

Textlets: Supporting Constraints and Consistency in Text Documents

Han L Han, Miguel Renom, Wendy Mackay, Michel Beaudouin-Lafon

▶ **To cite this version:**

Han L Han, Miguel Renom, Wendy Mackay, Michel Beaudouin-Lafon. Textlets: Supporting Constraints and Consistency in Text Documents. CHI '20 - 38th SIGCHI conference on Human Factors in computing systems, Apr 2020, Honolulu, HI, USA, United States. pp.1-13, 10.1145/3313831.3376804 . hal-02867300

HAL Id: hal-02867300
https://hal.science/hal-02867300

Submitted on 13 Jun 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Copyright

Textlets: Supporting Constraints and Consistency in Text Documents

Han L. Han **Miguel A. Renom** **Wendy E. Mackay** **Michel Beaudouin-Lafon**
Université Paris-Saclay, CNRS, Inria,
Laboratoire de Recherche en Informatique
F-91400 Orsay, France
{han.han, renom, mackay, mbl}@lri.fr

ABSTRACT
Writing technical documents frequently requires following constraints and consistently using domain-specific terms. We interviewed 12 legal professionals and found that they all use a standard word processor, but must rely on their memory to manage dependencies and maintain consistent vocabulary within their documents. We introduce *Textlets*, interactive objects that reify text selections into persistent items. We show how *Textlets* help manage consistency and constraints within the document, including selective search and replace, word count, and alternative wording. Eight participants tested a search-and-replace *Textlet* as a technology probe. All successfully interacted directly with the *Textlet* to perform advanced tasks; and most (6/8) spontaneously generated a novel replace-all-then-correct strategy. Participants suggested additional ideas, such as supporting collaborative editing over time by embedding a *Textlet* into the document to flag forbidden words. We argue that *Textlets* serve as a generative concept for creating powerful new tools for document editing.

Author Keywords
Text editing; Document processing; Reification

CCS Concepts
•**Human-centered computing** → **Graphical user interfaces; Interaction techniques;**

INTRODUCTION
Text editing was once considered a 'killer app' of personal computing [6]. Editing text is usually the first skill a novice computer user masters, and all personal computers are sold with a word processor. Many professions require advanced text editing skills to ensure consistent use of terms and expressions within structured documents, such as contracts, patents, technical manuals and research articles.

For example, lawyers begin each contract with a list of defined terms, and must use them consistently thereafter. This is critical, since 'minor' wording changes can have serious legal implications. For example, American patents define *"comprises"* as *"consists at least of"*; whereas *"consists of"* means *"consists only of"*, indicating a significantly different scope of protection. Sometimes terms are disallowed, e.g. the US Patent and Trademark Office (USPTO) does not accept *"new"*, *"improved"* or *"improvement of"* at the beginning of a patent title. Word limits are also common, such as the European Patent Office's 150-word limit for patent abstracts.

Despite their many features, standard word processors do not address all of these professional needs. For example, although spell checking is common, flagging forbidden words or ensuring consistent use of particular terms must be done manually. Real-time counts of words and characters can be displayed for the whole document, but not for a single section.

Text editing was an active research topic in the 1980s, when personal computers and word processors first became mainstream. Although current research focuses on 'new' technology, we argue that document editing should also remain a topic of central importance, as it touches the lives of hundreds of millions of users. We take a fresh look, seeking to apply modern interaction design principles to increase the power of expression, while preserving simplicity of interaction.

We focus on a group of 'extreme' users—authors of technical documents—and seek to answer the following questions:

1. How do current software tools support professional technical writers?
2. How do professional users manage constraints and consistency when editing technical documents?
3. How can we create tools that better support these needs?

After reviewing the literature, we describe an interview study with contract and patent writers and highlight the problems they face in their professional editing tasks. We then introduce the concept of *Textlet*, which reifies the notion of selection in text documents. We show examples of novel tools based on this concept to address the needs identified in the study. Next, we describe the design of two prototypes that implement different types of textlets, and report on the use of one prototype as a technology probe to better understand how textlets support selective search and replacement of text. We conclude by arguing that *Textlets* can serve as a generative concept for creating powerful new tools for document editing.

©ACM, 2020. This is the author's version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version was published in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems CHI 2020, April 25 – 30, 2020, Honolulu, HI, USA.*
ISBN 978-1-4503-6708-0/20/04.
http://dx.doi.org/10.1145/3313831.3376804

RELATED WORK

We review research related to both word processing and code editing tools and practices. The latter is a particularly interesting form of technical document that requires professional software developers to manage multiple internal constraints, and the specific tools developed to ensure internal consistency in code text may inform our design. We also discuss the theoretical foundations underlying the design of *Textlets*.

Text Editing Practices

Text editing was an active research topic in the 1980s when word processors became mainstream. For example, Card et al. [11] modeled expert users' behavior in manuscript-editing tasks; Tyler et al. [49] investigated the acquisition of text editing skills; and Rosson [42] explored the effects of experience on real-world editing behavior. Others examined paper-based editing practices to improve computer-based text editing [44, 40, 33] and collaborative writing [2, 14, 39].

More recent studies identify issues with modern word processors. For example, Srgaard et al. [45] found that users rarely take advantage of text styles, and argue that this is because styles do not impose restrictions on the document structure. Alexander et al. [1] found that although users often revisit document locations, they seldom use the specific revisitation tools found in Microsoft Word and Adobe Reader. Chapuis et al. [12] examined users' frustration with unexpected copy-paste results due to format conversion. This work identifies a clear mismatch between the advanced features offered by modern word processors and actual user practice, and highlights the need for new tools and concepts. While the above work focuses on general editing tasks, we are particularly interested in how authors manage constraints and ensure consistency when editing structured technical documents.

Tools to Support Text Editing

Researchers have created a variety of text editing tools to support annotation [53, 51, 43], navigation [1, 30, 50] and formatting [38]; as well as distributing editing tasks [7, 47] and taking advantage of a text's structure [36]. We focus here on copy-paste [46, 8], and search-and-replace [35, 3], both especially relevant to supporting internal document consistency.

Chapuis et al. [12] propose new window management techniques to facilitate copy-paste tasks. Citrine [46] extracts structure from text, e.g. an address with different components, that can be pasted with a single operation. Multiple selection [37] offers smart copy-paste that is sensitive to source and destination selections, while Entity Quick Click [8] extracts information to reduce cursor travel and number of clicks. Cluster-based search-and-replace [35] groups occurrences by similarity, allowing entire clusters to be replaced at once. Beaudouin-Lafon's [3] instrumental search-and-replace tool highlights all items at once, so users can make changes in any order, not only as they occur in the document.

Commercial applications such as *Grammarly*[1] check grammar and spelling by suggesting alternative wording, style and tone, among other features. However they do not ensure consistent use of specific terms, e.g. always referring to a party in a contract with a single name. Other software tools automatically generate consistent references, including *Mendeley*[2] and *EndNote*[3] for researchers, and *Exhibit Manager*[4] for legal professionals. Although automated reference management solves some problems, users still lack flexibility for others, e.g. creating a custom citation format. These tools are separate from the word processor, potentially distracting users from their documents and fragmenting workflow. Our goal, then, is to create unified tools that support user-defined constraints and ensure consistency in text documents.

Code Editing Practices

Code editing has been widely studied, especially copy-paste [26, 24], use of online resources [9] and drawings [13], and performing maintenance tasks [27]. A key challenge that emerges from these studies is how to manage dependencies. For example, Kim et al. [26] found that programmers rely on their memory of copy-pasted dependencies when they apply changes to duplicated code. Ko et al. [27] identified both 'direct' dependencies, e.g. going from a variable's use to its declaration, and 'indirect' ones, e.g. going from a variable's use to the method that computed its most recent value, and proposed ways of visualizing these dependencies in the editor. While technical document constraints are less stringent than in computer code, we hope to exploit certain commonalities.

Tools to Support Code Editing

We see program code as an extreme case of a technical document, with many internal constraints. For example, Toomim et al.'s [48] technique supports editing duplicated code and visualizing links among duplicates. To help programmers use web examples more efficiently, *Codelets* [41] treat snippets of code examples as 'first-class' objects in the editor, even after they are pasted into the code. Kery et al.'s [25] tool for lightweight local versioning supports programmers performing exploratory tasks, while *AZURITE* [52] lets programmers selectively undo fine-grained code changes made in the editor. *Barista* [29] supports enriched representations of program code, while *Whyline* [28] and *HelpMeOut* [20] support debugging tasks. Our challenge is how to build upon these concepts and tools but for non-programmers who manage less highly constrained technical documents.

Theoretical Foundations

We seek to create generic tools rather than ad hoc solutions, which requires adopting a principled design approach. Beaudouin-Lafon's [3] Instrumental Interaction model extends and generalizes the principles of direct manipulation, and is operationalized by three design principles [5]: *Reification* turns commands into first class objects or instruments; *polymorphism* applies instruments to different types of objects; and *reuse* makes both user input and system output accessible for later use. Although they have been explored in the context of graphical editing [15, 34]; our focus here is on text editing.

[1] https://grammarly.com

[2] https://mendeley.com

[3] https://endnote.com

[4] https://exhibitmanager.com

STUDY 1: INTERVIEW WITH LEGAL PROFESSIONALS

Editing technical documents requires a complex editing process [16], especially to maintain the document's constraints and internal consistency [18]. We conducted critical object interviews [32] to better understand how professionals manage such constraints and consistency in their technical documents.

Participants

We interviewed 12 participants (three women, nine men; aged 24-50). Their occupations include: contract manager, legal affairs director, candidate to a Ph.D. in law, lawyer, patent attorney, and patent engineer. All use Microsoft Word on either the Windows (11/12) or MacOS (1/12) platforms; only one uses the latest 2019 version.

Procedure

All interviews were conducted in English, each lasting from 45-60 minutes. We ran four pilot interviews with colleagues to establish the protocol, then visited participants in their offices and asked them to show us specific examples of their current digital and physical documents. We asked them to describe a recent, memorable event related to editing that document, either positive or negative. The first two authors conducted all interviews, alternating asking questions.

Data Collection

All interviews were audio recorded and transcribed. We also took hand-written notes. We were not allowed to videotape or take photographs for confidentiality reasons.

Data Analysis

We analyzed the interviews using reflexive thematic analysis [10]. We generated codes and themes both inductively (bottom-up) and deductively (top-down), looking for breakdowns, workarounds and user innovations. After interviewing eight participants, the first two authors conducted the first analysis together, grouping codes into larger categories and focusing on participants' editing behavior. We discussed any disagreements and rechecked the interview transcripts to reach a shared understanding. We also created story portraits [23] to graphically code the data, which helped us engage with the collected data and resolve disagreements. We arrived at the final themes after three iterations.

RESULTS AND DISCUSSION

We identified six themes: maintaining term consistency, managing dependencies by hand, reusing content, visiting and revisiting document locations, managing annotations, and collaboration.

Maintaining term consistency

All participants rely on their memories to maintain consistency across document terms, which are often defined the beginning of the document. This causes problems, e.g. when P7 (legal affair director) struggled to use the full name of a party across the document and P5 (patent attorney) often made the wrong choice between two words with highly similar meanings.

Sometimes terms must be changed, e.g. shifting from British to American English or if the client prefers another word. To avoid introducing inconsistencies, lawyers must update each term and its variations, e.g. singular vs. plural, and adjust verbs (P1), articles (P1,6,7,9) and pronouns (P9) accordingly.

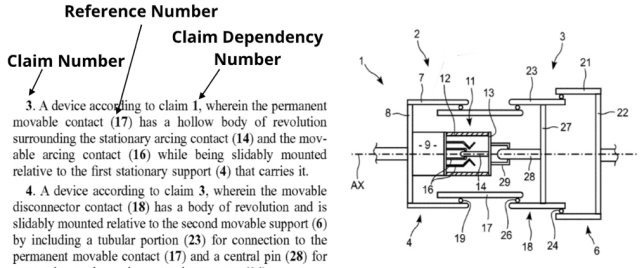


Figure 1. Patent claims use three different numbering systems (left). Patent illustration (right).

Although all participants use "search and replace" to make consistent edits, most (9/12) avoid "replace all": *"It is too risky."* (P4); *"I will not let the computer do it for me."* (P6); and *"I prefer to do it manually."* (P5). Instead, they use a one-by-one search-navigate-check-replace strategy to manually replace each term. They ensure correctness by viewing and assessing each term's context: *"We have to conjugate the verb with the subject. It's like a lot [of work]." (P4)* Checking context is also essential for avoiding partial matches, i.e. when the search term matches a subset of a longer word (P3,11), which requires performing additional search-and-replace operations.

In summary, participants maintain consistency across terms primarily by hand, which they find cumbersome and prone to error. Most avoid "replace all" because they do not trust the results and cannot easily check them.

Managing dependencies by hand

We define a dependency as two or more sections of text that must be kept identical or consistent. Most participants (8/12) rely on their memories to manage document dependencies, and synchronize them by hand. We identified three types of dependency problems: managing consistency across pasted copies, numbering items, and managing cross-references.

All patent attorneys (4/4) copy text from the *Claims* section to the *Summary of the Invention* when drafting the latter. However, when they change the claims, they often forget to update the summary accordingly: *"Because it is not automatically updated with the claims, I can easily forget to update."(P6).*

Patents contain three types of numbering systems (Fig. 1): claim number, claim dependency number (when the current claim depends upon another claim), and reference number (to specific parts of the illustration). Most patent attorneys (3/4) manage these numbers by hand instead of using Word's cross-reference feature, typically leaving a gap between consecutive references. This lets them add additional numbers later, while ensuring that the reference numbers remain in ascending order.

Most lawyers and patent writers insist on maintaining full control of the text, especially the critically important claims section, even if the process is tedious. Even participants who are comfortable using automatic features do not rely on automatic numbering: P7 said: *"most of the time, I prefer if*

something can be automatically achieved" yet avoids automatic numbering: *"I cannot really tell you why. One reason might be that if I have automatic numbering set up, this would have become paragraph 2 and all the numbering of the claims would have been changed...I would not be very happy."*.

In summary, their key reasons for avoiding automatic numbering include 1) their inability to differentiate automatic from normal numbering, unless they select the text; 2) incorrect display of references, e.g. when items are added to a list, until a manual update is triggered; and 3) invisibility of dependencies after an update, since they lack feedback and cannot be sure if the changes are correct.

Reusing content

All participants reuse previous document elements to create new documents, incorporating text, styles and templates. When copy-pasting a piece of text for reuse, they must often edit the content between copy and paste operations or adapt the format after pasting, e.g. using the brush tool (P6) or a macro (P7). If visual formatting results from applying a style, pasting new text can bring "bad" styles into the document and pollute existing styles: *"When you copy-paste into a document, you can import the style of the [original] document. Too many unnecessary styles makes the document heavier and you have to remember which style to use. This is a mess." (P4)*

Most participants (10/12) use templates to create new documents, including pre-written text, preset styles or both. Although useful for writing letters, filling cover pages, generating tables and managing formatting consistency, participants still struggle with formatting issues caused by style conflicts. In summary, they often reuse content, but are not satisfied with the corresponding introduction of format inconsistencies.

Visiting and revisiting document locations

Participants rarely write or edit documents sequentially and often revisit different parts of the document. For example, P7 created a set of keyboard shortcuts to *"jump to different parts of the document"* because he needs to switch often. This is consistent with Alexander et al.'s findings concerning users' revisitation behavior [1].

Participants also need better revisitation support when systematically going through the whole document, e.g. incorporating edits one by one or performing search-and-replace tasks. The latter often involves checking an earlier replacement, after the fact. Unfortunately, Word imposes sequential interaction, so users cannot return to the previous replacement: *"The problem is that I cannot check. It made the replacement and it goes to the next occurrence, so I don't see what just happened."* (P7). P8's workaround to address this problem involves turning on "track changes" to leave an inspectable trace of each replacement. In summary, participants experience problems navigating their documents, especially with respect to tracking recent or oft-visited parts of the document.

Managing annotations

Some participants (4/12) appropriate and customize their tools to support comments and annotation, rather than using the dedicated features of their word processor. For example, P5 uses footnotes to add comments for his clients because he dislikes how the text gets smaller when using Word's *Track Changes*. P7 avoids *Track Changes* altogether and uses different colors to encourage active reading and convey the importance of certain comments to his clients.

For documents with two or more co-authors, some participants (4/12) complained that the *Track Changes* feature introduces more problems than it solves (P2, P4) and makes it difficult to understand the modifications (P5, P7). Instead, some (3/12) use the comparison function *after* making changes, to make modifications visible to their clients.

Interestingly, P9 also used the comparison function to 'cheat': He modified the document with *Track Changes* on a Saturday night but did not want his client to know he worked over the week-end. So he accepted all the changes and then compared it to the original document on Monday morning, making it appear that the changes had been made on Monday. In summary, participants find annotation tools frustrating and constraining, and some creatively use other features to meet their needs.

Collaboration

Most participants (11/12) collaboratively edit documents. We categorize their collaboration strategy as branching (versioning and partitioning) and merging.

When versioning, participants exchange documents via email and save successive versions to keep track of changes made to the document. They use simple suffixes to identify versions over email, e.g. V1, V2, V3, so documents with similar content hang around and are hard to find again. P12 complained that she created eight versions of the same document even though she made only minor changes. The notion of File Biography [31] could help them manage these issues. Local versioning, explored for code editing in Variolite [25], would also be useful but standard word processors do not support it.

Some participants partition the master document for co-authors to edit and merge it later. The problem in the merge stage is style pollution, as discussed above, due to foreign styles being imported through copy-paste (P4) or forgetting to format text (P2). Because the style panel in Microsoft Word is not displayed by default when users open a document, it is often hidden from users. As a result, formatting and style inconsistencies are often undetected.

When a version of a document is sent out and then returns with proposed changes, participants have to merge these changes into the master document. Even though they use the *Track Changes* feature of Microsoft Word, they usually make the changes by hand, going through each document and deciding which edits to incorporate. They do not accept all the changes for various reasons: *"It might destroy the way [the text] was presented"* (P5), *"We do not consider all comments"* (P6), *"[clients'] comments are difficult to understand"* (P7), or the changes require other modifications to be made in other parts of the text (P7). In summary, we found that participants manually version their documents, even for minor edits, and merge documents by hand, incorporating changes one by one, as they struggle with style pollution.

Summary

Study 1 shows not only that professional technical writers must maintain consistent use of terms, but also that they manage the resulting dependencies mostly by hand. They struggle to maintain formatting consistency when reusing text and lack tools for keeping track of their navigation within their document, flexibly generating annotations, and collaborating asynchronously. Based on these results and the theoretical framework provided by Instrumental Interaction [3], we propose a general solution to address some of their needs.

TEXTLETS: REIFYING SELECTION

General-purpose word processors such as Microsoft Word have hundreds of features. As we saw in *Study 1*, even when users know that a feature exists, such as 'replace all' or 'automatic numbering', they often prefer making changes by hand to stay in control. Rather than proposing specific new features to address the various use cases we observed, we seek a general approach that fits how they actually deal with text.

Word processors rely heavily on the concept of selection: the user selects a piece of text and then invokes a command using a menu, toolbar or keyboard shortcut that affects the content of the selection. However, the selection is transient: selecting a new piece of text causes the previous selection to be lost.

We introduce the concept of *textlet* as the *reification* [5] of a text selection into a persistent, interactive, first-class object. A textlet represents a piece of the text document identified as interesting to the user. They can be highlighted in the document itself, listed in a side panel, or visualized through other interface elements, e.g. a scrollbar, for easy access.

To create a textlet, a user simply selects a piece of text and invokes a command, e.g. `Ctrl+T`. The selected text is highlighted and the textlet is listed in the side panel where a behavior (see below) can be assigned to it.

Textlets can also be created automatically by a higher-level object called a *grouplet*. For example, to create textlets that represent all the occurrences of a word in the document, the user creates a *search grouplet* (or *searchlet*), e.g. with the traditional Find command `Ctrl+F`. The *searchlet* appears in the side panel and the user can type the search string. A textlet is automatically created for each match of the search string and appears as an item underneath the *searchlet*. This list is automatically updated when editing the document or when changing the search string.

The power of textlets comes from the *behaviors* associated with them. The most basic behavior is to (re)select the piece of text from the textlet, e.g. by double-clicking the textlet representation in the side panel. Other behaviors include the ability to change or automatically generate the content of the text, to change its style, and to attach annotations or additional information, such as character or word count. Creating textlets with different behaviors leverages the power of polymorphism [5] because a single concept (reified text selection) addresses a variety of commands (searching, counting, referencing), providing users with a unifying concept to manage text documents. This slightly extends the definition in [5], which focused on polymorphic instruments.

The rest of this section illustrates the power of textlets by describing how different behaviors can address some of the issues observed in *Study 1*. Table 1 summarizes the use cases and the solutions we have implemented.

Textlets for Consistent Reuse

Study 1 showed that technical writers often reuse portions of text or entire templates when creating new documents. They rely on copy-paste to incorporate parts of other documents, but this requires precisely (re)selecting the text to be copied.

With textlets, users can create text snippets specifically for reuse, such as common vocabulary and phrases, list templates, or pre-written paragraphs with placeholders. Reusing a snippet simply involves a drag-and-drop or click-based interaction with the textlet. Placeholders can themselves be textlets to highlight the parts that need to be filled in, so that they can be easily identified, selected, and replaced with the proper text.

These snippets can be collected in dedicated documents or embedded into other documents. *Study 1* identified collaborative practices where users share a set of constraints and consistency criteria. By collecting reusable textlets in separate documents, they can easily share these documents and facilitate consistency across users and documents.

Textlets for Term Consistency

We observed that technical writers need to go back and forth in their documents to check for consistency or make consistent changes across the document. To that end, they often use the search command, but they do not trust the search-and-replace tool enough to perform replace-all actions blindly, and prefer to check the term and its context before each replacement.

Searchlets, briefly introduced earlier, can address these use cases by automatically searching for all the occurrences of a text in the document. A *searchlet* is a *grouplet* that creates *occurrence textlets* for each match they find in the document. These occurrences are listed under the *searchlet* in a side panel and automatically updated when the document changes. This supports fast navigation to each occurrence in the document, e.g. with a click on the occurrence in the side panel.

Searchlets support flexible search-and-replace. After specifying a replacement text for the *searchlet*, the user can replace all occurrences at once, or replace them one by one, in any order. At any time, including after a replace-all, it is possible to revert individual occurrences, giving users full control and visibility over their actions. Multiple *searchlets* can be active simultaneously, so that users can keep earlier searches around and get back to them later.

When users navigate the document to check for consistency and to make changes, they often lose track of where they were when they started the check. *Searchlets* facilitate navigation among occurrences, but do not address the need for location tracking in the document. Building on previous work such as Read Wear [21, 1] and Footprints [50], a *history grouplet* can record recent selections and let the user navigate among them. Previous selections can appear as individual textlets in a side panel or, to save space, the *grouplet* can display arrows to navigate the history of selections.

Use Case	Issue	Solution
Consistent Reuse	Recurrent copy-paste to start new documents from scratch requires re-selecting the text in one or more documents.	All textlets save their text, which can be reused using simple actions such as drag-and-drop.
Term Consistency	Repeatedly navigating across a document using search terms leaves no traces of scroll positions, making it hard to go back and forth.	*Searchlets* create *occurrence textlets* that let users navigate by interacting directly with them on the side panel.
Reference Consistency	Automated numbered lists and cross-references take control away from users. Numbered items and references do not update automatically.	*Numberlets* are counters that can be manipulated and applied to numbered lists, sections, figures, etc. References to *numberlets* can be created by copy-pasting them in the document. Item numbers and references are always up to date.
Length Constraints	Standard word processors require selecting text each time to count words in a specific area and get other metrics.	*Countlets* add a persistent decoration to the text of interest that displays a word count and updates it as users edit the content.
Exploratory Writing	Keeping track of alternatives is difficult. Undo/redo is not adapted to go back and forth between versions.	*Variantlets* store alternative versions of textlets that can be easily retrieved, compared and edited.

Table 1. How different textlet behaviors address some issues observed in *Study 1*.

Textlets for Reference Consistency

Standard word processors include tools for managing certain types of dependencies automatically, most notably numbered lists and cross-references. *Study 1* showed that participants distrust and struggle with automatically numbered lists, and thus avoid automated cross-reference management tools.

Documents often include numbered items such as sections, figures, patent claims or references. Both the numbered items and the references are good candidates for textlets: Both are *computed textlets*, i.e. their content is computed and updated as the document changes, but the user can still interact with them. A *numberlet* is a *grouplet* that creates numbered items and ensures that the number sequence matches the document's item order. Each numbered item is itself a *grouplet* for creating and managing textlets representing references to that item.

Numberlets, numbered items and references can be listed in the side panel for easy navigation. Creating new numbered items and new references involves a simple drag-and-drop or clicking on the corresponding textlet.

This design may seem complex compared to the automatic numbering and cross-referencing features of standard word processors, but it leaves users in control by turning numbered items and references into objects that they can see and manipulate while the system maintains consistency during document editing. It is also more powerful and flexible than the predefined types of references offered by standard word processors. For example, Microsoft Word 16 for Mac can cross-reference *Headings*, *Bookmarks*, *Footnotes*, *Endnotes*, *Equations*, *Figures* and *Tables*, but not *Articles* or *Claims*, which are used extensively by contract and patent writers. *Numberlets* let users control what types of numbered items they need, providing flexibility within a unified interface.

Textlets for Length Constraints

Word count and character count limits are common in technical documents. For example, patent offices limit the number of claims in a patent, the number of words in the abstract, and the number of characters in the patent title. Standard word processors include tools to count words and characters in a selection, but they require users to reselect the text and recount after every modification. Microsoft Word shows the total word count of the entire document and current selection in real time, but counting the characters in, e.g. a section of the document requires selecting the text and bringing up a modal dialog.

Counting textlets, or *countlets*, solve this problem by counting the number of words or characters in a segment of the document and displaying it in the document itself and/or a side panel. As the user edits the text, the counter updates, avoiding the need for special commands or re-selection. The user can set a threshold above which the textlet will signal that the text is too long. Additional metrics could easily be included, such as the text's estimated reading time. Such *timelets* would be useful, e.g. for journalists and authors of video subtitles.

Textlets for Exploratory Writing

Study 1 showed how professional technical writers often need to manage multiple alternatives for parts of a document, before deciding or agreeing on which one to keep. Although standard word processors support change tracking, this is insufficient, since it tracks all edits, not the intermediate versions the user may want to keep. Participants must either make copies of the entire document, or use colored text or comments to list alternatives within the document.

Variant textlets, or *variantlets*, let users keep track of the changes made to a selection rather than the entire document. We were inspired by *Explorable Multiverse Analyses* [17], where alternative analyses can be embedded in a research paper and selected by the reader to view them in context. A *variantlet* saves the original content of the selected text. After editing the text, the user can swap it with the original version for immediate comparison, and swap again with the edited version. More sophisticated behaviors can be added to manage multiple alternatives, such as displaying the alternatives side by side or displaying the changes in a manner similar to the track changes mode of word processors. *Variantlets* provide greater control on version management by supporting local

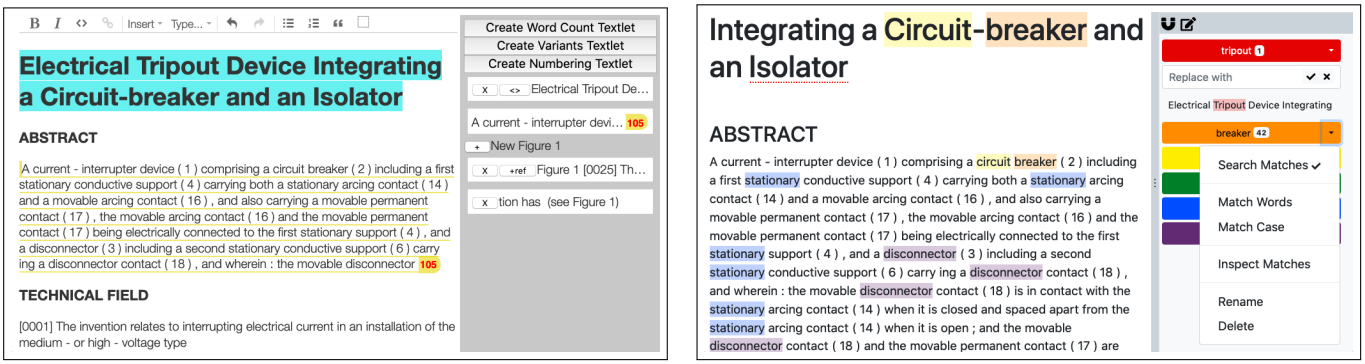


Figure 2. First prototype with the side panel showing a *variantlet*, a *countlet* and a *numberlet* containing a numbered item and a reference, and their visualization in the document (left). Second prototype with multiple *searchlets* that highlight corresponding occurrences in the document (right).

versioning rather than traditional document-level versioning. A similar concept is featured in *Variolite* [25] for code editing.

Generative Power

The previous examples show the power of textlets to support a variety of tasks. We have also identified other behaviors for textlets that could be useful for a wider range of use cases:

- Attaching comments, summaries, translations, word-scale graphics [19] or emojis and adding decorations to a textlet, e.g. highlighting or badges, to annotate the document;
- Supporting arbitrary computed content, such as Victor's Reactive Documents[5], where a textlet is defined by a formula that refers to other textlets, as in a spreadsheet;
- Controlling the style and formatting of the text by associating style attributes with the textlet;
- Crowdsourcing the text of a textlet or a collection of textlets for reviewing or grammar checking, as in Soylent [7]; and
- Organizing textlets freely in a canvas to help analyze or annotate the content of a document

The generative power [4] of textlets comes from the combination of a set of behaviors:

- Navigating to the text of the textlet in the document;
- Selecting the text of the textlet, leveraging all the existing commands that act on the selection;
- Replacing/modifying text either based on user edits or automatically;
- Modifying the style of the text;
- Adding decorations that are not part of the text itself; and
- Representing and manipulating textlets in a separate view, such as a list in a side panel.

Generative power also comes from the ability to create textlets not only directly, by selecting text in the document, but also automatically, by using *grouplets* that identify and live-update a set of matching textlets. *Grouplets* let users deal with dynamic collections of text in a concrete way, whereas standard word processors typically offer advanced, specialized commands that users hesitate to learn and use. Although textlets may involve more actions than these specialized commands, we argue that users are more likely to try them, and will save time compared to the manual solutions users resort to.

[5] http://worrydream.com/Tangle/

PROOF-OF-CONCEPT IMPLEMENTATION

In order to demonstrate the concept of textlets, we created a proof-of-concept implementation with four types of textlets: word count (*countlets*), text variants (*variantlets*), numbered references (*numberlets*), and search-and-replace (*searchlets*). These textlets address multiple use cases described in *Study 1*.

We created two prototypes as plugins to the ProseMirror[6] web-based word processing toolkit. The first prototype (Fig. 2a) was developed internally as our first proof of concept and implements *countlets*, *variantlets* and *numberlets*. The second prototype (Fig. 2b) implements *searchlets* and was developed in an iterative process with the participants in *Study 2*, where it was used as a technology probe [22].

Overall Interface

The main window contains the text document, with a traditional toolbar for basic formatting at the top, and a side panel dedicated to textlets on the right. The panel features a toolbar for creating new textlets and the list of textlets themselves. It also features *grouplets*, with their list of textlets below them. A textlet is created using any of three techniques:

a) Selecting the text content in the document and clicking a creation tool in the toolbar;
b) Clicking a creation tool in the toolbar and selecting the text content in the document; or
c) Entering a keyboard shortcut.

These techniques are also used to create *grouplets*, depending upon their type: some *grouplets* require a text selection, others not, and some may require additional information. Each textlet has a context menu that lets users navigate to the original text in the document, select that text, and delete the textlet. The menu also contains textlet-specific behaviors, such as *search* and *inspector* for the *searchlet* (see below).

Countlets

Our implementation of *countlets* (Fig. 3) decorates the selected text with a handle at each end. These handles let users change the scope of the textlet. The right handle also displays the word count of the text in the textlet, which is updated in real time as the user edits the content. A right-click on the *countlet* lets users set a threshold. The counter is displayed in red

[6] http://prosemirror.net

when its value is higher than the threshold. Deleting the textlet simply removes the word count.

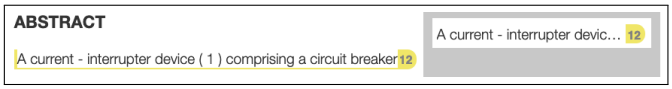


Figure 3. *Countlet*: a textlet for counting words.

Variantlets

Our implementation of *variantlets* (Fig. 4) supports a single alternative text. When the user creates the *variantlet*, its content is stored. The user can edit the content, and swap it with the stored one by clicking a button in the side panel representation of the *variantlet*. The user can thus easily view and edit the two variants. Combining a *variantlet* with a *countlet* lets the user instantly compare the two lengths by switching between the two alternatives. A more complete implementation of the *variantlet* should include an additional button to save additional versions and a way to navigate through the versions and swap any one of them with the selection.

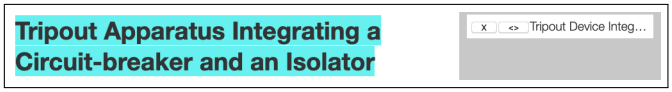


Figure 4. *Variantlet*: a textlet for editing local versions.

Numberlets

Our implementation of *numberlets* (Fig. 5) uses *grouplets* to create counters, new numbered items for a given counter, and new references to a given numbered item. The user creates a new counter by selecting a piece of text that contains a number or a hash sign (#), e.g. Article #. This text serves as a template for the numbering scheme. The new counter appears in the side panel as a button. Clicking this button inserts a new numbered item (the *numberlet*) at the cursor position, with the proper number. This *numberlet* is added to the side panel and is also a *grouplet*: clicking it inserts a reference to that item in the text at the cursor position, as well as the corresponding *reference* textlet (or *reflet*) in the side panel.

Numbered items and references are updated when the content of the document changes. The numbering of items follows their order of appearance in the document, and is therefore updated when moving text around. If a numbered item is removed and there are dangling references to it, these references show the error. All updates are immediately visible in both the text and the side panel, ensuring consistent numbering at all times. An additional feature (not implemented) should let users drag a reference textlet below another numbered item to change the reference to that item. This would make it possible to re-attach dangling references.

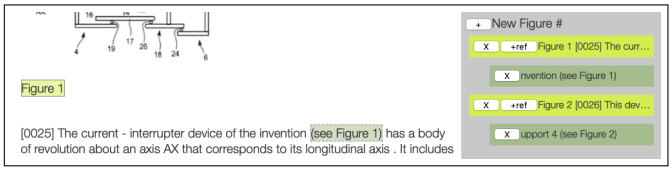


Figure 5. *Numberlet*: a textlet for numbering and referencing.

Searchlets

Our implementation of *searchlets* (Fig. 6) supports flexible search and replace by extending Beaudouin-Lafon's search-and-replace instrument [3]. A *searchlet* is created by clicking the creation tool then specifying the search text, or selecting the search text in the document and clicking the creation tool. Users can also create a blank *searchlet* and then enter the search string. Enabling the *search* behavior finds all occurrences of the search text, highlights them in the document and displays the number of occurrence in the panel. The usual "word matching" and "case sensitive" options become available in the menu to refine the search (Fig. 2b).

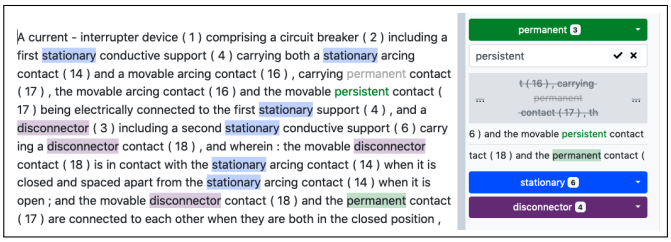


Figure 6. *Searchlet*: a textlet for searching and replacing text.

Navigating Occurrences

Enabling the *inspector* behavior generates the list of occurrences below the *searchlet* in the panel, highlights them in the document, and gives access to the replace capability (Fig. 6). Changing the search string or the search settings re-runs the search and updates the list of occurrences underneath it. Editing the document also dynamically updates the list of occurrences: typing the searched text in the document creates a new occurrence and changing the text of an occurrence in the document removes it from the list of textlets if it does not match anymore.

Each occurrence is a textlet that displays the text surrounding the match in the document and updates it in real time. An occurrence can be expanded by clicking it to better show the context (Fig. 6). The user can then click the ellipsis buttons to show more context.

Occurrences can be moved, including under another *searchlet*, giving users flexibility to organize the search results as they see fit. For example, occurrences of different mispellings of a word can be identified with different *searchlets* and then grouped under one *searchlet*, after which they can all be replaced at once. When moved, occurrences adopt the color of their new host *searchlet*. They also "belong" to their new host for the purposes of the *replace-all* action. In the current implementation, they disappear from the list when the search string or the search settings of the new host are changed.

Replacing Text

Selecting "Replace Matches" in the *searchlet* context menu (Fig. 2b) shows a text input field for typing a replace string and a button for replacing all occurrences in the list. Each occurrence textlet also includes three buttons that: replace only that occurrence, revert to the previous text, or ignore this occurrence from future replace-all operations. These actions can also be performed in the document itself using keyboard shortcuts.

Replaced occurrences stay in the textlet's occurrence list until a new search string is entered for that *searchlet*. This lets users work with the occurrences and make changes to the document after they perform a replace operation without losing track of the positions that were originally matched.

Searchlets extend Beaudouin-Lafon's previous work [3] by supporting multiple simultaneous searches. Each occurrence is reified as an item in the side panel, which supports additional functions such as disabling an occurrence in a global replace, or moving an occurrence to another *searchlet*. Our design is also grounded in our observations of the real-world challenges experienced by a group of professional users.

STUDY 2: SEARCH-AND-REPLACE TEXTLET

We used our second prototype as a technology probe [22] to evaluate *searchlets* with an observational study. We did not run a comparative study with, e.g., Microsoft Word as a baseline because many features that we implemented do not exist in Word or are clearly faster, e.g., a persistently-displayed character count with *countlets*, versus highlighting text and invoking Word's word count command.

Our goals were to gather feedback, identify potential novel and unexpected uses, and discuss new ideas with the participants in order to refine our design. The study focused on *searchlets*, but we also showed the participants the other textlets from the first prototype. We incorporated suggestions incrementally so that successive participants used slightly different versions of the probe.

Participants

We recruited eight participants: three patent attorneys, one patent inventor (one woman, three men; aged 29-50 who use various versions of Microsoft Word) and four researchers (one woman, three men; aged 24-26 who use LaTeX). Three of the patent attorneys had participated in *Study 1*. We included researchers because we believe that textlets address the needs of a wider range of users than those in *Study 1* and authors of research articles must also manage consistency in their papers.

Apparatus

The prototype is a Web application accessed with the participant's choice browser on their own computer. We provided a 13" MacBook Pro laptop running macOS 10.14 and Firefox 68.0 for participants who did not have a computer at hand. We created two sets of documents to match the participant's background: two patents and two research papers.

Procedure

We started by describing the features of the *Textlets* prototype, and gave participants 10 minutes to experiment with it. We used a think-aloud protocol and asked participants to perform two similar tasks on two documents: one using the editor of their choice and the other using the *Textlets* prototype. We counterbalanced for document order across participants.

Each task consisted of three small exercises with increasing difficulty: 1) replace a word by another and then change it back; 2) replace a word by another but only in certain contexts; and 3) replace two words with similar meanings with another word, including all relevant variations. Thus replacing "mouse" with "rodent" also requires changing "mice" with "rodents".

The two tasks, each with three exercises, took approximately 20 minutes. After an interview, participants completed a short questionnaire. The session ended with a debriefing to identify additional use cases and discuss ideas for improvement. We also showed participants the *countlets* and *variantlets* from the first prototype, and asked them to describe scenarios for which they might be useful.

Data Collection

We recorded audio, took hand-written notes during the session, and collected the answers to the questionnaire.

Results and Discussion

All participants successfully interacted with the textlet prototype and found the tasks representative of their everyday work. The textlet side panel was "faster to use" (P1, P3). It avoids jumping to the main text (P1, P2, P3, P6), so that they can focus on the relevant document parts, thus reducing mental workload. Most participants (6/8) preferred making changes directly with a *searchlet* over Word's non-interactive side panel. Two participants (P1, P2) asked for even greater interactivity with *searchlets*, such as one-by-one replacement directly from the panel, which we added in a later version, and merging two *searchlets* to apply the same replacement to their occurrences. We added other small improvements based on participants' feedback, including better colors and icons, and decluttering the textlet interface by using a menu instead of a series of buttons.

Replace-all-then-correct Strategy

Most participants (7/8) used a one-by-one search-check-replace strategy with both Microsoft Word and LaTeX: They search for the word, go to each occurrence in the main document to check the context and then perform the replacement, either by clicking a button or retyping.

Participants used a different strategy with textlets, which we characterize as search-overview-replace. They started by creating one or more *searchlets*, scanned the overview of the occurrences to see the variations and assessed which ones to replace. P1 said: *"I can see immediately what variations are in the text [from the side panel]. So I see it will work by replacing all matches"*.

The combination of overview and contextual information around each match encouraged participants to spontaneously develop two different strategies for the final search-overview-replace step: Six participants used a replace-all-then-correct strategy, first replacing all occurrences, then checking each replacement in the overview list for errors, which they corrected either with the 'revert' button or by retyping in the document. The other two participants (P6, P7) used an ignore-replace-all strategy, first pressing the 'ignore' button to skip outliers, then applying 'replace-all', similar to the 'perfect selection'strategy in [35]. In summary, although participants were reluctant to use replace-all with their regular word processor, they felt comfortable using the *searchlets*' replace-all and quickly developed strategies for selective replacement.

Persistent Selection: Keep Track, Individual Undo
Although both Microsoft Word and TexWorks (LaTeXeditor) provide an overview list of all search occurrences, they do not track them by position. By contrast, *searchlets* create persistent occurrences that help users keep track of what happened. P5 felt more confident with the prototype, saying: *"Here (pointing at the side panel) I can see the changes in context. It helps me [and] reassures me that I did the right thing."*

Furthermore, *searchlets* let users check the results of their previous replacements. The overview of occurrences persists in the panel even as users edit the document. This differs from other word processors that clear the search whenever the user types in the document, which forces users to tediously re-enter the search text. For example, P3 said: *"I have this list of all the occurrences. When I want to do some replacements, I choose some of them and I keep the whole list that I can always check [in the side panel]. This is quite important...I do not need to proof-read the whole text."*

Because each occurrence is also a textlet with its own history of changes, it can be undone individually and ignored in a replace-all command while still remaining in the overview list. These novel behaviors contributed to most participants (6/8) spontaneously adopting a replace-all-then-correct strategy. For example, P3 said while performing a task: *"Maybe it is better to replace all and check the ones that do not work."*. This suggests that making changes persistent, visible and reversible increases users' trust in the system.

Representing Constraints
One participant suggested embedding a group of *searchlets* as *"a highlight [feature] for forbidden words."(P4)*, arguing that making co-authors aware of these words as the document circulates would help them maintain consistency and improve collaborative editing. Textlets can thus embody constraints and serve as an active guideline when embedded in a document.

Feedback for countlet and variantlet
Participants also described situations in which they wanted to use *countlets* and *variantlets*. For example, P3 wanted to count the words in patent abstracts: *"I think this could be very useful because many times you are going to count words and [the system] does not keep it."* P4 wanted to use *variantlets* as a local versioning tool: *"If you can version one paragraph [instead] of the whole document, it could be very useful. In that case, you can track which part you have changed."*

Scalability and Limitations
A potential limitation of our approach is scalability: Searchlets that generate large numbers of matches or large numbers of textlets and grouplets in the side panel could cause problems when dealing with large documents. We did not observe such problems during the study, probably due to its short-term nature. Several features mitigate scalability issues: users can collapse grouplets, e.g. search results, to save space, or disable them to remove highlighting in the main text. Scrolling between the document and the side panel could also be synchronized, and future textlets could combine behaviors, e.g. *countlet +variantlet*, to save space.

One participant found that *searchlets* might be less useful in simple cases with few matches or variations of the same word: *"[With] only 3 matches, I would like to change it directly in the main text"* (P3). Another participant wanted *searchlets* to support regular expressions. Both features could easily be supported in a future prototype.

Summary
This study demonstrated the value of *searchlets*, the most complex textlet we developed, as well as the potential of other textlets. By turning search matches into persistent objects that users can manipulate directly, users were willing to use functions, such as replace-all, that they otherwise avoid with traditional word processors. They also spontaneously devised novel strategies and appropriated the textlet concept in unexpected ways, such as embedding *searchlets* for forbidden words. This study provides evidence for the validity of the textlet concept, and encourages us to further develop and assess the textlets we have developed, as well as design new ones.

CONCLUSION AND FUTURE WORK
Writing technical documents frequently requires following constraints and consistently using domain-specific terms. We interviewed 12 legal professionals and showed that technical writers are reluctant to use advanced features of their word processors, and must instead rely on their memory to manage dependencies and maintain consistent vocabulary within their documents. We introduced a simple, immediately accessible but powerful concept called *Textlets*, interactive objects that reify text selections into persistent items.

Textlets are a deceptively simple but powerful idea, based on the premise that creating persistent, interactive objects to represent abstract or transient concepts such as the selection can empower users. We showed five use cases where textlets can be applied to support consistent reuse, term and reference consistency, word count constraint, and exploratory writing. We presented two prototypes that implement a proof-of-concept of four textlets, and used one as a technology probe to assess a search-and-replace textlet. All participants successfully used the prototype to perform advanced tasks, and most spontaneously generated a novel replace-all-then-correct strategy. Several also invented novel uses and ideas for new textlets.

Future work will focus on creating a more advanced prototype that can be tested in a longitudinal study. We also plan to design and evaluate new textlets for commenting, formatting and computing text. Our findings on collaboration practices also open the door to investigating the potential of textlets for collaborative work. Beyond the examples illustrated in this article, textlets offer a generative concept for creating powerful new tools for document editing.

ACKNOWLEDGMENTS
This work was partially supported by European Research Council (ERC) grant n° 695464 ONE: Unified Principles of Interaction.

REFERENCES

[1] Jason Alexander, Andy Cockburn, Stephen Fitchett, Carl Gutwin, and Saul Greenberg. 2009. Revisiting Read Wear: Analysis, Design, and Evaluation of a Footprints Scrollbar. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '09)*. ACM, New York, NY, USA, 1665–1674. DOI:http://dx.doi.org/10.1145/1518701.1518957

[2] Ronald M. Baecker, Dimitrios Nastos, Ilona R. Posner, and Kelly L. Mawby. 1993. The User-centered Iterative Design of Collaborative Writing Software. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems (CHI '93)*. ACM, New York, NY, USA, 399–405. DOI: http://dx.doi.org/10.1145/169059.169312

[3] Michel Beaudouin-Lafon. 2000. Instrumental Interaction: An Interaction Model for Designing post-WIMP User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '00)*. ACM, New York, NY, USA, 446–453. DOI: http://dx.doi.org/10.1145/332040.332473

[4] Michel Beaudouin-Lafon. 2004. Designing Interaction, Not Interfaces. In *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI '04)*. ACM, New York, NY, USA, 15–22. DOI: http://dx.doi.org/10.1145/989863.989865

[5] Michel Beaudouin-Lafon and Wendy E. Mackay. 2000. Reification, Polymorphism and Reuse: Three Principles for Designing Visual Interfaces. In *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI '00)*. ACM, New York, NY, USA, 102–109. DOI: http://dx.doi.org/10.1145/345513.345267

[6] Tim Bergin. 2006. The Origins of Word Processing Software for Personal Computers: 1976-1985. *Annals of the History of Computing, IEEE* 28 (11 2006), 32–47. DOI:http://dx.doi.org/10.1109/MAHC.2006.76

[7] Michael S. Bernstein, Greg Little, Robert C. Miller, Björn Hartmann, Mark S. Ackerman, David R. Karger, David Crowell, and Katrina Panovich. 2015. Soylent: A Word Processor with a Crowd Inside. *Commun. ACM* 58, 8 (July 2015), 85–94. DOI: http://dx.doi.org/10.1145/2791285

[8] Eric A. Bier, Edward W. Ishak, and Ed Chi. 2006. Entity Quick Click: Rapid Text Copying Based on Automatic Entity Extraction. In *CHI '06 Extended Abstracts on Human Factors in Computing Systems (CHI EA '06)*. ACM, New York, NY, USA, 562–567. DOI: http://dx.doi.org/10.1145/1125451.1125570

[9] Joel Brandt, Philip J. Guo, Joel Lewenstein, Mira Dontcheva, and Scott R. Klemmer. 2009. Two Studies of Opportunistic Programming: Interleaving Web Foraging, Learning, and Writing Code. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '09)*. ACM, New York, NY, USA, 1589–1598. DOI: http://dx.doi.org/10.1145/1518701.1518944

[10] Virginia Braun and Victoria Clarke. 2019. Reflecting on reflexive thematic analysis. *Qualitative Research in Sport, Exercise and Health* 11, 4 (2019), 589–597. DOI: http://dx.doi.org/10.1080/2159676X.2019.1628806

[11] S. K. Card, T. P. Moran, and A. Newell. 1987. Computer Text-editing: An Information-processing Analysis of a Routine Cognitive Skill. In *Human-computer Interaction*, R. M. Baecker and W. A. S. Buxton (Eds.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, Chapter Computer Text-editing: An Information-processing Analysis of a Routine Cognitive Skill, 219–240. http://dl.acm.org/citation.cfm?id=58076.58096

[12] Olivier Chapuis and Nicolas Roussel. 2007. Copy-and-paste Between Overlapping Windows. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '07)*. ACM, New York, NY, USA, 201–210. DOI: http://dx.doi.org/10.1145/1240624.1240657

[13] Mauro Cherubini, Gina Venolia, Rob DeLine, and Andrew J. Ko. 2007. Let's Go to the Whiteboard: How and Why Software Developers Use Drawings. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '07)*. ACM, New York, NY, USA, 557–566. DOI: http://dx.doi.org/10.1145/1240624.1240714

[14] Elizabeth F. Churchill, Jonathan Trevor, Sara Bly, Les Nelson, and Davor Cubranic. 2000. Anchored Conversations: Chatting in the Context of a Document. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '00)*. ACM, New York, NY, USA, 454–461. DOI: http://dx.doi.org/10.1145/332040.332475

[15] Marianela Ciolfi Felice, Nolwenn Maudet, Wendy E. Mackay, and Michel Beaudouin-Lafon. 2016. Beyond Snapping: Persistent, Tweakable Alignment and Distribution with StickyLines. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology (UIST '16)*. ACM, New York, NY, USA, 133–144. DOI: http://dx.doi.org/10.1145/2984511.2984577

[16] Andrew L Cohen, Debra Cash, and Michael J Muller. 1999. Awareness, planning and joint attention in collaborative writing: From fieldwork to design. *LOTUS CODE# 1999.02* (1999), 94–101.

[17] Pierre Dragicevic, Yvonne Jansen, Abhraneel Sarma, Matthew Kay, and Fanny Chevalier. 2019. Increasing the Transparency of Research Papers with Explorable Multiverse Analyses. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. ACM, New York, NY, USA, Article 65, 15 pages. DOI:http://dx.doi.org/10.1145/3290605.3300295

[18] David K Farkas. 1985. The concept of consistency in writing and editing. *Journal of Technical Writing and Communication* 15, 4 (1985), 353–364.

[19] P. Goffin, J. Boy, W. Willett, and P. Isenberg. 2017. An Exploratory Study of Word-Scale Graphics in Data-Rich Text Documents. *IEEE Transactions on Visualization and Computer Graphics* 23, 10 (Oct 2017), 2275–2287. DOI:http://dx.doi.org/10.1109/TVCG.2016.2618797

[20] Björn Hartmann, Daniel MacDougall, Joel Brandt, and Scott R. Klemmer. 2010. What Would Other Programmers Do: Suggesting Solutions to Error Messages. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10)*. ACM, New York, NY, USA, 1019–1028. DOI: http://dx.doi.org/10.1145/1753326.1753478

[21] William C. Hill, James D. Hollan, Dave Wroblewski, and Tim McCandless. 1992. Edit Wear and Read Wear. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '92)*. ACM, New York, NY, USA, 3–9. DOI: http://dx.doi.org/10.1145/142750.142751

[22] Hilary Hutchinson, Wendy Mackay, Bo Westerlund, Benjamin B. Bederson, Allison Druin, Catherine Plaisant, Michel Beaudouin-Lafon, Stéphane Conversy, Helen Evans, Heiko Hansen, Nicolas Roussel, and Björn Eiderbäck. 2003. Technology Probes: Inspiring Design for and with Families. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '03)*. ACM, New York, NY, USA, 17–24. DOI: http://dx.doi.org/10.1145/642611.642616

[23] Ghita Jalal, Nolwenn Maudet, and Wendy E. Mackay. 2015. Color Portraits: From Color Picking to Interacting with Color. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. ACM, New York, NY, USA, 4207–4216. DOI:http://dx.doi.org/10.1145/2702123.2702173

[24] Cory J. Kapser and Michael W. Godfrey. 2008. "Cloning Considered Harmful" Considered Harmful: Patterns of Cloning in Software. *Empirical Softw. Engg.* 13, 6 (Dec. 2008), 645–692. DOI: http://dx.doi.org/10.1007/s10664-008-9076-6

[25] Mary Beth Kery, Amber Horvath, and Brad Myers. 2017. Variolite: Supporting Exploratory Programming by Data Scientists. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 1265–1276. DOI:http://dx.doi.org/10.1145/3025453.3025626

[26] Miryung Kim, Lawrence Bergman, Tessa Lau, and David Notkin. 2004. An Ethnographic Study of Copy and Paste Programming Practices in OOPL. In *Proceedings of the 2004 International Symposium on Empirical Software Engineering (ISESE '04)*. IEEE Computer Society, Washington, DC, USA, 83–92. DOI: http://dx.doi.org/10.1109/ISESE.2004.10

[27] Andrew J. Ko, Htet Aung, and Brad A. Myers. 2005. Eliciting Design Requirements for Maintenance-oriented IDEs: A Detailed Study of Corrective and Perfective Maintenance Tasks. In *Proceedings of the 27th International Conference on Software Engineering (ICSE '05)*. ACM, New York, NY, USA, 126–135. DOI: http://dx.doi.org/10.1145/1062455.1062492

[28] Andrew J. Ko and Brad A. Myers. 2004. Designing the Whyline: A Debugging Interface for Asking Questions About Program Behavior. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '04)*. ACM, New York, NY, USA, 151–158. DOI: http://dx.doi.org/10.1145/985692.985712

[29] Andrew J. Ko and Brad A. Myers. 2006. Barista: An Implementation Framework for Enabling New Tools, Interaction Techniques and Views in Code Editors. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '06)*. ACM, New York, NY, USA, 387–396. DOI: http://dx.doi.org/10.1145/1124772.1124831

[30] Sari A. Laakso, Karri Pekka Laakso, and Asko J. Saura. 2000. Improved Scroll Bars. In *CHI '00 Extended Abstracts on Human Factors in Computing Systems (CHI EA '00)*. ACM, New York, NY, USA, 97–98. DOI: http://dx.doi.org/10.1145/633292.633350

[31] Siân E. Lindley, Gavin Smyth, Robert Corish, Anastasia Loukianov, Michael Golembewski, Ewa A. Luger, and Abigail Sellen. 2018. Exploring New Metaphors for a Networked World Through the File Biography. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM, New York, NY, USA, Article 118, 12 pages. DOI: http://dx.doi.org/10.1145/3173574.3173692

[32] Wendy E Mackay. 2002. Using video to support interaction design. *DVD Tutorial, CHI* 2, 5 (2002).

[33] Catherine C. Marshall. 1997. Annotation: From Paper Books to the Digital Library. In *Proceedings of the Second ACM International Conference on Digital Libraries (DL '97)*. ACM, New York, NY, USA, 131–140. DOI: http://dx.doi.org/10.1145/263690.263806

[34] Nolwenn Maudet, Ghita Jalal, Philip Tchernavskij, Michel Beaudouin-Lafon, and Wendy E. Mackay. 2017. Beyond Grids: Interactive Graphical Substrates to Structure Digital Layout. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 5053–5064. DOI: http://dx.doi.org/10.1145/3025453.3025718

[35] Robert C. Miller and Alisa M. Marshall. 2004. Cluster-based Find and Replace. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '04)*. ACM, New York, NY, USA, 57–64. DOI:http://dx.doi.org/10.1145/985692.985700

[36] Robert C. Miller and Brad A. Myers. 2002a. LAPIS: Smart Editing with Text Structure. In *CHI '02 Extended Abstracts on Human Factors in Computing Systems (CHI EA '02)*. ACM, New York, NY, USA, 496–497. DOI:http://dx.doi.org/10.1145/506443.506447

[37] Robert C. Miller and Brad A. Myers. 2002b. Multiple Selections in Smart Text Editing. In *Proceedings of the 7th International Conference on Intelligent User Interfaces (IUI '02)*. ACM, New York, NY, USA, 103–110. DOI: http://dx.doi.org/10.1145/502716.502734

[38] Brad A. Myers. 1991. Text Formatting by Demonstration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '91)*. ACM, New York, NY, USA, 251–256. DOI: http://dx.doi.org/10.1145/108844.108904

[39] Sylvie Noël and Jean-Marc Robert. 2004. Empirical Study on Collaborative Writing: What Do Co-authors Do, Use, and Like? *Comput. Supported Coop. Work* 13, 1 (Jan. 2004), 63–89. DOI: http://dx.doi.org/10.1023/B:COSU.0000014876.96003.be

[40] Kenton O'Hara and Abigail Sellen. 1997. A Comparison of Reading Paper and On-line Documents. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI '97)*. ACM, New York, NY, USA, 335–342. DOI: http://dx.doi.org/10.1145/258549.258787

[41] Stephen Oney and Joel Brandt. 2012. Codelets: Linking Interactive Documentation and Example Code in the Editor. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. ACM, New York, NY, USA, 2697–2706. DOI: http://dx.doi.org/10.1145/2207676.2208664

[42] Mary Beth Rosson. 1983. Patterns of Experience in Text Editing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '83)*. ACM, New York, NY, USA, 171–175. DOI: http://dx.doi.org/10.1145/800045.801604

[43] Bill N. Schilit, Gene Golovchinsky, and Morgan N. Price. 1998. Beyond Paper: Supporting Active Reading with Free Form Digital Ink Annotations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '98)*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 249–256. DOI: http://dx.doi.org/10.1145/274644.274680

[44] Abigail Sellen and Richard Harper. 1997. Paper As an Analytic Resource for the Design of New Technologies. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI '97)*. ACM, New York, NY, USA, 319–326. DOI: http://dx.doi.org/10.1145/258549.258780

[45] Pål Srgaard and Tone Irene Sandahl. 1997. Problems with Styles inWord Processing: AWeak Foundation for Electronic Publishing with SGML. In *Proceedings of the 30th Hawaii International Conference on System Sciences: Digital Documents - Volume 6 (HICSS '97)*. IEEE Computer Society, Washington, DC, USA, 137–. http://dl.acm.org/citation.cfm?id=938438.938857

[46] Jeffrey Stylos, Brad A. Myers, and Andrew Faulring. 2004. Citrine: Providing Intelligent Copy-and-paste. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology (UIST '04)*. ACM, New York, NY, USA, 185–188. DOI: http://dx.doi.org/10.1145/1029632.1029665

[47] Jaime Teevan, Shamsi T. Iqbal, and Curtis von Veh. 2016. Supporting Collaborative Writing with Microtasks. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. ACM, New York, NY, USA, 2657–2668. DOI: http://dx.doi.org/10.1145/2858036.2858108

[48] Michael Toomim, Andrew Begel, and Susan L. Graham. 2004. Managing Duplicated Code with Linked Editing. In *Proceedings of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing (VLHCC '04)*. IEEE Computer Society, Washington, DC, USA, 173–180. DOI: http://dx.doi.org/10.1109/VLHCC.2004.35

[49] Sherman W. Tyler, Steven Roth, and Timothy Post. 1982. The Acquisition of Text Editing Skills. In *Proceedings of the 1982 Conference on Human Factors in Computing Systems (CHI '82)*. ACM, New York, NY, USA, 324–325. DOI: http://dx.doi.org/10.1145/800049.801803

[50] Alan Wexelblat and Pattie Maes. 1999. Footprints: History-rich Tools for Information Foraging. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '99)*. ACM, New York, NY, USA, 270–277. DOI: http://dx.doi.org/10.1145/302979.303060

[51] Dongwook Yoon, Nicholas Chen, and François Guimbretière. 2013. TextTearing: Opening White Space for Digital Ink Annotation. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology (UIST '13)*. ACM, New York, NY, USA, 107–112. DOI: http://dx.doi.org/10.1145/2501988.2502036

[52] Young Seok Yoon and Brad A. Myers. 2015. Supporting Selective Undo in a Code Editor. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1 (ICSE '15)*. IEEE Press, Piscataway, NJ, USA, 223–233. http://dl.acm.org/citation.cfm?id=2818754.2818784

[53] Qixing Zheng, Kellogg Booth, and Joanna McGrenere. 2006. Co-authoring with Structured Annotations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '06)*. ACM, New York, NY, USA, 131–140. DOI: http://dx.doi.org/10.1145/1124772.1124794