraneous cognitive load (ECL) makes up the second portion of CL. It describes the excess cognitive resources devoted to processing information indirectly related to the task goal (e.g., distracting stimuli, irrelevant instructions, confusing wording) [5, 16]. ECL can be problematic because these distracting stimuli, which do not contribute to the task goals, draw attention and resources away from productive learning processes. **This is why, in both learning and task performance, minimizing ECL is associated with better outcomes** [5]. Reducing ECL redirects mental resources towards productive, goal-directed processes. ECL is important in the realm of design because it is the **element of CL which can most easily be externally modified**. This is particularly true when ICL is high because these components

are additive. High ICL leaves few resources to spare, so designs which minimize ECL facilitate the efficient use of cognitive resources [5, 16].

3.3 Germane Cognitive Load

Germane cognitive load (GCL) is the final element of CL. While it is often the most difficult to conceptualize, it is also the most important. It is easiest to think about GCL as the mental effort directed towards goal-related information processing. This can involve making connections between old and new knowledge as well as applying learning strategies to understand information [5, 16]. In a way, GCL can be described as "good" CL. Although it is still ideal to keep CL low, if a large portion of the general load is made up of GCL, then those cognitive resources are being utilized productively. High GCL can be an indication that the individual is using their mental effort towards learning strategies, attention, and memory, rather than distracting stimuli as in ECL [11]. This type of load can be externally influenced to a limited extent by teaching learning strategies and schema construction, but the effect is not as substantial as that of focusing on ECL [5].

4 Cognitive Offloading

Cognitive offloading is defined as "the use of physical action to alter the information processing requirements of a task so as to reduce cognitive demand" [39]. It is a mechanism which enables people to work around the limitations of mental resources like working memory and perception. The idea of offloading is that a physical action takes on some of the demand of the task so there is less demand placed on the cognitive resources of the individual. As discussed, working memory capacity is limited, and the same is true about other cognitive resources It is not always feasible to change information presentation or clarify instructions on a large scale in order to minimize CL. This is why developing strategies on the individual level to cope with excessive CL is so beneficial. Offloading provides a method for people to manage CL levels in specific contexts.

There are a variety of methods for cognitive offloading, but the most important category to understand in this case is called "into-the-world". This term refers to the type of tool the processing demand is being offloaded onto in the method. "Onto-the body" is second method in which some cognitive requirements of the task are absorbed by a physical action of the body [39]. For example, someone counting on their fingers or tilting their head to view a crooked image would fall under the category of "onto-the-body". However, many of the offloading strategies which are most easily recognized fall under "into-the-world". Offloading into-the-world involves the use of a tool or device to create an external representation of information. Rather than an individual having to store the information in memory, it is stored in a physical, separate form which utilizes no (or fewer) cognitive resources [39]. Additionally, offloading into-the-world has the benefit of being helpful for both retrospective memory (remembering events, people, experiences, or information from the past) as well as prospective memory (remembering to complete a future planned action). Both of these types of memory are essential for task completion. Retrospective memory allows us to remember all of the relevant experiences and information that inform how we accomplish a goal. However, prospective memory is just as important for prioritizing and managing workflow. Offloading provides techniques to better manage both retrospective and prospective memory requirements within a task.

Cognitive offloading is beneficial for anyone, regardless of background or abilities. It has been demonstrated to reduce cognitive load and improve performance on a variety of tasks [39]. However, this strategy is not as accessible to blind and low vision (BLV) individuals as it is to sighted individuals. **The primary reason is that the most commonly used tools for cognitive offloading are vision-based in design.** They include devices like paper calendars

and agendas, written lists, and google calendar [6]. Although some of these tools can still be used by BLV individuals, many of their most helpful features are inaccessible to someone with impaired vision. For this reason, BLV individuals face unique challenges with regard to managing CL compared to sighted people. This is partially because BLV individuals have fewer offloading options available to manage their cognitive resources productively.

5 Cognitive Load in Design of Accessible Solutions

Interestingly, it is specifically the process of making working memory space available which appears to cause performance increases [25]. The act of offloading information from working memory to external memory aids is the most common form of this operation in everyday life. There has long been uncertainty about the mechanisms behind removing information from working memory leading to performance benefits. Some scholars believed it was primarily due to reduced proactive interference (the action of prior information in memory inhibiting the ability to learn new information). However, other researchers thought it was more likely a result of freeing up space in working memory which had previously been occupied by those items. Most recent findings suggest that working memory capacity is the more probable of these two explanations, although both doubtlessly play a role in the complicated cognitive interaction [25].

It is for this reason that researchers should consider cognitive load and cognitive offloading in the design of accessibility tools for BLV individuals. In some areas of accessibility research, this is already common practice. However, it is still an exception in many accessible programming tools. We will explore the state of existing accessible programming tools, a majority of which do not examine cognitive load in their design. We will also discuss how these tools can improve accessibility further by considering cognitive load and providing tools for cognitive offloading within their models.

5.1 Non-Programming Tasks

It would be a stretch to say that designing with CL in mind is the norm in BLV accessibility research, but it is true that many more tools for BLV accessibility have taken it into account in recent years. Especially for accessibility solutions outside of programming, there are a number of tools which factor CL into their design. A great, and recent, example of this is the Aerial Guide Dog, a tool to help BLV individuals with navigation indoors [62]. Early on in the design process of the navigation tool, the authors address that many navigation solutions provide an overwhelming amount of information to the user. The many cues and sensory signals can require a huge amount of mental effort to decode, and are coupled with a pretty steep learning curve. In some cases, this can look like having to learn the meaning of a dozen different auditory cues. This far exceeds the capacity of an average person's working memory, so it's very difficult to learn quickly when an individual has not yet established any strategies or schemas to store the information effectively outside the working memory. The Aerial Guide Dog design instead takes advantage of instinctive tactile senses by physically guiding the user, much like a real guide dog. The idea is that this requires far less conscious mental effort from the user and frees up resources for other processes.

Similarly, a 2022 study evaluated BLV users interacting with a computer using different methods from a CL perspective [31] One group in the study used braille to interact with the computer, and the other used dactylology, a method utilizing gestures for computer interaction. There were also three conditions for CL (low, medium, high) mediated by task complexity (ICL) which allowed the researchers to compare performance on the task at different levels of load. Their goal was to identify if the pattern of performance between the interaction methods differed at different levels of CL. Performance was measured based on response time and false responses.

These are just a couple of the accessibility tools for BLV individuals in the last few years which have considered CL in their design, but they highlight the point that CL is an important factor. Particularly for BLV people, designing tools which minimize CL is critical to optimizing performance for the people using those tools.

5.2 Programming Tasks

In order to evaluate the current level of consideration for CL in existing programming accessibility solutions, we assessed a collection of recent tools developed in that area. The tools evaluated for CL consideration come from a 2022 literature review [34]. In the review, the authors give an overview of some of these existing accessibility tools and discuss the coding challenge addressed by each of them. We evaluated the same list of tools for our purposes. The results of the review are found in Table 1. The programming challenge addressed by each tool, as identified in the original paper, is included. The new addition to the table is the final column which recognizes the extent to which CL was considered in the design of the solution. Tools listed as 'No' did not include any discussion of CL in the design rationale or any evaluation of CL when measuring participant performance with and without the tool. Those listed as 'Yes' explicitly mentioned CL-related concepts such as the difficulty of distinguishing many different commands or wanting to avoid overwhelming users with excessive information. Finally, the 'Maybe' designation is given to those tools which either discussed CL-adjacent concepts, but did not incorporate them into the tool design, or included elements in the tool design which were intended to reduce CL, but did not mention the motivation behind the choice.

The design of accessible programming tools for BLV individuals seems to be a little behind non-programming design in regards to consideration for CL. Some of these accessibility solutions address CL in their design process, but it is not common. This is important because evidence supports that CL plays a role in learning and task performance. Also because it represents a gap between programming and non-programming accessibility tool design. This known link between CL, learning, and task performance suggests that it would be beneficial to the area of BLV accessibility research to consider CL in studies. Quantifying the influence of CL on the effectiveness of accessibility tools can deepen the understanding researchers have and open doors to improved designs.

6 Opportunity: Creating a "Virtual Code View"

In SQL, a *view* is a virtual table based on the result-set of an SQL statement. It presents data in rows and columns, similar to a real table, but it does not store the data itself. Instead, a view's data is dynamically generated by executing the underlying SQL query whenever the view is accessed.

In the same vein, we propose the need of a "virtual code view" for BLV developers. This "view" presents code in a manner that is a "relatively lighter" load for BLV developers when performing downstream programming tasks. The initial set of criteria for the refactoring and creating this "view" are:

- **Flattening Code**: Nested methods, conditionals, or control statements are creative choices that impact BLV developers for code navigation-based tasks. We would refactor the code to "flatten it" such that navigating the code requires less of jumping around and jumping within.
- **Minimize variable or function call stack size**: As we pointed out, it is ideal for task performance if the working memory has to store (or recall) 3-5 items of information. We would refactor the code such that the combination of the size of the call stack, responsible for keeping track of functions call, and scope of variables, is minimized to lowest possible.

Table 1. Cognitive Load in Design of Accessible Programming Tools

Tool Name	Year	Programming Challenge	CL?
ACONT [15]	2018	Code Navigation, Code Skimming	No
APL [56, 57]	2005	Inaccessibility of programming languages	Yes
AudioHighlight [3]	2018	Code Navigation, Code Skimming	Maybe
Aural Tree Navigator [49]	2003	Code Navigation, Code Skimming	No
Blockly (Accessible Version; Caraco et al.) [8]	2019	Inaccessibility of BBLs	No
Blockly (Accessible Version; Ludi et al.) [27]	2015	Inaccessibility of BBLs	No
Blocks4All [29, 30, 35]	2018	Inaccessibility of BBLs	No
Bonk [18]	2018	Inaccessibility of programming through media	No
COBRIX [2]	2017	Inaccessibility of programming tools	No
Code Mirror Block [43]	2019	Code Navigation, Code Editing	Maybe
CodeBox64 [61]	2017	Inaccessibility of BBLs	No
CodeRhythm [40, 41]	2020	Challenges of tangible programming environments	No
CodeTalk [36]	2018	Code Navigation, Code Comprehension, Code Editing, Code Debugging	No
Donnie [28]	2017	Inaccessibility of programming languages and robotics toolkit	No
GUIDL [19]	2012	Inaccessibility of GUI-based applications	No
JavaSpeak [50]	2000	Code Navigation, Code Skimming	No
JBrick [26]	2010	Inaccessibility of robotics programming environments	Maybe
Music Blocks [42]	2020	Inaccessibility of BBLs	Maybe
Noodle [24]	2014	Inaccessibility of visual programming systems	Yes
P-CUBE [17]	2013	Inaccessibility of BBLs	No
Phogo [32]	2017	Inaccessibility of programming languages	No
Pseudospatial Blocks [22]	2016	Inaccessibility of BBLs	No
Robbie [14, 38]	2012	Inaccessibility of visual feedback produced by robots	Maybe
Scripting Language [13, 45–48, 56]	2002	Inaccessibility of GUI-based applications	Maybe
SODBeans [52–54]	2008	Code Debugging	No
Sparsha [44]	2020	Editing visual layout templates	No
Story Blocks [20, 21]	2017	Inaccessibility of BBLs	No
StructJumper [4]	2015	Code Navigation, Code Skimming	Maybe
Tactile Code Skimmer [12]	2019	Code Navigation, Code Skimming	Yes
Tangible Programming Tool Prototype [59]	2020	Inaccessibility of BBLs	No
TIP-Toy [7]	2020	Inaccessibility of BBLs	Maybe
Torino [33, 58, 60]	2017	Inaccessibility of BBLs	Maybe
Wicked Audio Debugger [51]	2007	Code Debugging	Maybe

- **Character limit for each line**: There are a small subset of BLV developers who rely on external devices, such as Braille displays, for code comprehension. There is no single "standard" size for Braille displays. One can find displays with up to 20 braille cells and up to 40 braille cells. Having lines of code span more than 40 characters causes an inconvenience because it requires storing (or recalling) prior information. We would refactor the code to have a upper bound of 40 characters per line.

These would be the "version 1.0" of guidelines for "Code Accessibility" that LLMs could use to generate "code views" with lighter cognitive load. With a "virtual view" based on cognitive load theory, we are preparing to run a user study to evaluate task performance across different sub-tasks.

7 Conclusion

This effort is the first step and we plan to extend and/or modify virtual code view based on other psychological, and environmental factors. As always, any changes to our conceptual design will continuously be tested with BLV developers. Given the fundamental importance of the role of cognitive load in the performance of tasks, it is surprising that we discovered its sporadic use as a measure for evaluating or a basis for designing solutions to support programming for BLV individuals. Through our detailed review of existing solutions, we highlight shortcomings and, thereby, a gaping opportunity to present BLV developers the tools that would make all of programming significantly more accessible.

References

[1] Kirsten C.S. Adam, Edward K. Vogel, and Edward Awh. 2017. Clear evidence for item limits in visual working memory. *Cognitive Psychology* 97 (Sept. 2017), 79–97. doi:10.1016/j.cogpsych.2017.07.001

[2] Jung-Hyun Ahn, Woonhee Sung, Sun Woong Lee, and Jang Hee I. 2017. COBRIX: A Physical Computing Interface for Visually Impaired/Blind Students to Learn Programming. In *Proceedings of Society for Information Technology & Teacher Education International Conference*. Association for the Advancement of Computing in Education (AACE), Austin, TX, USA, 727–733. https://www.learntechlib.org/primary/p/177876/

[3] Ameer Armaly, Paige Rodeghero, and Collin McMillan. 2018. AudioHighlight: Code Skimming for Blind Programmers. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, Madrid, 206–216. doi:10.1109/ICSME.2018.00030

[4] Catherine M. Baker, Lauren R. Milne, and Richard E. Ladner. 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*. ACM, Seoul Republic of Korea, 3043–3052. doi:10.1145/2702123.2702589

[5] Maria Bannert. 2002. Managing cognitive load—recent trends in cognitive load theory. *Learning and Instruction* 12, 1 (Feb. 2002), 139–146. doi:10.1016/S0959-4752(01)00021-4

[6] Catarina Baptista, Rosa Marina Afonso, and Ana Rita Silva. 2023. Practitioners' knowledge, acceptability and use of external memory aids with individuals with cognitive deficits: An exploratory study. *Neuropsychological Rehabilitation* 33, 5 (May 2023), 745–763. doi:10.1080/09602011.2022.2044354

[7] Giulia Barbareschi, Enrico Costanza, and Catherine Holloway. 2020. TIP-Toy: a tactile, open-source computational toolkit to support learning across visual abilities. In *Proceedings of the 22nd International ACM SIGACCESS Conference on Computers and Accessibility*. ACM, Virtual Event Greece, 1–14. doi:10.1145/3373625.3417005

[8] Logan B. Caraco, Sebastian Deibel, Yufan Ma, and Lauren R. Milne. 2019. Making the Blockly Library Accessible via Touchscreen. In *Proceedings of the 21st International ACM SIGACCESS Conference on Computers and Accessibility*. ACM, Pittsburgh PA USA, 648–650. doi:10.1145/3308561.3354589

[9] Meghan D. Caulfield, David C. Zhu, J. Devin McAuley, and Richard J. Servatius. 2016. Cerebellar response to familiar and novel stimuli: An fMRI study. *Behavioral Neuroscience* 130, 6 (Dec. 2016), 585–592. doi:10.1037/bne0000173

[10] Nelson Cowan. 2010. The Magical Mystery Four: How Is Working Memory Capacity Limited, and Why? *Current Directions in Psychological Science* 19, 1 (Feb. 2010), 51–57. doi:10.1177/0963721409359277

[11] Nicolas Debue and CÃ©cile Van De Leemput. 2014. What does germane load mean? An empirical contribution to the cognitive load theory. *Frontiers in Psychology* 5 (Oct. 2014), 12 pages. doi:10.3389/fpsyg.2014.01099

[12] Olutayo Falase, Alexa F. Siu, and Sean Follmer. 2019. Tactile Code Skimmer: A Tool to Help Blind Programmers Feel the Structure of Code. In *Proceedings of the 21st International ACM SIGACCESS Conference on Computers and Accessibility*. ACM, Pittsburgh PA USA, 536–538. doi:10.1145/3308561.3354616

[13] Kenneth G. Franqueiro and Robert M. Siegfried. 2006. Designing a scripting language to help the blind program visually. In *Proceedings of the 8th international ACM SIGACCESS conference on Computers and accessibility*. ACM, Portland Oregon USA, 241–242. doi:10.1145/1168987.1169035

[14] Ayanna M. Howard, Chung Hyuk Park, and Sekou Remy. 2012. Using Haptic and Auditory Interaction Tools to Engage Students with Visual Impairments in Robot Programming Activities. *IEEE Transactions on Learning Technologies* 5, 1 (2012), 87–95. doi:10.1109/TLT.2011.28

[15] Joe Hutchinson and Oussama Metatla. 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*. ACM, Montreal QC Canada, 1–6. doi:10.1145/3170427.3188696

[16] Didem Issever, Mehmet Cem Catalbas, and Fecir Duran. 2023. Examining Factors Influencing Cognitive Load of Computer Programmers. *Brain Sciences* 13, 8 (July 2023), 1132. doi:10.3390/brainsci13081132

[17] Shun Kakehashi, Tatsuo Motoyoshi, Kenfichi Koyanagi, Toru Ohshima, and Hiroshi Kawakami. 2013. P-CUBE: Block Type Programming Tool for Visual Impairments. In *2013 Conference on Technologies and Applications of Artificial Intelligence*. IEEE, Taipei, Taiwan, 294–299. doi:10.1109/TAAI.2013.65

[18] Shaun K. Kane, Varsha Koushik, and Annika Muehlbradt. 2018. Bonk: accessible programming for accessible audio games. In *Proceedings of the 17th ACM Conference on Interaction Design and Children*. ACM, Trondheim Norway, 132–142. doi:10.1145/3202185.3202754

[19] Mario Konecki. 2012. A new approach towards visual programming for the blinds. In *2012 Proceedings of the 35th International Convention MIPRO*. IEEE, Opatija, Croatia, 935–940.

[20] Varsha Koushik, Darren Guinness, and Shaun K. Kane. 2019. StoryBlocks: A Tangible Programming Game To Create Accessible Audio Stories. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM, Glasgow Scotland Uk, 1–12. doi:10.1145/3290605.3300722

[21] Varsha Koushik and Shaun K. Kane. 2017. Tangibles + Programming + Audio Stories = Fun. In *Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility*. ACM, Baltimore Maryland USA, 341–342. doi:10.1145/3132525.3134769

[22] Varsha Koushik and Clayton Lewis. 2016. An Accessible Blocks Language: Work in Progress. In *Proceedings of the 18th International ACM SIGACCESS Conference on Computers and Accessibility*. ACM, Reno Nevada USA, 317–318. doi:10.1145/2982142.2982150

[23] Olave E. Krigolson, Hayley Heinekey, Courtney M. Kent, and Todd C. Handy. 2012. Cognitive load impacts error evaluation within medial-frontal cortex. *Brain Research* 1430 (Jan. 2012), 62–67. doi:10.1016/j.brainres.2011.10.028

[24] Clayton Lewis. 2014. Work in Progress Report: Nonvisual Visual Programming. In *Proceedings of the 25th Psychology of Programming Annual Conference (PPIG 2014)*. PPIG, Brighton, UK, 6 pages.

[25] Chenyu Li, Gidon T. Frischkorn, Hannah Dames, and Klaus Oberauer. 2025. The benefit of removing information from working memory: Increasing available cognitive resources or reducing interference? *Cognition* 260 (July 2025), 106134. doi:10.1016/j.cognition.2025.106134

[26] Stephanie Ludi, Mohammed Abadi, Yuji Fujiki, Priya Sankaran, and Spencer Herzberg. 2010. JBrick: accessible lego mindstorm programming tool for users who are visually impaired. In *Proceedings of the 12th international ACM SIGACCESS conference on Computers and accessibility*. ACM, Orlando Florida USA, 271–272. doi:10.1145/1878803.1878866

[27] Stephanie Ludi and Mary Spencer. 2017. Design Considerations to Increase Block-based Language Accessibility for Blind Programmers Via Blockly. *Journal of Visual Languages and Sentient Systems* 3, 1 (July 2017), 119–124. doi:10.18293/VLSS2017-013

[28] Guilherme H. M. Marques, Daniel C. Einloft, Augusto C. P. Bergamin, Joice A. Marek, Renan G. Maidana, Marcia B. Campos, Isabel H. Manssour, and Alexandre M. Amory. 2017. Donnie robot: Towards an accessible and educational robot for visually impaired people. In *2017 Latin American Robotics Symposium (LARS) and 2017 Brazilian Symposium on Robotics (SBR)*. IEEE, Curitiba, 1–6. doi:10.1109/SBR-LARS-R.2017.8215273

[29] Lauren R. Milne and Richard E. Ladner. 2018. Blocks4All: Overcoming Accessibility Barriers to Blocks Programming for Children with Visual Impairments. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, Montreal QC Canada, 1–10. doi:10.1145/3173574.3173643

[30] Lauren R. Milne and Richard E. Ladner. 2019. Position: Accessible Block-Based Programming: Why and How. In *2019 IEEE Blocks and Beyond Workshop (B&B)*. IEEE, Memphis, TN, USA, 19–22. doi:10.1109/BB48857.2019.8941230

[31] Gourav Modanwal, Shashi Bhushan Rai, Aishwarya Jaiswal, Tushar Singh, and Kishor Sarawadekar. 2022. Can Visually Impaired Use Gestures to Interact With Computers? A Cognitive Load Perspective. *IEEE Transactions on Human-Machine Systems* 52, 2 (April 2022), 267–275. doi:10.1109/THMS.2022.3144002

[32] Pablo Molins-Ruano, Carlos Gonzalez-Sacristan, and Carlos Garcia-Saura. 2018. Phogo: A low cost, free and "maker" revisit to Logo. *Computers in Human Behavior* 80 (March 2018), 428–440. doi:10.1016/j.chb.2017.09.029

[33] Cecily Morrison, Nicolas Villar, Anja Thieme, Zahra Ashktorab, Eloise Taysom, Oscar Salandin, Daniel Cletheroe, Greg Saul, Alan F Blackwell, Darren Edge, Martin Grayson, and Haiyan Zhang. 2020. Torino: A Tangible Programming Language Inclusive of Children with Visual Disabilities. *Human–Computer Interaction* 35, 3 (May 2020), 191–239. doi:10.1080/07370024.2018.1512413

[34] Aboubakar Mountapmbeme, Obianuju Okafor, and Stephanie Ludi. 2022. Addressing Accessibility Barriers in Programming for People with Visual Impairments: A Literature Review. *ACM Transactions on Accessible Computing* 15, 1 (March 2022), 1–26. doi:10.1145/3507469

[35] Jacqueline Shao Yi Ong, Nana Adwoa O. Amoah, Alison E. Garrett-Engele, Mariella Irene Page, Katherine R. McCarthy, and Lauren R. Milne. 2019. Demo: Expanding Blocks4All with Variables and Functions. In *Proceedings of the 21st International ACM SIGACCESS Conference on Computers and Accessibility*. ACM, Pittsburgh PA USA, 645–647. doi:10.1145/3308561.3354588

[36] Venkatesh Potluri, Priyan Vaithilingam, Suresh Iyengar, Y. Vidya, Manohar Swaminathan, and Gopal Srinivasa. 2018. CodeTalk: Improving Programming Environment Accessibility for Visually Impaired Developers. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, Montreal QC Canada, 1–11. doi:10.1145/3173574.3174192

[37] Charan Ranganath and Gregor Rainer. 2003. Neural mechanisms for detecting and remembering novel events. *Nature Reviews Neuroscience* 4, 3 (March 2003), 193–202. doi:10.1038/nrn1052

[38] Sekou L. Remy. 2013. Extending access to personalized verbal feedback about robots for programming students with visual impairments. In *Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility*. ACM, Bellevue Washington, 1–2. doi:10.1145/2513383.2513384

[39] Evan F. Risko and Sam J. Gilbert. 2016. Cognitive Offloading. *Trends in Cognitive Sciences* 20, 9 (Sept. 2016), 676–688. doi:10.1016/j.tics.2016.07.002

[40] Zhiyi Rong, Ngo Fung Chan, Taizhou Chen, and Kening Zhu. 2020. CodeRhythm: Designing Inclusive Tangible Programming Blocks. In *Companion Publication of the 2020 ACM Designing Interactive Systems Conference*. ACM, Eindhoven Netherlands, 105–110. doi:10.1145/3393914.3395895

[41] Zhiyi Rong, Ngo Fung Chan, Taizhou Chen, and Kening Zhu. 2020. *Toward Inclusive Learning: Designing and Evaluating Tangible Programming Blocks for Visually Impaired Students*. Lecture Notes in Computer Science, Vol. 12183. Springer International Publishing, Cham, 326–338. doi:10.1007/978-3-030-49065-2_23

[42] Alpay Sabuncuoglu. 2020. Tangible Music Programming Blocks for Visually Impaired Children. In *Proceedings of the Fourteenth International Conference on Tangible, Embedded, and Embodied Interaction*. ACM, Sydney NSW Australia, 423–429. doi:10.1145/3374920.3374939

[43] Emmanuel Schanzer, Sina Bahram, and Shriram Krishnamurthi. 2019. Accessible AST-Based Programming for Visually-Impaired Programmers. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. ACM, Minneapolis MN USA, 773–779. doi:10.1145/3287324.3287499

[44] Ashrith Shrinivas Shetty. 2020. *Tangible Web Layout Design for Blind and Visually Impaired People*. Masters Thesis. University of Maryland, College Park. https://umw.idm.oclc.org/login?url=https://www.proquest.com/dissertations-theses/tangible-web-layout-design-blind-visually/docview/2434840938/se-2?accountid

[45] Robert M. Siegfried. 2002. A scripting language to help the blind to program visually. *ACM SIGPLAN Notices* 37, 2 (Feb. 2002), 53–56. doi:10.1145/568600.568611

[46] Robert M. Siegfried. 2006. Visual programming and the blind: the challenge and the opportunity. In *Proceedings of the 37th SIGCSE technical symposium on Computer science education*. ACM, Houston Texas USA, 275–278. doi:10.1145/1121341.1121427

[47] Robert M. Siegfried, Denis Diakoniarakis, Kenneth G. Franqueiro, and Amol Jain. 2005. Extending a scripting language for visual basic forms. *ACM SIGPLAN Notices* 40, 11 (Nov. 2005), 37–40. doi:10.1145/1107541.1107547

[48] Robert M Siegfried, Denis Diakoniarakis, and Uchechukwu Obianyo-Agu. 2005. Teaching the Blind to Program Visually. *Information Systems Education Journal* 3, 23 (2005), 9 pages. http://isedj.org/3/23/

[49] Ann C. Smith, Justin S. Cook, Joan M. Francioni, Asif Hossain, Mohd Anwar, and M. Fayezur Rahman. 2003. Nonvisual tool for navigating hierarchical structures. In *ACM SIGACCESS Accessibility and Computing*. ACM, Atlanta, Georgia, USA, 133–139. doi:10.1145/1029014.1028654

[50] Ann C. Smith, Joan M. Francioni, and Sam D. Matzek. 2000. A Java programming tool for students with visual disabilities. In *Proceedings of the fourth international ACM conference on Assistive technologies*. ACM, Arlington Virginia USA, 142–148. doi:10.1145/354324.354356

[51] A. Stefik, R. Alexander, R. Patterson, and J. Brown. 2007. WAD: A Feasibility study using the Wicked Audio Debugger. In *15th IEEE International Conference on Program Comprehension (ICPC '07)*. IEEE, Banff, Alberta, BC, 69–80. doi:10.1109/ICPC.2007.42

[52] Andreas Stefik, Andrew Haywood, Shahzada Mansoor, Brock Dunda, and Daniel Garcia. 2009. SODBeans. In *2009 IEEE 17th International Conference on Program Comprehension*. IEEE, Vancouver, BC, Canada, 293–294. doi:10.1109/ICPC.2009.5090064

[53] Andreas Mikal Stefik. 2008. *On the Design of Program Execution Environments for Non-Sighted Computer Programmers*. Doctoral dissertation. Washington State University.

[54] Andreas M. Stefik, Christopher Hundhausen, and Derrick Smith. 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*. ACM, Dallas TX USA, 571–576. doi:10.1145/1953163.1953323

[55] John Sweller. 1988. Cognitive Load During Problem Solving: Effects on Learning. *Cognitive Science* 12, 2 (April 1988), 257–285. doi:10.1207/s15516709cog1202_4

[56] Jaime Sánchez and Fernando Aguayo. 2005. Blind learners programming through audio. In *CHI '05 Extended Abstracts on Human Factors in Computing Systems*. ACM, Portland OR USA, 1769–1772. doi:10.1145/1056808.1057018

[57] Jaime Sánchez and Fernando Aguayo. 2006. *APL: Audio Programming Language for Blind Learners*. Lecture Notes in Computer Science, Vol. 4061. Springer Berlin Heidelberg, Berlin, Heidelberg, 1334–1341. doi:10.1007/11788713_192

[58] Anja Thieme, Cecily Morrison, Nicolas Villar, Martin Grayson, and Siân Lindley. 2017. Enabling Collaboration in Learning Computer Programing Inclusive of Children with Vision Impairments. In *Proceedings of the 2017 Conference on Designing Interactive Systems*. ACM, Edinburgh United Kingdom, 739–752. doi:10.1145/3064663.3064689

[59] Emmanuel Utreras and Enrico Pontelli. 2020. *Accessibility of Block-Based Introductory Programming Languages and a Tangible Programming Tool Prototype*. Lecture Notes in Computer Science, Vol. 12376. Springer International Publishing, Cham, 27–34. doi:10.1007/978-3-030-58796-3_4

[60] Nicolas Villar, Cecily Morrison, Daniel Cletheroe, Tim Regan, Anja Thieme, and Greg Saul. 2019. Physical Programming for Blind and Low Vision Children at Scale. In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM, Glasgow Scotland Uk, 1–4. doi:10.1145/3290607.3313241

[61] Zirui Wang and Amber Wagner. 2019. Evaluating a Tactile Approach to Programming Scratch. In *Proceedings of the 2019 ACM Southeast Conference*. ACM, Kennesaw GA USA, 226–229. doi:10.1145/3299815.3314464

[62] Xiaochen Zhang, Ziyi Pan, Ziyang Song, Yang Zhang, Wujing Li, and Shiyao Ding. 2024. The Aerial Guide Dog: A Low-Cognitive-Load Indoor Electronic Travel Aid for Visually Impaired Individuals. *Sensors* 24, 1 (Jan. 2024), 297. doi:10.3390/s24010297

Received 11 September 2025