Gridlock: Plain Text with Guides

Sam Cohen
samcohen@uchicago.edu
University of Chicago
Chicago, IL, USA

Ravi Chugh
rchugh@cs.uchicago.edu
University of Chicago
Chicago, IL, USA

Abstract

Plain text documents are pervasive in software development for code files, documentation, as well as human-readable data serialization formats. They can be diffed, checked in to version control systems, easily searched, and manipulated by countless other tools which understand this universal format. Although unstructured, plain text documents typically conform to a formal syntax, such as in a programming language, or visual conventions, such as the alignment of columns in a table. Maintaining such documents often requires tedious secondary edits to keep the formatting in sync.

We present Gridlock, a text editor with *guides*, which users manipulate to align, group, or hide parts of their plain text documents. Vertical and horizontal guides in Gridlock form a kind of cellular structure, in which the underlying data is partitioned into semantic fragments while retaining raw text-editing capabilities. Our design adapts familiar notions from spreadsheets—such as rows, columns, and cell resizing and clipping—into a user interface that is fundamentally centered around a plain-text editing experience. We evaluate our design through several small but suggestive examples, involving code, tabular data, and ASCII diagrams.

CCS Concepts

• **Software and its engineering** → *Integrated and visual development environments*; • **Human-centered computing** → *Graphical user interfaces.*

Keywords

Plain Text, Code Formatting, Code Editors, Spreadsheets

ACM Reference Format:
Sam Cohen and Ravi Chugh. 2026. Gridlock: Plain Text with Guides. In *The 39th Annual ACM Symposium on User Interface Software and Technology (UIST '26), November 02–05, 2026, Detroit, MI, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3830398.3830679

1 Introduction

Word processors, spreadsheet applications, and other document editors provide an array of interactive tools for styling and positioning text. Yet in software development, programmers favor "plain text editors" that provide direct, low-level control over the individual characters that comprise a file. Crucially, plain text editors grant authors precise control over the *formatting* of text.

This manual formatting is implemented by judiciously using spaces, newlines, and other characters to ensure the *alignment* of certain fragments across lines. A form of secondary notation [2], formatted text may convey additional information to readers beyond what is strictly required by the syntax of the underlying document. In code, alignment can draw the eye to similarity between phrases, or show which parts of an expression or statement are important, and which parts are just syntactic noise. In comments, "glue" characters, like spaces or hyphens or pipes, can be used to implement diagrams which describe algorithms or data formats. Beyond code, human-readable document formats, such as Markdown, reStructuredText (RST), and TEX, use formatting to imitate WYSIWYG editors where plain text is more desirable.

As formatted text comprises informal constraints spanning multiple lines, edits to one line often require secondary edits elsewhere in the file. Depending on the amount of text, as well as the features provided by a specific editor, these secondary edits may be extensive. In "rich" document editors, such as Google Docs or Microsoft Word for writing, and Adobe Illustrator or Figma for diagramming, users can define *guides* (*vertical rules* or *tab stops*) that help automate the process of aligning or "snapping" different types of textual and visual elements. The question that motivates our work is: ***Can plain text editors be extended with guides to help users edit and maintain alignment constraints in formatted text?***

In this short paper, we present the design and prototype implementation of Gridlock, an editor that augments conventional plain-text editing capabilities with new, spreadsheet-like interactions for manipulating cells containing plain-text characters. Next, we present an overview of Gridlock through several examples (§2). Then we discuss related work (§3) and conclude with opportunities for future work (§4). The source code of our prototype is available at https://github.com/sbcohen2000/gridlock.

2 Overview

The toolbar in Gridlock (Figure 1) includes several *tools* and *modes*, which allow the user to interact with the document in different ways. In the examples that follow, we introduce features as they become relevant.

Each example in this section is presented as a series of frames (the sum of which we call an *edit script*), depicting the editing area over

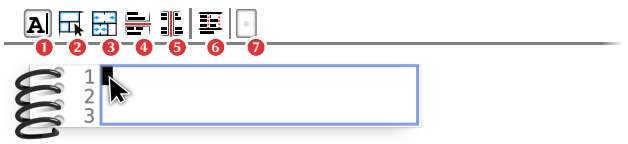

Figure 1: The tools and editing area of Gridlock. The tools are for: ❶ text editing, ❷ selecting cells, ❸ merging cells, ❹ inserting horizontal and ❺ vertical guides, ❻ disabling guides, and ❼ setting the glue character.

UIST '26, Detroit, MI, USA

ACM ISBN 979-8-4007-2856-3/2026/11
https://doi.org/10.1145/3830398.3830679

Figure 2: Example 1. Inserting a soft newline to a multiline string with and without guides. Many of the secondary edits from the first edit script can be inferred or simplified by the guide. E.g., the final column is re-adjusted via direct manipulation in F.

time. These scripts are annotated with marks which indicate how a hypothetical user interacted with the document. In addition to these scripts, the supplementary materials contain a video demonstration of each scenario.

Visual Conventions. To help explain the edit scripts, we have developed several kinds of annotations and other visual conventions. So that the annotations won't be confused with the interface itself, we will draw annotations using only the color red. A click is notated with , the endpoints of a drag with , and the endpoints of a rectangular selection with (though we will omit where it is clear from the context that a click must have occurred). Furthermore, sometimes multiple interactions occur in the same frame. If the order of these interactions is important, they are labeled with ❶ and ❷. Things that the user typed appear in the "tape." For example,

means that the user clicked, then pressed the "delete" (⌫) key, followed by the "space" (␣) key. Lastly, text that changed between frames is highlighted in yellow to make it easier to identify.

Example 1: Multiline String Literal

In JavaScript, it is common to represent a multiline string as a concatenation of single-line string literals, both to control the width of the expression, but also so that the literal syntax of the string

resembles the way it would be printed. Figure 2 shows such a string, which is broken into three lines. As written, the first line of the string is still very long, and so we might want to break it into two, inserting some leading spaces to signal that the long line has been *continued* onto the next.

In the top script of Figure 2, this edit is performed without guides, as it would be done in an ordinary text editor. In frames A-D, the user inserts the newline, and fixes-up the string delimiters so that the expression is syntactically valid JavaScript. But, this document also has an informal *secondary* syntax, which stipulates that the delimiters separating multiline strings should be aligned. Frames E-G show the edits necessary to satisfy this secondary notation, which involve many insertions and deletions.

In Gridlock, we can eliminate most of these secondary edits by inserting a *guide*. The bottom half of Figure 2 shows the same document as the top, save for the new guide preceding the last column. The guide makes the alignment of the delimiters explicit, and in doing so, allows the aligned objects to be manipulated directly.

The concrete effect of the guide can be seen in frames B and F. In frame B (c.f. frame B of the top script), the user types a newline, causing the text after `me,` to fall onto the subsequent line. However, in Gridlock, the behavior of newline insertion has been generalized so that, as opposed to an ordinary text editor, the text after the point is shifted down, *in its respective cell*. Since the + character is in a different cell, it *remains* in its cell, rather than following the text preceding it on its line (see 👁). The second effect of the guide is demonstrated in frame F. Because guides are explicit in Gridlock, they can directly manipulated. In this case, by dragging the guide to the left, the user can eliminate the secondary edits needed to re-position the delimiters.

Example 2: ASCII Art

In code files, practitioners use code formatting as an information channel, but information is also conveyed through formatted comments, sometimes in the form of ASCII diagrams [9]. This example deals with a *sequence diagram* [19]—rendered as ASCII art—describing a traditional tea brewing process [18], simplified to use only a cup instead of a cup and a bowl. In particular, it shows the steps to attach a label to the arrow on line 4 of the document. We accomplish an efficient edit script in this task by using *glue characters*, which are a feature of Gridlock that arise from the observation that, oftentimes, a row or column of the plain-text document is filled with repetitions of the same character.[1] Such is the case in Figure 3 A, where we'd like to represent each vertical line in the sequence diagram by a column of pipe characters.

In frame B, the user selects cells—delineated by guides—which represent these vertical lines. Clicking the glue-character tool in the toolbar and typing a character sets it on the selected cells (see C). The immediate effect is that all of the "empty" spaces in the cell are rendered as the glue character. This can be useful in reducing typing if the user needs to fill a large part of the document with the same character. But there is a more important secondary benefit, which is that when a cell must expand in response to a guide being dragged, or, as in this example, by typing a newline (see F), the system can

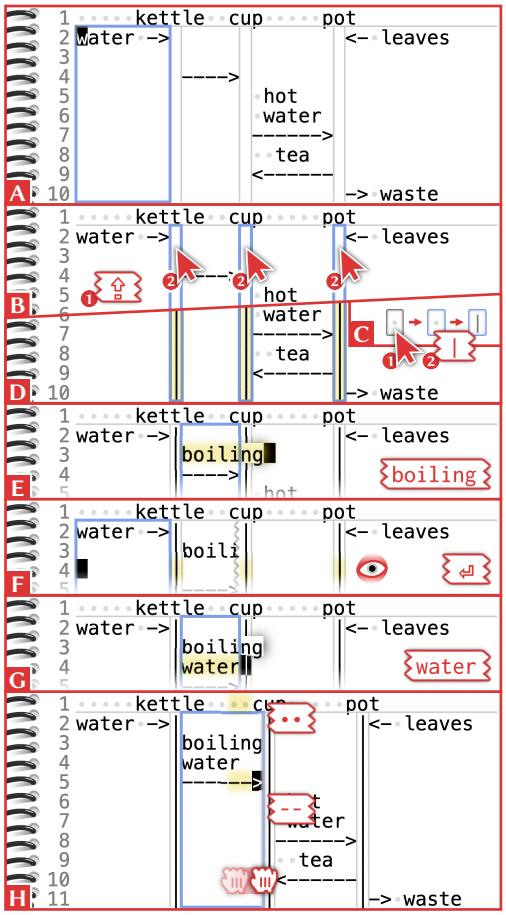

Figure 3: Example 2. Adding an arrow label to an ASCII diagram. From the initial document A, cells are selected B and filled with pipe characters C&D. In E-H, the label is inserted and its containing column is resized to fit.

fill the gaps with the cell's glue character (see 👁), eliminating a class of secondary edits.[2]

Figure 3 also demonstrates the behavior of text when it exceeds the boundary of its cell. Like Google Sheets [6] or Microsoft Excel [15], text is permitted to *overflow* its cell when the cell is selected (e.g. E&G), but clips to the cell boundary when unselected, as in frame F. It is the user's choice to expand the size of the cell to accommodate its content, as seen in frame H. Like Sheets and Excel, when a cell is selected its contents are shown in full, atop any cells which appear to the right.

When a cell is overflowing, the document is no longer representable by a plain-text file with the same formatting. To save documents with clipped cells, Gridlock first expands the cells, remembering their original dimensions so that they can be restored upon re-opening the document.

[1] Glue characters in Gridlock are the analog of custom borders in spreadsheets.

[2] There's a complementary benefit when a cell is shrunk, since glue characters at all times occupy just as much space as necessary to fill a cell, never causing an overflow.

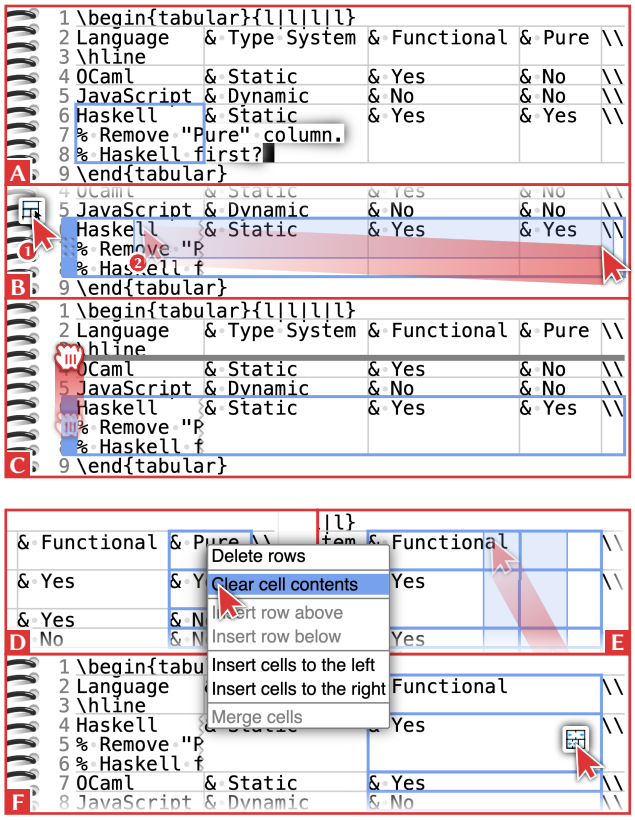

Figure 4: Example 3. Re-ordering a row, and removing a column in a LaTeX table. The last row is selected and re-ordered in B&C. The last column is removed in D-F.

Example 3: Tabular Data

In addition to diagrams, tables in various forms are common in code documentation, and in markup languages. Figure 4 shows a table in LaTeX, along with some spreadsheet-like operations which are possible because Gridlock's substrate is similar to that of spreadsheets. In the top half of the figure, the user selects a row of cells using a rectangular selection (see B), then reorders the row by dragging it near the margin (see C), in a movement that is essentially identical to Sheets and Excel.

In the second half of the figure, the user deletes the last column of the table by clearing the cells therein (see D), then merging with the cells to the left (see F). As opposed to the first script in this example, this second one deviates from how it might be accomplished in a traditional spreadsheet application: In Gridlock, it is not possible in general to merge cells spanning multiple rows because, as opposed to typical spreadsheet application, the boundaries of cells are local to their row, rather than spanning the entire document. The asymmetry between rows and columns in Gridlock exists because it seems more suited to the domain of editing text files, which are typically far longer than they are wide. Furthermore, a typical text document has many different formats in the *same file*, making it advantageous to use rows as a mechanism to "scope" the structure imposed by their cells.

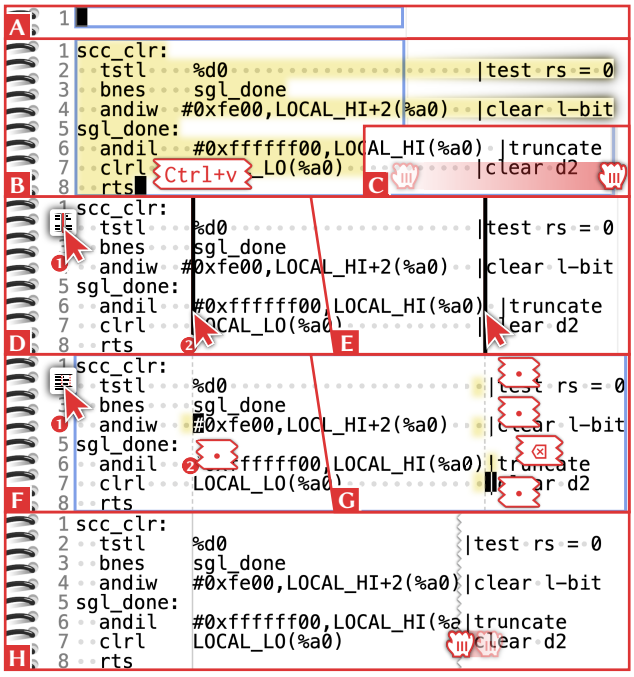

Figure 5: Example 4. Editing an assembly code fragment. The code is pasted into the document (A), annotated with guides (D&E), then cleaned up (F&G).

Example 4: Assembly Code

Our examples so far have demonstrated edits in documents which are already annotated with guides. In this final example, we build a document from scratch, first copying a pre-existing code fragment into the editor, then annotating it with guides. Figure 5 depicts an assembly program fragment taken from the Linux kernel, which is annotated with lots of comments. However, the alignment of the comments is uneven, since the instruction on line 6 is longer than the others. Our goal is to make the comment alignment explicit using a guide, correcting the misaligned row in the process.

We begin by copying the fragment into the editor (see B). Note that, like in Example 2, overflowing text does not automatically induce changes to the location of guides, but is instead clipped to the cell edge, which helps prevent non-local changes to the document when a cell is edited.

When copying unstructured code into a structured context, it can be challenging to place guides. In frame D, the user selects the vertical slice tool, and then places a new guide approximately at the column where the instruction operands start, splitting the cell into two. In this frame, and in frame E, where the user places a second guide, there is no ideal placement of guides which prevents all secondary edits; it is often necessary, as seen in frames F and G, to shuffle text *across* guides after they have been placed.

In order to accommodate this pattern, Gridlock provides a tool to temporarily disable the vertical guides, so that each row can be edited as if it were a single cell. The user disables guides in frame F (disabled guides are shown as dashed lines, rather than solid), and then proceeds to perform some adjustments in frames F and G to

move the instruction operands and comments to the correct side of the new guides. Guides are automatically re-enabled when the user performs some action which interacts with guides, such as dragging one, as in frame H.

3 Related Work

Gridlock aids manual ***code formatting*** through ***GUI interactions with text***, specifically via ***guides*** which add ***grid structure*** to text. This work contributes to research on ***substrates*** for "document-oriented" user interfaces, which attempt to replace disparate domain-specific tools with malleable, content-agnostic ones. We discuss each of these areas of related work below.

Guides. Our implementation is inspired by tabular spreadsheet editors such as Sheets and Excel, but also by guides as found in tools like Illustrator or Adobe Photoshop. StickyLines [3] provides additional first-class interactions with guides of the latter sort, such as "tweaking" their position to affect the layout of objects they constrain. Style Blink [22] also features a hand-drawn guide-like object (called a *divider* in their system) which, like Gridlock's guides, can be used to make horizontal or vertical space between elements on a canvas, and can be used to form tables.

Multi-cursor editing, as implemented in many mainstream text editors such as VS Code [16], can be seen as a kind of ephemeral guide; placed along several lines, a multi-cursor selection could be used to perform secondary edits more quickly than is possible with a single selection. In some ways multi-cursor editing is more flexible than editing with guides, because the cursors are not required to lie on the same column.

GUI Interactions with Text. Various GUIs have been proposed for editing text documents. These vary in the level of control the user has over the source text. On one end of the spectrum, tools like tablesgenerator.com produce read-only views of the source text which are meant to be copy-pasted into a text editor. But it is often useful to be able to edit the source as well as the high-level representation. i-LaTeX [5] is a LaTeX editor with domain-specific projectional editors (called *transitionals* in their system) for parts of a document, such as equations, images, and tables. Gridlock presents an intermediate approach which foregoes projectional editing, but encourages the document author to use formatting which imitates the projectional view, enabling some high-level operations (such as inserting new rows or columns) without leaving the source text.

Other systems offer even higher-level operations which cannot be phrased as one-to-one rewrites like a projection. Textoshop [14] extends a rich text editor with brushes, sliders, and GUI widgets to invoke various textual transformations. There are also similar innovations for editing code. In Code Shaping [23] and Code Brushes [17], code transformations are triggered by brushes that are more familiar to drawing editors rather than text editors.

Code Formatting. If formatting is not a priority, users may opt for autoformatting tools (e.g., prettier [12]). Reformulating pretty printing techniques [1, 20] to grant users the ability to retain granular control of formatting constraints—through direct-manipulation interactions with the text—is an alluring possibility.

Grid Structure for Text. Elastic Tabstops [7] reformulate ordinary tab stops so that they are local to sections of the document (like Gridlock), but are still introduced by explicitly-typed `tab` characters. In this system, guides (which are not explicitly visualized) are moved only when they are "pushed" by text to their left (as opposed to Gridlock, which instead temporarily renders overflowing text atop neighboring cells). Because of this, the guides in Elastic Tabstops can be controlled entirely by the keyboard. Elastic Tabstops represent a sort-of dual to Gridlock, where the guides are subject to the text contents of the document, rather than being tools for "shaping" the text.

Grid structures have also been superimposed on code for navigation. In Grid-Coding [4], a document is split into rows for each line, each line being further divided into columns based on the number of scopes therein. Ehtesham-Ul-Haque et al. [4] showed that this arrangement helped coders with low visibility navigate their files more effectively, the additional structure providing a scaffold by which they could traverse the program semantically rather than lexically. As opposed to Gridlock, the grid structure is inferred from the program text rather than being explicitly annotated.

Gridlock as a Substrate. Interaction Substrates [13] is a theory of user interface design which takes inspiration from physical tools (e.g., a knife), which have properties (such as being sharp), permitting them to act on objects in a predictable way. (The knife can cut fruit, for example, but also paper or any soft object.) Gridlock can be understood as a substrate which *contains* and *structures* its "objects of interest," which are guides and text. Guides have properties: they can *move* text to their right, *hide* text to their left, and enclose text into *cells*, which benefit from additional operations. The user can exploit these properties to structure their documents, using guides to develop for themselves some (specific, albeit limited) apparatus for viewing or editing their document, akin to how an animator might construct a *rig* to expedite animation by restricting the kinds of motion which are allowed.

Gridlock does not go as far as it could in embodying all of the aspects of a substrate. For example, though it is possible for text formatting to depend on a guide, it is not possible for guides in Gridlock to depend on one another, or for a dependency to exist between the contents of multiple cells. Both forms of dependency would be interesting to explore in future work, but the latter form is especially interesting because it could bring Gridlock closer to several related works including Textlets [8] and Potluck [11], both of which permit dependencies on fragments of text. Like Gridlock, these systems permit documents to contain additional structure, but the dependency relationship permitted among fragments of text effectively turns them into lightweight programming systems. Unlike Gridlock, this structure is annotated through selections or queries, rather than explicitly constructed cells.

Interaction substrates should permit *tool re-purposing*: the use of a tool for a purpose other than its learned or designed purpose. Renom et al. [21] study this idea using a task which focuses on editing text with a combination of text-focused and graphics-focused tools. They find that users may resist tool re-purposing when they can instead use a familiar tool. One implication for Gridlock is that, to encourage tool re-purposing (which guides are intended to promote), guides should differentiate themselves from familiar tools

such as tab stops, hopefully causing users to engage in *technical reasoning*. Questions about how users perceive guides, especially in comparison to familiar tools, would be best answered through observation of Gridlock in use, a pursuit we leave for future work.

4 Conclusion and Future Work

We presented Gridlock, a text editor with *guides*, which break a plain-text document into cells. By annotating a document with guides, the user declares its informal structure; in return, Gridlock helps maintain that structure and affords additional operations on it. These operations, like fixing-up ASCII delimiters (Figure 3), or adding a new row or column to a table (Figure 4), would amount to many secondary edits when the structure is opaque to the editor. Below, we discuss several future avenues to extend our design.

At its core Gridlock is a text editor, and text editors have enjoyed numerous usability enhancements such as find-and-replace, macros, and so on. We believe most of these feature are orthogonal to guides, so we haven't implemented them in our prototype. As described above, however, multiple cursors present an opportunity to integrate more meaningfully with guides, perhaps as a way to introduce guides into a document.

In the previous section, we discussed Gridlock through the lens of Interaction Substrates [13]. This perspective points to ways that Gridlock could be extended by permitting more kinds of relationships between objects in the system. Currently, Gridlock does not try to parse or comprehend the contents of its cells, but permitting the system to do so would allow more kinds of programming and templating workflows like those seen in Textlets [8] and Potluck [11]. We imagine programming systems built upon Gridlock whose cells are not homogeneous. Rather, some cells might be used for code, but others for documentation, diagrams, or program outputs—building on interface paradigms provided by literate programming [10] and computational notebooks.

Gridlock's guides could also carry dependency information. One obvious improvement would be to permit vertical guides to be constrained to one another, making tables easier to manipulate, and permitting synchronized non-local structures that could span multiple rows. Currently in Gridlock, guides also suffer from being manipulable only using the mouse (unlike cells, which operate like ordinary text buffers). If cells and guides instead shared some common language, it would be possible to manipulate guides using the keyboard, which could make editing more efficient.

In addition to functional enhancements, an important next step is to understand how Gridlock (and future extensions) may be used in practice. Although the edit scripts we have presented in this paper suggest that guides can expedite plain-text formatting, there is no substitute for user experience in understanding how guides might be used in more realistic contexts.

References

[1] Jean-Philippe Bernardy. 2017. A Pretty But Not Greedy Printer (Functional Pearl). *Proceedings of the ACM on Programming Languages (PACMPL), Issue ICFP*. doi:10.1145/3110250.

[2] Alan F. Blackwell et al. 2001. Cognitive Dimensions of Notations: Design Tools for Cognitive Technology. In *International Conference on Cognitive Technology: Instruments of Mind (CT 2001)*.

[3] Marianela Ciolfi Felice, Nolwenn Maudet, Wendy E. Mackay, and Michel Beaudouin-Lafon. 2016. Beyond Snapping: Persistent, Tweakable Alignment and Distribution with StickyLines. In *Symposium on User Interface Software and Technology (UIST)*. doi:10.1145/2984511.2984577.

[4] Md Ehtesham-Ul-Haque, Syed Mostofa Monsur, and Syed Masum Billah. 2022. Grid-Coding: An Accessible, Efficient, and Structured Coding Paradigm for Blind and Low-Vision Programmers. In *Symposium on User Interface Software and Technology (UIST)*. doi:10.1145/3526113.3545620.

[5] Camille Gobert and Michel Beaudouin-Lafon. 2022. i-LaTeX: Manipulating Transitional Representations between LaTeX Code and Generated Documents. In *Conference on Human Factors in Computing Systems (CHI)*. doi:10.1145/3491102.3517494.

[6] Google. 2026. Google Sheets. (2026). https://workspace.google.com/products/sheets/.

[7] Nick Gravgaard. 2024. Elastic Tabstops. (2024). https://nick-gravgaard.com/elastic-tabstops/.

[8] Han L. Han, Miguel A. Renom, Wendy E. Mackay, and Michel Beaudouin-Lafon. 2020. Textlets: Supporting Constraints and Consistency in Text Documents. In *Conference on Human Factors in Computing Systems (CHI)*. doi:10.1145/3313831.3376804.

[9] Devamardeep Hayatpur, Brian Hempel, Kathy Chen, William Duan, Philip Guo, and Haijun Xia. 2024. Taking ASCII Drawings Seriously: How Programmers Diagram Code. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. ACM. doi:10.1145/3613904.3642683.

[10] Donald E. Knuth. 1984. Literate programming. *Comput. J.*, 27, 2, 97–111. doi:10.1093/comjnl/27.2.97.

[11] Geoffrey Litt, Max Schoening, Paul Shen, and Paul Sonnentag. 2022. Potluck. (2022). https://www.inkandswitch.com/potluck/#text-documents-as-user-interfaces.

[12] James Long and contributors. 2026. Prettier. (2026). https://prettier.io.

[13] Wendy E. Mackay and Michel Beaudouin-Lafon. 2025. Interaction Substrates: Combining Power and Simplicity in Interactive Systems. In *Conference on Human Factors in Computing Systems (CHI)*. doi:10.1145/3706598.3714006.

[14] Damien Masson, Young-Ho Kim, and Fanny Chevalier. 2025. Textoshop: Interactions Inspired by Drawing Software to Facilitate Text Editing. In *Conference on Human Factors in Computing Systems (CHI)*. doi:10.1145/3706598.3713862.

[15] Microsoft. 2026. Microsoft Excel. (2026). https://www.microsoft.com/microsoft-365/excel.

[16] Microsoft. 2015–2026. Visual Studio Code. https://code.visualstudio.com/. (2015–2026). https://code.visualstudio.com/.

[17] GitHub Next. 2025. Code Brushes. (2025). https://githubnext.com/projects/code-brushes/.

[18] Brother Anthony of Taizé and Hong Kyeong-Hee. 2007. *The Korean Way of Tea*, 45–48. ISBN: 9788991913172.

[19] Object Management Group Standards Development Organization. 2017. Unified Modeling Language 2.5.1. (2017).

[20] Sorawee Porncharoenwase, Justin Pombrio, and Emina Torlak. 2023. A Pretty Expressive Printer. *Proceedings of the ACM on Programming Languages (PACMPL), Issue OOPSLA2*. doi:10.1145/3622837.

[21] Miguel A. Renom, Baptiste Caramiaux, and Michel Beaudouin-Lafon. 2023. Interaction Knowledge: Understanding the 'Mechanics' of Digital Tools. In *Conference on Human Factors in Computing Systems (CHI)*. doi:10.1145/3544548.3581246.

[22] Hugo Romat, Nicolai Marquardt, Ken Hinckley, and Nathalie Henry Riche. 2022. Style Blink: Exploring Digital Inking of Structured Information via Handcrafted Styling as a First-Class Object. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. doi:10.1145/3491102.3501988.

[23] Ryan Yen, Jian Zhao, and Daniel Vogel. 2025. Code Shaping: Iterative Code Editing with Free-form AI-Interpreted Sketching. In *Conference on Human Factors in Computing Systems (CHI)*. doi:10.1145/3706598.3713822.