

Sharing Is Scaring: Linking Cloud File-Sharing to Programming Language Semantics

Skyler Austen

Brown University
Providence, USA

skyler_austen@brown.edu

Shriram Krishnamurthi

Brown University
Providence, USA

shriram@brown.edu

Kathi Fisler

Brown University
Providence, USA

kfisler@cs.brown.edu

Abstract

Users often struggle with cloud file-sharing applications. Problems appear to arise not only from interface flaws, but also from misunderstanding the underlying semantics of operations like linking, attaching, downloading, and editing. We argue that these difficulties echo long-standing challenges in understanding concepts in programming languages like aliasing, copying, and mutation.

We begin to examine this connection through a formative user study investigating general users' understanding of file sharing. Our study casts known misconceptions from the programming-education literature into semantically-similar cloud file-sharing tasks. It also uses tasks that echo two kinds of analyses used in programming-education: tracing and programming. Our findings reveal widespread misunderstandings across several tasks.

We also develop a formal semantics of cloud file-sharing operations, reflecting copying, referencing, and mutating shared content. By explicating the semantics, we aim to provide a formal foundation for improving mental models, educational tools, and automated assistance. This semantics can support applications including trace checking, workflow synthesis, and interactive feedback.

CCS Concepts: • **Human-centered computing** → **Empirical studies in collaborative and social computing**; *Empirical studies in HCI*; **Collaborative and social computing systems and tools**; • **Software and its engineering** → **Semantics**; • **Social and professional topics** → **Computing literacy**.

Keywords: cloud file-sharing, end-user computing, formal semantics, human-computer interaction

ACM Reference Format:

Skyler Austen, Shriram Krishnamurthi, and Kathi Fisler. 2025. Sharing Is Scaring: Linking Cloud File-Sharing to Programming Language Semantics. In *Proceedings of the 2025 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! '25)*, October 12–18, 2025, Singapore, Singapore. ACM, New York, NY, USA, 22 pages. <https://doi.org/10.1145/3759429.3762621>

1 Motivation

Many of us who work with computers have been called in to support family members who are baffled and frustrated by the systems they use. These are not systems designed for highly specialized users—e.g., configuring the options for SSL [21]. Rather, they are basic, mass-market cloud file-sharing applications (cfs), like Google Docs or Dropbox.

This should be especially baffling. These systems are generally created by experts at large corporations (Google, Microsoft, etc.). Their creators are presumably motivated to create interfaces that are enabling rather than frustrating. Despite all that investment, these problems persist (section 3). Beyond user-interface issues, could there be *technical* reasons for this?

Here is an actual story (names altered) that illustrates this. We start by posing the situation we were asked to help with:

Alice and Bob want to collaborate on a flyer for a social event. They are more comfortable with Word than with cloud-based tools like Google Docs. They have both heard that Dropbox is a good way to share and jointly edit files.

Alice thus creates a draft of the flyer F on Dropbox. She then shares F with Bob. Bob makes a change using Word and informs Alice he has done so. Alice opens F and sees no change. Bob confirms he is editing a file that is in Dropbox. They have several baffling rounds of exchange.

Do you see the problem? No? Here's the critical question: How did Alice *share* F ? Alice dragged the file from the Folder interface into Google Mail. (Do you see it now?)

Let us take a moment to consider a seemingly unrelated topic: the semantics of conventional programming languages. There is a large body of work [11, 16, 17, 22, 33, 34, 42, 48, 50, 52, 55], ranging across many years, countries, and programming languages, that shows that students specifically



This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Onward! '25, Singapore, Singapore

© 2025 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2151-9/25/10

<https://doi.org/10.1145/3759429.3762621>

and programmers more generally struggle with standard language concepts such as aliasing and state. The recent work on the SMoL Tutor [33] consolidates this work and shows how even small programs can be quite challenging.

This is, of course, not unrelated at all! The problem above is that Alice very reasonably thought she was sharing “the file”, but what she was doing was making a *copy* of the file. Bob was downloading the copy and storing it in his Dropbox. His changes were being synchronized to Dropbox, but of course this file had nothing to do with *F* itself.

In other words, the problem has to do with sharing and modification of content. “Sharing” can mean creating an alias, but colloquially, can also include operations that make copies. In the absence of mutation, this conflation is mostly harmless. But just as we see in the literature on programming, the moment mutation enters the picture, the differences are exposed, and there are clear parallels between the problems programmers and CFS users face.

The CFS users are, of course, not “programming” in the conventional sense, but they are if we change our perspective a little. To put it in context, David Patterson referred to MapReduce [7] as the “first instruction” of the data-center computer [39]. That claim is from a *conventional programmer’s* perspective. The *user* also “programs”, but with different operations: creating documents, editing them, attaching them, linking to them, and so on. Such operations form the “end-user instruction set architecture” (EUISA).¹

This paper explores the mapping between misconceptions about programming-languages semantics and difficulties that end-users face when sharing and editing documents. Specifically, we observe the semantic analogies between sharing document links \leftrightarrow aliasing, downloading or attaching documents \leftrightarrow copying objects, and editing documents \leftrightarrow mutation. We term this the **Central Analogy**, and posit that it could explain some end-user difficulties. Concretely, the paper makes several contributions. It:

- Provides a catalog of problems we have seen in our “tech support” roles (section 2).
- Employs the central analogy to design instruments to investigate how users perform on document-sharing tasks (section 4).
- Runs a formative user study that shows that the central analogy has legs, and that the problems are not limited to our personal experience (section 5, section 6).
- Formalizes the EUISA (section 8.1).
- Discusses ways this semantics could be used to create different tools (section 8.3 and section 8.4).

Our goal, ultimately, is to apply the programming community’s hard-won knowledge about state, aliasing, and related operations to help end-users use CFS applications with fewer errors and less frustration. This paper marks the first step by

trying to confirm our hunch about the central analogy and proposing a formalization that could provide ways forward.

2 A Catalog of Woes

For many years we have served as informal technical support for family and friends and through public help sessions. During this time, we have recorded a catalog of actual problems that people have run into in practice and come to us for help with. We described one of these actual situations in the introduction; we present several more in fig. 1. It is important to note that these are cleaned-up accounts of *actual problems* for which *real people* sought help.

It is important to read this catalog with an empathetic mind. While some of the issues may be obvious to a technical reader, they do not account for the “naïve models” that most end-users have. Furthermore, some of the end-users we have helped are uncomfortable with technology in general. (We hesitate to report that many of the end-users are older people, because we do not believe this problem is limited to older adults. If anything, they may be more willing to ask for help, whereas middle-aged and younger people may feel they are expected to know how to function with technology and may therefore be reluctant to express their struggles.)

We also note in passing that the issues reported here were mostly encountered by people who were still in full control of their mental faculties. As we encounter people with memory loss effects, we see the potential for these issues to become significantly more problematic. We discuss work related to this in section 3 and in more detail in section 10.

3 Related Work

A robust body of computing-education research has shown that students struggle to reason about programming-language semantics, particularly around aliasing, references, mutation, and state [11, 16, 17, 22, 33, 34, 42, 48, 50, 52, 55]. The SMoL Tutor [33] provides a comprehensive inventory of these misconceptions and demonstrates their persistence even after formal instruction. Our study, by contrast, examines how analogous misunderstandings appear in everyday computing tasks like emailing attachments or sharing Google Docs. It does so by drawing on this literature to identify possible problems and construct scenarios.

Prior research on file sharing and synchronization has largely used interviews, longitudinal studies, or in-person usability testing. Interview- and survey-based studies have investigated user practices, tool selection, and trust in file-sharing technologies, revealing a wide range of ad hoc behaviors—including emailing oneself or reverting to USB drives—and confusion around sharing and document version control [4, 24, 58, 59]. Long-term studies have surfaced file sharing scalability concerns and predictors of error [3]. Diaz and Harari employed observational assessments to evaluate task completion in Google Drive, where file sharing emerged

¹We specifically use *end-user* when we want to emphasize people who likely have limited understanding of technical details.

Woe	Discussion
Bob receives an email message that contains an attached Word document. He downloads it to his desktop. He would like to edit it. However, he is worried that if he does so, he will end up making changes to the document on the sender's machine.	It is worth noting that this is the exact reverse of the situation in section 1!
Alice starts using a password manager. She knows she should stop using her old, insecure passwords. Therefore, she allows the password manager to suggest new, secure passwords. However, changing the password in the password manager does not automatically change it on the Web site! In fact, Alice is now worse off because she has lost her old password, and has to go through a potentially complicated reset process. Even worse, until she tries to log in she does not realize the site has the old password—which is insecure, and hence much more vulnerable.	Password managers do not always prompt a user to go to a site to change the password there. Of course, there are numerous good critiques of the entire password infrastructure, though we believe passkeys introduce their own concerns.
Charlie travels from his home in New York to Hawaii. While in Hawaii, he creates a calendar entry for an event he will attend once he returns home. When he returns, he is confused because it appears to him that his phone is “still on Hawaiian time”. In fact, the phone has reset, but the event shows up at the wrong (Hawaiian) time because it was created in local time while he was in Hawaii.	Google Calendar offers the ability to display a “secondary time zone”, and can suggest this automatically. However, the interface is not always clear to users and, anyway, it still requires careful operation to check time zones for events.
Bob has the Google Drive synchronizer installed. He is running out of space on his machine, which is much smaller than the available space on Google Drive. To make room on his laptop, he deletes files that they have synchronized. He does not realize that because it's a <i>synchronizer</i> , this also deletes the files remotely, and after a while they are purged. Thus, the files he deleted because he thought they were securely stored elsewhere are the very files he permanently lost.	Google Drive's synchronizer, on some platforms and at some points in its history, will prompt a user with a warning when deleting a local file. But it is easy to miss the warning, or to suppress it in the future.
Alice sets up auto-payment on her credit card, where the bill is automatically paid from her bank account. At this point she becomes nervous about using her credit card online. Her threat model is that she has now linked her credit card account to her bank account, so a person who has access to her credit card may have access to her bank account as well. Along similar lines, once she starts paying for Zoom access she is nervous about leaving Zoom open, because “Zoom” now has access to her credit card.	Arguably, having performed the linkage, there <i>is</i> some tiny new threat. Having gained knowledge of one piece of identity, one could escalate privilege and learn more [27]. For this reason, modern sites make it easy to <i>change</i> information like the credit card number but not <i>read</i> (all) of it. Nevertheless, it is easy to see why Alice has trouble, for instance, distinguishing Zoom “the site” versus “the app”.

Figure 1. A partial catalog of woes.

as the single most difficult task [10]. In their study, Capra et al. noted user struggles with file synchronization, but also raised end-user concerns about privacy and work-personal separation of files [4]. While these works identify important usage challenges and concerns, our study focuses on relating these problems to programming semantics. Additionally, none of these works present a formal model of file sharing.

Numerous additional studies have explored the usability of document editing and collaboration tools, particularly focusing on user experiences across diverse populations [15, 18, 28, 29, 31, 35, 49, 54, 56, 58, 60], especially older adults

and individuals with cognitive impairments [18, 31, 35, 54]. These studies often report on what participants find helpful about document editing and sharing tools, as well as the specific usability concerns faced by these populations.

Other prior work has proposed user-visible affordances and prototypes for file sharing. Volda et al. [57] introduced a “sharing palette” UI that unified sharing dimensions and surfaced common confusions. Lindley et al. presented File Biography to visualize file histories [32]. Nebeling et al.'s MUBox [36] focused on simplified sharing controls. Siebers and Schmid's Dare2Del [51] system proposed a semantics

and interface to recommend irrelevant file deletions, but has nothing to do with file sharing. Finally, Dewan and Shen introduced an access control framework for collaborative applications with sophisticated ownership and access features with parallels to programming semantics (i.e. reachability) [9]. While these works provide valuable insights into user behavior or possible interface designs, they too do not frame the issues in terms of semantic misunderstandings.

Finally, some prior work models systems like web security protocols [1] and file sharing [20, 62]. However, these models are more system- than user-focused and have no study of usability or human factors. Additionally, the file sharing models focus on peer-to-peer file sharing systems rather than cloud sharing. One work [46] presents a usability-focused semantics of permissions for platforms like Google Drive, but it contains nothing about the end-user semantics of file sharing, the core focus of our contributions.

4 Formative User Study: Design

Our paper is motivated by the idea that the central analogy may explain some end-user difficulties. However, the studies discussed in section 3 are not centered on our premise; while they broadly demonstrate difficulties, we often have to read between the lines to find evidence for our claims. We therefore used the central analogy to design a study that translates programming misconceptions into CFS operations. If our premise is correct, then we should find difficulties in the cloud setting also.

For instance, the misconception literature (section 3) tells us that the following kind of program (in pseudocode) is often misinterpreted:

```
function Bob(receivedA):
  receivedA[0] := 42
end

aliceDoc = [5, 10, 15]
Bob(aliceDoc)
print(aliceDoc)
```

In most languages (what [33] calls the “Standard Model of Languages”), `receivedA` is an *alias* to `aliceDoc`, so the modification persists and is visible after Bob finishes. The common misunderstanding is that `receivedA` is a *copy*, so the change does not persist.

In CFS, Bob reflects Bob and the rest of the computation reflects Alice. So Alice first authors a document. Alice then shares a link to it with Bob. Bob modifies the document. Then Alice examines the document’s current state.

Having translated *problems*, how do we turn these into *tasks* for participants? Again exploiting the central analogy, we sought inspiration from the programming education literature. There is a long tradition of having students both *trace* (given this program, what is the output?) and *program* (given this specification, what is the program?).

To get a large and diverse set of participants, we chose to design for a crowdsourcing platform. This in turn imposes many limitations and creates specific threats to validity (section 9), not least that we cannot assume, for instance, specific programming abilities. In this section we describe the study design, in section 5 the logistics of deployment, and in section 6 our findings.

To begin, participants were given the following instructions:

In this survey, you’ll be asked to work through scenarios about individuals who are trying to share and edit documents. They could be using either Mac or Windows computers. Answer the questions based on whichever kind of computer you’re familiar with.

Each question includes a video explaining the steps taken before the question being asked. **Please watch the associated video before answering each question.**

The individuals are only using **Google Suite (Docs, Slides)** and **Microsoft Office (Word, PowerPoint)** for editing their documents and files. Additionally, they are only using **email** and **Google Docs** file sharing to send or share their files. They are **not** using any other programs that automatically save and sync files across different devices.

There are five scenarios in total. **No details carry over between scenarios.**

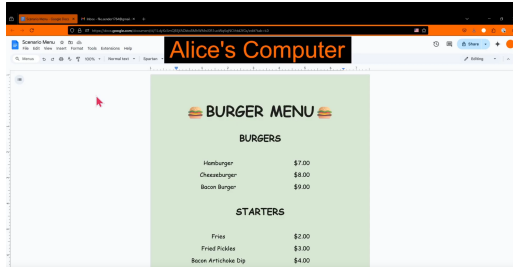
4.1 Tracing Tasks

In a tracing task, we show the participant a sequence of operations—in our case, through a video that screen-captures the use of Google Docs, Google Mail, and the Windows file explorer. The videos are accompanied by a voiceover explaining what they are showing. They also include transition slides alerting the viewer when the actor in the scenario changes. The videos range from 15 to 95 seconds in length. We gave each scenario a single-word internal codename related to the content of the file being shared.

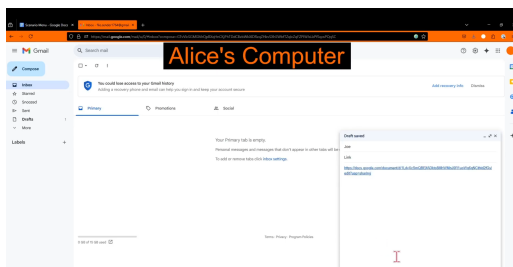
Scenario Menu. This scenario illustrates a *shared mutable reference*, where Alice and Joe collaborate on a single Google Docs file. Edits made by one party are immediately visible to the other, reflecting shared access to one file.

The video shows the following sequence:

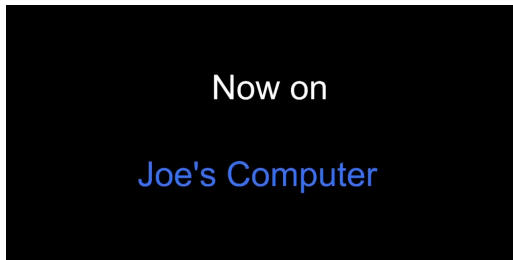
1. The narrator already has on screen a Google Docs file, created by Alice, containing a restaurant menu full of burgers. The heading says “Alice’s computer” in orange.



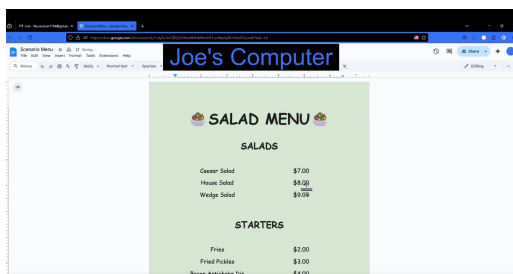
- The narrator then uses the share button to give edit access to Joe.
- Next, the narrator copies the link to the document, showing it can be done both from the URL-bar and from the Share menu.
- The narrator opens Google Mail and composes a message to Joe that shares the URL.



- A black transition screen appears with the text “Now on Joe’s computer”.



- The narrator then says “Now on Joe’s computer”, the heading says “Joe’s computer” in blue, and we see Joe’s Google Mail inbox with the email sent by Alice opened.
- Next, the narrator clicks on the link, which opens the document in Google Docs.
- Finally, the narrator then shows that Joe replaces the menu with one full of salads.



We *strongly* recommend that the reader stop here and view the scenario’s video (1m35s long):

<https://www.youtube.com/watch?v=l8x-1VR45KU>

After the video, the study asks the participant:

When **Alice** opens the document **after Joe’s edits**, which list will she see?

- ☐ Alice will see the burgers list 🍔.
- ☐ Alice will see the salads list 🥗.
- ☐ It depends (please explain).

Correct Answer. Alice will see the salads list.

Corresponding Program. For each scenario, we provide a program with the corresponding aliasing, copying, and mutation operations. We represent Joe as a function.

Joe edits the shared list.

```
function Joe(receivedDoc):
  receivedDoc[0] := "Caesar"
  receivedDoc[1] := "House"
  receivedDoc[2] := "Wedge"
end
```

Alice's steps:

```
menu = ['Ham', 'Cheese', 'Bacon']
Joe(menu)
print("Alice sees:", menu) # ["Caesar", ...]
```

This program mirrors Google Docs’s built-in sharing behavior: when Joe modifies the shared document, those changes are visible to Alice. This reflects aliasing, where both users operate on the same underlying object.

Additional Tasks. We have two additional tracing scenarios. Both of these ask two questions rather than one. The participants are shown a video before each of the questions. The video shown before the second question depends on their answer to the first. While the videos in the follow-up questions all depict the same action, the state of the document shown at the beginning of the second video is consistent with their answer to the first question, even if that answer was incorrect.

Scenario Logo. In the initial video,² Frank emails Kate a PowerPoint file containing a square-shaped logo. Kate then downloads the file and changes the logo from a square to a circle. Participants are then asked:

After **Kate** has made her change, if **Frank** opens the PowerPoint file that is **on his laptop**, which shape will he see?




- ☐ Frank will see the square logo ■.
- ☐ Frank will see the circle logo ●.
- ☐ It depends (please explain).

²Video: <https://youtu.be/cMr6ddHcbmE>

Correct Answer. Frank will see the square logo.

Depending on their answer, participants are shown one of two follow-up videos³ where Frank opens the original file he emailed to Kate and changes the logo to a hexagon. The logo begins as the answer they chose (square or circle). If the participant answers “It depends”, they are shown the square video, which is the correct one. In all cases, participants are then asked the following:


If **Kate** downloads the file from Frank’s original email **again**, which logo shape will Kate see in that **newly downloaded file**?

- ☐ Kate will see the square logo  version.
- ☐ Kate will see the circle logo  version.
- ☐ Kate will see the hexagon logo  version.
- ☐ It depends (please explain).

Correct Answer. Kate will see the square logo version.

This scenario illustrates the behavior of *isolated copies*. When Frank sends Kate a file via email, she downloads and edits her own local version. Changes to this copy do not propagate back to Frank’s original.

Corresponding Program.

```
coroutine Frank():
  localFile = { "shape": "
end



Frank()
```

This program reflects the fact that an email attachment is a copy. Kate’s downloading it creates a further copy. Her edits to localCopy remain isolated from Frank’s original localFile, which Frank later mutates independently. Kate’s second download creates yet another copy. We use coroutines for this program because each person resumes where they last left off.

³Follow-up videos: <https://youtu.be/aCmdeJ9tpPw> (if square), <https://youtu.be/84EUn3wunb0> (if circle).

Scenario Farm. In the first video,⁴ Marie shares a farm brochure document made in Google Docs with Aaron. The brochure features a cow. Aaron then opens the document in Google Docs and downloads it to his laptop as a Microsoft Word file. Finally, Aaron opens the Word file and changes the cow to a pig and saves the document.




If **Marie** opens the Google Doc **after Aaron edits the file on his laptop**, which animal will she see?

- ☐ Marie will see a cow .
- ☐ Marie will see a pig .
- ☐ It depends (please explain).

Correct Answer. Marie will see a cow.

Again, based on their answer, participants see a follow-up video⁵ in which Marie changes the animal to a chicken inside the Google Doc. The animal begins as the answer they chose (cow or pig). As in Scenario Logo, if the participant answers “It depends”, they are shown the (correct) cow video. In all cases, participants are then asked the following:



After Marie’s edits, Aaron opens the Word document **on his laptop**. Which animal will he see?


- ☐ Aaron will see a cow .
- ☐ Aaron will see a pig .
- ☐ Aaron will see a chicken .
- ☐ It depends (please explain).

Correct Answer. Aaron will see a pig.

This scenario shows the semantics of *local copies derived from shared references*. Downloading creates a second document, so subsequent changes are isolated and no longer synchronized. In the program below, this is reflected in downloading creating a copy.

Corresponding Program.

```
coroutine Marie():
  onlineDoc = { "animal": "
  print("Marie sees:", onlineDoc) # 
  Aaron()
end

coroutine Aaron(optional: receivedDoc):
  # Download local (Word) copy of Google Doc.
  localCopy = receivedDoc.copy()
  localCopy["animal"] := "
```

⁴Video: <https://youtu.be/e7jfwSJnws>

⁵Follow-up videos: <https://youtu.be/VLsl-pCRNqY> (if cow), https://youtu.be/blvuaQVB0_w (if pig).

```
Marie()
  print("Aaron sees:", localCopy) # 🤖
end

Marie()
```

4.2 Programming Tasks

Designing tracing tasks is relatively straightforward: we can show participants a video and ask them what should have happened by the end. Programming tasks are much harder, because they require expressing instructions unambiguously. How can we have participants do this? Using actual Google Docs, say, poses various problems. First, participants would need to create a second account, in ways that may affect the Terms of Use, cause their accounts to be blocked, etc. Second, we have to wait for email to be sent and received, which can take a while and can have errors in transit (e.g., spam filtering). Next, participants may inadvertently leak their identity in sharing information back with us. Finally, if their attention wanders, they may not complete the task.

Another option is to create a completely artificial environment in which they perform all their tasks. To make this correctly resemble the actual systems, however, we would have had to expend enormous programming effort, and small discrepancies might still have become notable confounds.

We decided to instead have them write down their intended instructions. While this does not let them check the output after each step, it does give us a sense of how well they understand cfs. It also matches a situation where, e.g., they have to tell someone else what to do over the phone without actually executing it themselves.

“Programming” Medium. We wanted our “language” and tool for capturing participants’ programs to yield a structured format that we could process with scripts. Drawing on programming-education literature, we chose *Parsons’ Problems* [38], a question form in which students select from and order candidate code snippets to complete a given task. Research has shown that Parsons’ Problems are as effective as “write this program from scratch” tasks for assessing novice students ability to construct programs [8]. In addition, students complete Parsons’ Problems in less time (again with similar learning gain) than with block-based programming [61]. In section 8.5, we will revisit the question of medium and how LLMs might fit into the picture.

In a Parsons setup, a student is given a blank target area and a set of cards with steps on them. Students must select and drag the relevant ones into the correct order. There may also be some *distractor* cards that are not part of the solution; students must be careful to not choose these. In this way, the student can demonstrate their semantic understanding independent of low-level syntactic issues.

Figure 2 shows our version, built in Qualtrics [45]. The steps are on the left; the region to assemble the program is

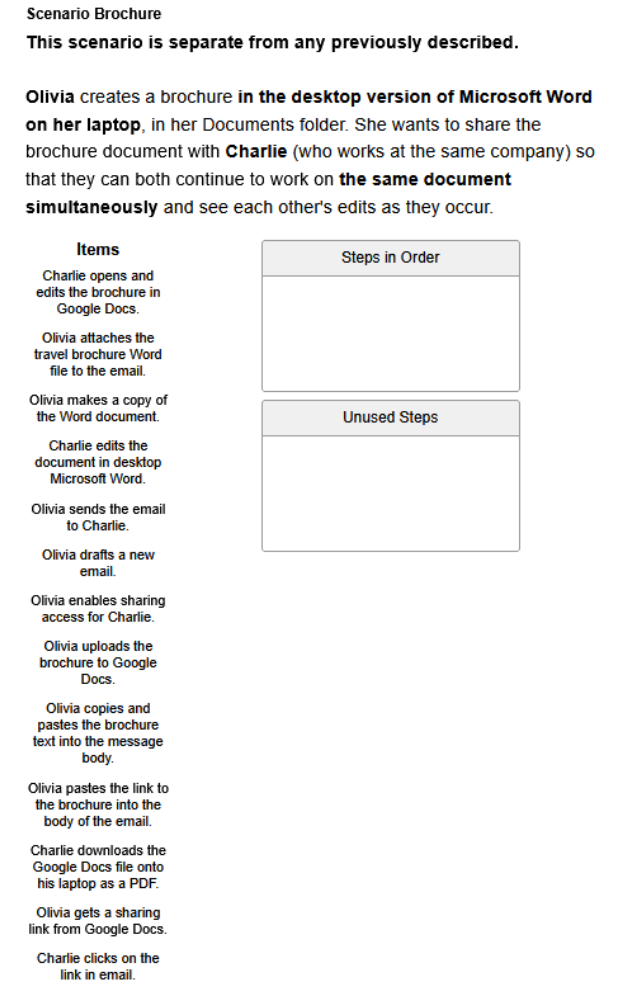


Figure 2. Parsons interface used in the programming tasks.

on the right. Qualtrics randomizes the order of the steps for each participant.

Training. Before beginning the main programming tasks, participants were introduced to the Parsons-style interface via a short practice task. This ensured they understood how to classify and reorder elements within the drag-and-drop environment, mitigating interface confusion as a source of error in subsequent tasks.

Participants were shown a (randomly ordered) list of six items on the left of the screen under the heading “Items”: “Apple”, “Bicycle”, “Onion”, “Straw”, “1764”, “1981”. To the right, two boxes were displayed: “Words in Alphabetical Order” and “Non-words”. They were instructed as follows:

On this page, you see a list of words and numbers, a box labeled “Words in Alphabetical Order” and a box labeled “Non-words”. Your task is to pick out all the words and put them in alphabetical (A–Z) order. You can drag

words into and out of the boxes, and reorder them by dragging within the box. Anything you don't want to use you can either leave in the item column or put in the box labeled "Non-words". **The survey will not let you proceed until you have the words in alphabetical order.**

This activity provided a preview of the interface's affordances: filtering relevant steps, reordering elements, and rejecting distractors. Participants had to successfully complete this practice task before continuing to the actual programming scenarios involving CFS workflows.

Post-Training. After completing this question, participants were given the following instructions:

You will now use the same interface to help the user. In the next two tasks, you will be given a list of possible instructions. The list you are given is in no particular order, and may contain superfluous or even incorrect instructions. Your task is to pick out the necessary instructions and put them in the correct order. You can drag instructions into and out of the boxes, and reorder them by dragging within the box.

Participants were then shown two scenarios in randomized order. We limited ourselves to no more than 13 total steps including 4 or 5 distractors. This was to avoid overwhelming the participants with too many options and to ensure the entire panel of instructions fit on-screen.⁶

Scenario Brochure. This has the following prompt:

Olivia creates a brochure **in the desktop version of Microsoft Word on her laptop**, in her Documents folder. She wants to share the brochure document with **Charlie** (who works at the same company) so that they can both continue to work on **the same document simultaneously** and see each other's edits as they occur.

The 8 correct steps in their intended order are as follows:

1. Olivia uploads the brochure to Google Docs.
2. Olivia enables sharing access for Charlie.
3. Olivia drafts a new email.
4. Olivia gets a sharing link from Google Docs.
5. Olivia pastes the link to the brochure into the body of the email.
6. Olivia sends the email to Charlie.⁷

⁶Both in traditional Parsons Problems and in our world, even though programs are written strictly sequentially, there can sometimes be a partial ordering. For instance, it does not matter whether Alice starts composing the email at the beginning or a few steps in; she just needs to have done so before she can paste a link into it. We therefore admit a family of answers, not just one.

7. Charlie clicks on the link in the email.
8. Charlie opens and edits the brochure in Google Docs.

The distractor steps are:

- Charlie downloads the Google Docs file onto his laptop as a PDF.
- Charlie edits the document in desktop Microsoft Word.
- Olivia attaches the brochure Word file to the email.
- Olivia copies and pastes the brochure text into the message body.
- Olivia makes a copy of the Word document.

This scenario captures the creation of a *shared reference derived from an originally local copy*. By uploading a local file to Google Docs and sharing it, Olivia makes a live, shared instance of it. The distractors reflected the belief that attaching a file, pasting its contents, or editing it locally in Word would create a shared reference, indicating confusion about aliasing and mutability and the role Google Docs plays in maintaining shared state.

Corresponding Program.

```
function Charlie(receivedDoc):
  # Edit the brochure in Google Docs.
  receivedDoc["data"] := "Edited"
end

# Olivia's steps:
# Create the brochure locally in Word.
localFile = { "data": "original" }
# Upload, creating a shared Google Docs.
uploadedDoc = localBrochure.copy()
# Email Charlie the Google Docs link.
Charlie(uploadedDoc)
print("Olivia sees:", uploadedDoc) # Edited
```

Here, Olivia's `uploadedDoc = localFile.copy()` step represents uploading her Word brochure to Google Docs and establishing a live, shared instance. In the Charlie function, he mutates that same shared object, so Olivia's subsequent `print("Olivia sees:", uploadedDoc)` reflects his edits. This models how Google Docs maintains a single, aliased document rather than independent copies.

Scenario Sensitive. This scenario is intended to reveal whether participants recognize the constraints posed by data sensitivity and the implications for tool selection. It begins with the prompt:

⁷Each "send the email to X" step implicitly includes the `setRecipients` operation in the corresponding programs.

David has a **Microsoft Word file on his laptop**. It contains sensitive company data. He needs to send it to **Erica** for her to review and edit. **David** also needs to receive **Erica's edits** back for review. Their company **prohibits uploading such files to third-party sites like Google Docs**.

The correct steps in their intended order are as follows:

1. David drafts a new email and attaches the document file to it.
2. David sends his email to Erica.
3. Erica downloads the file from David's email to her laptop.
4. Erica opens and edits the document on her laptop.
5. Erica drafts a new email and attaches the edited document to her email.
6. Erica sends her email to David.
7. David downloads the file from Erica's email to his laptop.
8. David opens and views the document on his laptop.

The distractor steps are:

- Erica uploads the Word file to Google Docs and edits it.
- David enables sharing access to the document for Erica.
- David downloads the file from Google Docs.
- David opens his original file in his Documents folder.

Semantically, this scenario emphasizes distributing *isolated copies* to prevent data leakage. The sensitive document is exchanged using email attachments to avoid persistent shared references. The four distractors probed for confusion about data locality and respecting the locality constraints.

Corresponding Program.

```
function Erica(receivedFile):
  # Download the file from David's email.
  localCopy = receivedFile.copy()
  # Edit the file.
  localCopy["data"] := "Edited"
  # Create attachment, which is a copy.
  outgoingAttachment = localCopy.copy()
  # Send the attachment to David.
  return outgoingAttachment
end

# David's steps:
localFile = { "data": "Original" }
# Create attachment, which is a copy.
attachment = localFile.copy()
```

```
# Send the attachment to Erica.
returnedAttachment = Erica(attachment)
# Download Erica's edited file from her email.
downloadedEdits = returnedAttachment.copy()
print("David sees:", downloadedEdits) # Edited
```

In this program, each `copy()` method represents creating an email attachment or downloading it. Every handoff uses a fresh copy, so Erica's edit doesn't affect David's original.

5 Formative User Study: Logistics

We deployed the Qualtrics study on Prolific [44], a crowdsourcing platform. We selected Prolific based on our own prior experience and on studies comparing crowdsourcing platforms [12, 40], which show that Prolific responses tend to be of higher quality than other platforms like Amazon Mechanical Turk. Additionally, Prolific offers participants stronger anonymization guarantees than Amazon.

The study was listed under the generic name “Technology Study.” Participants were shown the following description before opting in:

This study will ask a series of questions about specific technologies. Payouts will be issued within a week of completion.

The survey must not be taken on mobile devices (phones or tablets). Participants must be able to watch a video with included audio. No text captions will be provided.

We used Prolific's built-in filters to limit participation to self-identified fluent English speakers. All participants were paid above the US minimum wage at a rate of USD 8 per hour. The average completion time was approximately 30 minutes and 14 seconds, with individual times ranging from 10 minutes and 29 seconds to 1 hour and 17 minutes. (Because Prolific pays by duration, we suspect some participants artificially inflated their participation time.) A total of 46 participants completed the study.

The rest of this section provides details on our participant selection and demographics for those who are interested. Some readers may prefer to jump ahead to section 6 to learn our findings.

Details on Screening and Demographics

We started the study with two screener questions. The first screener asked them:

What's your experience level with using computers to share documents or presentations with others?

- ☐ I have never shared a document before
- ☐ I have shared documents once or twice
- ☐ I share documents occasionally
- ☐ I share documents regularly

Question	Correct	Visualized Accuracy	Semantic Concept	Action Description
Menu Q1	85%		Shared Mutable Reference	Shared via Google Docs link
Logo Q1	63%		Isolated Copy	File emailed and edited locally
Logo Q2	63%		Fresh Isolated Copy	File re-downloaded from email
Farm Q1	52%		Unsynced Local Edit	Downloaded from Docs and edited in Word
Farm Q2	41%		Diverged State	Word file reopened after online edits

Table 1. Participant performance on tracing questions ($N = 46$), annotated with semantic concept and action description.

Scenario	FC+CR+GC	MR	INC	WR	ICH	Visualized Accuracy	Semantic Concept
Brochure (share)	43%	20%	4.4%	24%	8.7%		Shared Mutable Reference
Sensitive (attach)	48%	20%	2.2%	28%	2.2%		Copy, Mutate, and Send

Table 2. Participant performance on programming questions ($N = 46$), annotated with semantic concept.

Participants who selected “never” were immediately excluded from the study and do not show up in our numbers. Of the remainder, about 70% chose “regularly”, 28% “occasionally”, and 2% (one person) “once or twice”.

Our second screener question asked:

What’s your experience level with Google Docs?

- ☐ I have never used Google Docs
- ☐ I have used Google Docs once or twice
- ☐ I use Google Docs occasionally
- ☐ I use Google Docs regularly

Again, we excluded people who selected “never”. Of the remaining respondents, 61% answered “regularly”, 30% “occasionally”, and 9% “once or twice”. We kept the “once or twice” group in our study, since there are many cloud file-sharing systems with similar characteristics.

Participants who passed screening performed the main study, as described above.

After completing the main study, participants were asked for some additional information. First, we asked them for their confidence with sharing:

How confident are you in your knowledge of how to effectively share documents with others?

- ☐ I’ve never done it before
- ☐ Not at all confident
- ☐ Somewhat confident
- ☐ Confident

44% were “somewhat confident” and 56% were “confident”. We also asked:

What’s your general comfort level with using computers to prepare documents or presentations?

- ☐ I’ve never done it before

- ☐ Not at all comfortable
- ☐ Somewhat comfortable
- ☐ Comfortable

Similarly, 44% were “somewhat comfortable” and 56% were “comfortable”.

These results confirm that participants had prior familiarity with document sharing and editing, making them a reasonable population for evaluating conceptual understanding of cloud-based file operations. We intentionally asked these questions *after* the scenarios; had seeing them reduced confidence, this would be reflected in the response.

Finally, we asked participants for their age range:

18-24	28%
25-34	39%
35-44	13%
45-54	13%
55-64	7%

Nobody marked being 65+ or declined to answer.

6 Formative User Study: Findings

6.1 Performance on Tracing Questions

Table 1 shows the proportion of participants answering each sub-question correctly. Participants performed best on Scenario Menu, where 84.8% of respondents selected the correct answer. This scenario was the most straightforward and involves only sticking to one tool (Google Docs), so it is by far the simplest. By contrast, questions that required reasoning about local vs. cloud-edited documents had notably lower performance, with the final Scenario Farm question answered correctly by just 41.3% of participants.

Two participants answered “It depends” on Scenario Menu. Both believed that Alice would only see Joe’s changes if he saved the document or if autosave was enabled. Since saving

would indeed affect whether changes persist, we marked both responses as correct.

In Scenario Farm, question 2, two participants also selected “It depends.” Both had previously answered question 1 incorrectly (choosing pig). One gave no explanation; the other (one of the participants who answered “It depends.” on Scenario Menu) again cited saving as a factor. However, because this question concerned changes made to the online document, saving would not affect the diverged local copy. As a result, we marked both of these responses as incorrect.

6.2 Performance on Programming Questions

Performance on the Parsons programming questions is more complicated than a simple binary outcome. We coded responses using the following categories:

Fully Correct (FC) The response achieved the stated goal without any unnecessary or incorrect steps.

Correct but Redundant (CR) The response achieved the stated goal, but included one or more semantically unnecessary steps.

Generously Correct (GC) The response achieved the stated goal, but contained minor errors, such as a small misordering or an omitted low-impact step.

Middle of the Road (MR) The response worked toward the stated goal but included one or more concerning steps that indicate partial misunderstandings or mixed conceptual models.

Incomplete (INC) The response made progress towards the stated goal, but was not complete (and hence did not attain the goal).

Incorrect (WR) The response contained one or more fundamentally incorrect or invalid operations that contradicted the scenario’s stated goal or constraints.

Incoherent (ICH) The response’s steps appeared to be chosen arbitrarily, with no meaningful alignment to the stated goal.

Table 2 uses these codes to report on performance. Even after grouping together the different kinds of correctness, we see that the overall success rate is lower than for tracing.

Scenario Brochure Analysis. In this scenario, Olivia must share a brochure with Charlie such that both can simultaneously edit it. We expected participants to provide a sharing workflow in Google Docs.

Responses were marked as **Incorrect (IO)** when:

- Only email attachments were used to send, and not Google Docs, violating the collaboration goal.
- The file was uploaded and sent as a Google Doc, but downloaded and edited locally.
- The file was both uploaded and sent as a link alongside an attachment, and both “Charlie opens and edits the brochure in Google Docs” and “Charlie edits the document in desktop Microsoft Word” steps were used, indicating a clear multi-modal confusion.

Responses were classified as **Middle of the Road (MR)** when they completed the task using Google Docs steps, but concluded with “Charlie edits the document in desktop Microsoft Word” either as the only edit step or in addition to “Charlie opens and edits the brochure in Google Docs”, indicating some confusion.

Scenario Sensitive Analysis. In this scenario, David must share a sensitive Word document with Erica without uploading to Google Docs.

Thus, there is a positive goal (sharing) and a negative goal (no data leakage). We expected participants to provide a workflow with an exchange of email attachments.

The majority of **Incorrect (IO)** responses failed because they included the step: “Erica uploads the Word file to Google Docs and edits it”. This directly violated a core constraint and thus rendered the solution invalid.

The only other incorrect response reversed the direction of file exchange—having Erica send the document to David first, who then returned it—which left the intent of the workflow ambiguous and open to multiple interpretations.

Middle of the Road (MR) responses were generally sound in their step selection and respected the data sensitivity constraint, but included one step that introduced potential confusion: “David enables sharing access to the document for Erica”. While this did not necessarily violate the leakage constraint (no actual upload occurred), it reflects a conceptual blend between email and cloud file-sharing, suggesting a partial or imprecise model of how sharing access works across different platforms.

The presence of these semantically “near-miss” responses in both scenarios underscores the importance of nuance when evaluating the output from the Parsons tasks. Overly rigid correctness criteria may obscure meaningful patterns in participant understanding.

These findings suggest a gap between participants’ perceived proficiency and their ability to correctly reason about the semantics of cloud and local document interactions. The results also reinforce the value of assessing both tracing and “programming” tasks when evaluating user understanding.

Comparing Tracing and Programming. The programs for Scenario Menu (tracing) and Scenario Brochure (programming) are almost identical, with the exception that the former creates a cloud document directly while the latter creates its cloud document by uploading a local file. Programming-education research has established that reading and writing programs are distinct yet correlated skills [19]; novices typically perform better on the former when both are tested together. Table 1 and table 2 also reflect this finding, showing correctness rates of 85% (Scenario Menu) vs 43% (Scenario brochure) respectively. While our design varies from traditional reading-vs-programming studies in not using the same notation for both kinds of tasks, the contrast lends further credence to our central analogy.

7 Taking Stock

At this point in the paper, it is useful to take stock of where we are and what we’ve learned so far:

- We began by describing real problems we have observed that people have with cloud-based applications (section 1 and section 2).
- We *argued* (section 1) that these problems are analogous to difficulties found in the literature with programming semantics.
- We saw (section 3) that other researchers have also found that users have difficulties with cloud applications. However, none of them have taken our theoretical stance, which is our central analogy. Their studies are therefore not designed to investigate this connection.
- Thus, we used the analogy to design a user study. We drew on the literature on programming misconceptions to create potentially confusing user interactions (section 4).
- We ran the user study (section 5) and saw (section 6, most concretely in table 1 and table 2) that participants do not do well on most tasks. Their problems are similar to those in the programming misconception literature (section 3), highlighting struggles with aliasing and copying that are revealed through inspecting state. Errors on Scenario Menu can reflect failure to detect aliasing when it occurs. Errors on Scenario Logo and Scenario Farm can reflect assumptions that aliasing has occurred when indeed it has not.

This finding lends credence to our theory: namely, users’ difficulties with CFS are analogous to student difficulties with stateful programming.⁸

It is also useful to clarify what we have **not** done. While we have shown that users make errors on our semantically-inspired scenarios, we have not shown that misconceptions about copies and aliases among documents are the root cause of those mistakes. Perhaps some other issue led our participants to perform poorly in our study, and users actually have a good conception of document copies. To be clear, our *hypothesis* remains that such misconceptions about document sharing exist and are a root cause; we just need more work to confirm or refute it.

8 Looking Forward: Semantics and Applications

To move forward, it would be helpful to (a) systematically design further studies to explore where users trip up while trying to achieve CFS goals, and (b) develop software tools

to help users achieve their goals. Both of these tasks would benefit from a formal *end-user semantics* of CFS.

It’s worth clarifying what we mean by “end-user semantics”. We do not mean “the semantics that the user believes is in effect”; that object is called a *mental model* in education and cognitive science. Each individual user could have their own (faulty!) mental model. Instead, we mean “the semantics of user-facing actions in this domain”, reflecting the behavior of tools like Google Docs and email clients, with primitives at the level of user actions: i.e., the EUISA.

This object is called a *notional machine* [13] in the computing education literature. For decades [52, 53], programming educators have used notional machines—which have been described as a “human-accessible operational semantics” [30]—to help students better understand programming models. Given a semantics of CFS (section 8.1), one could build a variety of supports that guide users around their misconceptions, which represent gaps between users’ mental models and the actual notional machines.

However, conventional tutoring does not make sense in this setting. Cognitively, learning something sufficiently for longer-term recall requires multiple iterations spaced out over time [14]. Investing time in tutoring for a one-off or infrequently performed task is behaviorally irrational, as Herley has argued (in the context of phishing training) [25]. We instead envision two potential support tools that we could build atop the semantics. The first would simulate the outcomes of specific actions (section 8.3); the second would synthesize valid action sequences to achieve desired sharing goals (section 8.4). Either of these uses could be built into tools or agents for sharing documents. We also discuss how large language models can help provide a user-friendly interface to these formal methods (section 8.5).

8.1 Formal Semantics

Our semantics is written in Forge [37], which is a variant of the popular Alloy [2] tool for modeling systems. The visualizations shown in this paper are generated using Cope-and-Drag [43], a tool for visualizing Alloy/Forge models. Forge can be used both for simulation and for limited forms of synthesis. For example, given a system model, Forge can generate a sequence of operations that lead to a goal state.

Our Forge model is quite large and detailed, with 11 sig (data) types, 19 operations, and more than 2000 lines of code. It not only depends on details of the Alloy and Forge languages, it also contains a good deal of bookkeeping (e.g., to maintain invariants). We do not expect the reader to find the details especially interesting. Accordingly, we provide an overview of the model and its relationship to conventional programming-language semantics here. Our full formal model is at <https://cs.brown.edu/research/pltdl/onward-2025/>. From the overview, a reader comfortable with formal modeling and familiar with the EUISA should be able to construct a semantics in their modeling system of choice.

⁸Of course, there may be a deeper problem—that people are just not very good at reasoning about sharing and mutation—and the cloud and programming settings may just be two concrete instantiations of that abstract difficulty. These problems would then also manifest in other domains. Investigating this deeper question is beyond the scope of this paper.

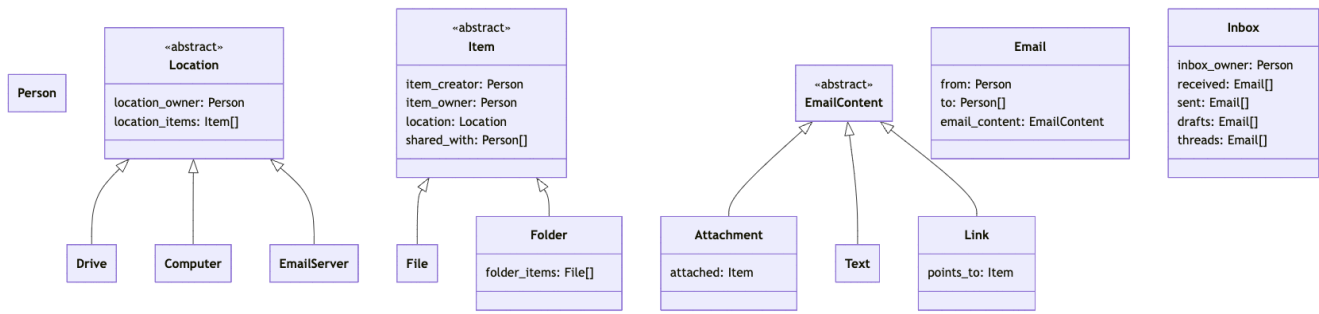


Figure 3. The datatypes in our semantic model. The datatypes towards the left capture files and folders, while the ones towards the right capture emails.

Atomic Entities. The basic types of a document-sharing system consist of documents (files) and folders (of files) that are stored in the cloud accounts or computers of individual people. As our model includes sharing by email, our datatypes also cover email messages, attachments, inboxes, and email servers. Figure 3 shows the high-level data types as a UML diagram. The full model also contains some datatype attributes that capture reverse relationships, such as a list of shared items in a Drive object that mirrors the shared_with relation within Item.

State Invariants. The state of a program consists of the atomic entities that exist within an instance of the model. To ensure fidelity to real-world sharing behaviors, the model enforces the following key consistency properties:

No dangling references All links and attachments must point to existing items in valid locations.

Ownership consistency Items and their containing locations must agree on owner.

Structural constraints Items may appear in at most one folder and one location; folders cannot be nested cyclically.

Primitive Operations. The primitive operations support creating and sharing documents, as well as creating and sending emails that share documents. Figure 4 lists the operations that create files, folders, and emails, as well as the operations that create copies, create aliases, and mutate potentially-shared data. The table elides some operations (such as removing email recipients) that our scenarios do not exercise; these appear in the full model.

Figure 4 shows that only files and folders can be aliased, shared, or mutated. Emails cannot; they are merely a user-facing mechanism for sharing documents and references to them.

To track when a copy has been made, our model maintains a relation between Items named same_content. Editing a

File creation and organization	
createFile	new object
createFolder	new object
duplicateFile	create copy
moveItem	-
File transfer between location types	
uploadFileToDrive	create copy
downloadDriveFile	create copy
Sharing	
shareItem	create alias
Email lifecycle	
createEmail	new object
setRecipients	-
addText	-
sendEmail	-
sendReply	-
Content embedding	
addLink	create alias
attachFile	create copy
attachFolder	create copy
Local retrieval	
downloadFileAttachment	create copy
downloadFolderAttachment	create copy
Editing	
editFile	mutation

Figure 4. Summary of main operations in the semantic model. The right column summarizes the impact on sharing, aliasing, and mutation. When the right column contains a hyphen, the operation adjusts other relations in the model, but does not impact these semantic features.

file breaks its same_content relationships. Our state invariants require same_content to be symmetric, transitive, and irreflexive.

The information in same_content is not typically maintained in a programming-language semantics. Programming

languages provide functions or methods to compute whether two objects are structurally equal. Our `same_content` is finer-grained than an `equals` function, however, because it only captures equality derived from copying. It does not capture pairs of objects that would be structurally equal by virtue of being created with the same attribute values. Capturing copies explicitly is helpful for generating traces that illustrate misconceptions (as we will see in section 8.4), or for explaining why a proposed program failed to achieve the desired results.

Each operation is encoded as a transition predicate over pre/post-states, where a state is the collection of entities (or objects) that currently exist. For example, the key constraints of the `sendEmail` operation are specified as follows:

```
pred sendEmail[actor: Person, email: Email] {
  email in inbox_owner.actor.drafts
  some email.to and some email.email_content
  inbox_owner.actor.sent' =
    inbox_owner.actor.sent + email
  all r: email.to |
    inbox_owner.r.received'
    = inbox_owner.r.received + email
}
```

This predicate states that `sendEmail` is an operation involving a `Person` (the sender) and an `Email` that has already been created. The predicate checks that the email has both a recipient (`email.to`) and content (`email_content`). In the post-state, the email has been added to the person's set of `sent'` emails and to the `received` sets for all recipients (the apostrophe denotes the relation in the post-state).

Programs. Programs in this semantics are sequences of actions, as will be explained in section 8.3. A visualization of one of these programs is shown in fig. 5. Full mappings of each scenario to the Forge semantics operations and their pseudocode programs appear in Appendix A.

8.2 Insights on Designing End-User Semantics

Our formal model highlights critical differences between end-user semantics and typical operational semantics for programming languages.

For starters, *end-user semantics can have multiple primitive operations that do basic linguistic tasks*. A typical programming language semantics may be much smaller than ours. That is because it is for a minimal *core* language. It is driven by goals such as keeping the number of cases small to facilitate (say) proofs by structural induction. But it does not represent the full reality of the surface language, whose many features are either left implicit or whose complexity is pushed somewhere else (such as desugaring operations [23]). In contrast, we faithfully represent all the end-user operations. We *could* have decomposed our semantics into a core into which we desugar these operations, but that means the output might be much harder to understand, since it would

all be in terms of core operations, not those the user recognizes. (Solutions like resugaring [41] may be of help here, but are still in a preliminary state.)

Secondly, *an end-user semantics should directly expose relationships that will have explanatory power for its users*. Here, we reiterate the point made earlier about `same_content`: if this relationship is visible in our semantics, we can use it to try to explain program behavior to end-users. In our user-study, many participants thought two references to files referred to the same object, missing the point at which copies were created and content diverged. Maintaining the `same_content` relation means we can use it to identify this point in time.

8.3 Program Simulation

In one modality, given a user's desired sequence of operations, we use the semantics to simulate the steps; the user adjusts the sequence until they obtain their desired behavior. We have already seen instances of this in section 4.2. Now we examine this through the lens of a semantics.⁹

Suppose Mary wants to share a file with John in such a way that he can view and edit it simultaneously. She might describe a sequence of steps where she:

1. Creates a file on her computer.
2. Uploads the file to Google Drive.
3. Opens Google Drive and configures sharing so that John can edit the file.
4. Copies the sharing link to the file.
5. Creates a new draft email.
6. Adds John as a recipient.
7. Pastes the link into the email body.
8. Sends the email.

which turns into the following operations over the semantics:

```
createFile[Mary, MaryComputer] # Create Item0
uploadFileToDrive[Mary, Item0] # Create Item1
shareItem[Mary, Item1, John]
createEmail[Mary] # Create Email0
setRecipients[Mary, Email0, John]
addLink[Mary, Item1, Email0]
sendEmail[Mary, Email0]
```

(Note: in the semantics, the fourth step about copying the sharing link occurs as part of the `addLink` operation.)

Running this program through the semantics shows the trace shown in fig. 5. Mary would examine this sequence of states and confirm that it has the effect she wants. Space limits preclude us including readable images in the print version of the paper (though zooming into the PDF will help). Instead, the images in the supplement show the result of each step in full detail. We do show the final state to give a better sense of what it looks like, but note that it would be the result of stepping through six previous steps:

⁹A natural next experiment would be to repeat those studies with these traces, to see whether errors reduce.

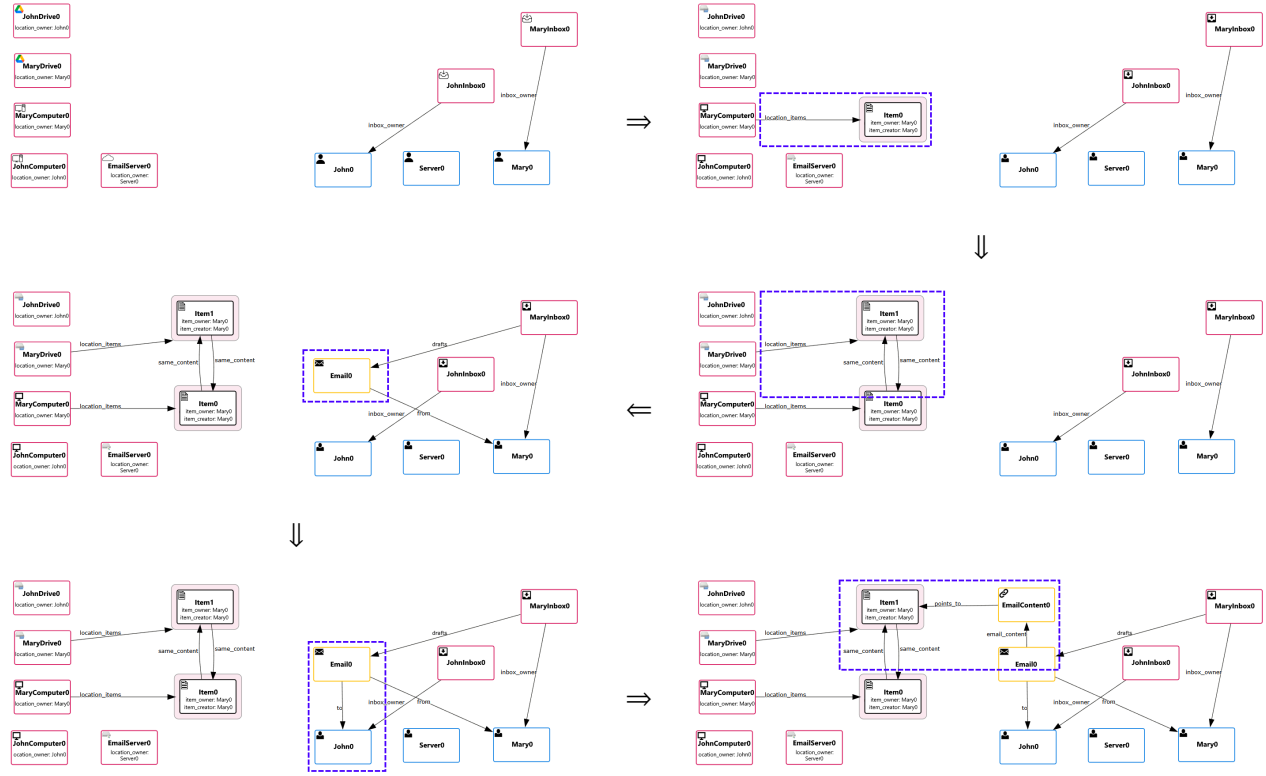
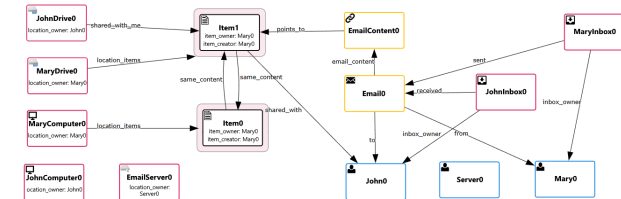


Figure 5. Trace steps (sans the last step) illustrating the progression of sharing actions in a boustrophedonic layout. The purple dashed-line boxes highlight what has changed.



8.4 Program Synthesis

More often, we expect users will have a desired goal—e.g., “I want to share this file with Joe in such a way that we can both edit it at the same time”—and want instructions that achieve it. Using the semantics in reverse amounts to a synthesis problem: producing a sequence of actions that results in this property. This sequence of actions can then be simulated (as above). This lets the user confirm that their intent has been accomplished. There are several reasons why it might not: they have made a mistake in their description; their description may have been ambiguous; their description might have been misinterpreted (e.g., by an LLM); or, seeing the actual simulation, they may realize it has undesired consequences.

Because the simulation is based on the precise semantics, these problems have a much better chance of being caught than without it. Once the user confirms the simulation, the sequence of instructions can be described to them in human-friendly form (and perhaps even partially executed on their behalf, e.g., by browser plugins or agents).

8.5 A Role for Language Models

To actually apply the semantics, we must help users translate their intent into a form that can exploit it. We can certainly consider block-based editors or a Parsons-style interface, as in section 4.2. However, both of these presume some familiarity with the available operations and their semantic implications. Those interfaces reduce surface-level complexity but still require users to explicitly reason about system behaviors and construct sequences of steps.

Large language models (LLMs) offer a compelling interface layer to bridge this usability gap. On the one hand, many LLMs are trained on common CFS workflows, allowing them to generate plausible, human-readable instructions. On the other hand, they are unreliable and can produce incorrect or subtly misleading suggestions, which are especially unsuitable when helping end-users or operating in security-sensitive contexts (e.g., when adding users to a Signal chat).

To mitigate this risk while retaining their accessibility, we propose a hybrid approach. LLMs work well as a front-end that accepts high-level natural instructions such as a sequence of instructions or a desired end-state. The LLM can then generate either an instruction sequence or a predicate—both in formal language—to be given to the simulator or synthesizer, respectively. This hides the surface language of the formal model from the user; they then interact with it through visualizations (as seen in fig. 5) to confirm that their intent has been properly understood. The translated or synthesized instructions can then be presented to the user and, in some cases, even executed directly (e.g., by a browser plugin). Thus, the LLM facilitates conversational interaction and clarification, but the formal model ensures that this remains grounded in precise and accurate semantics.

9 Threats to Validity

The main purpose of our user study is to build a bridge across the central analogy: that problems in programming could also be present in CFS. It also validates that the problems we have observed in person may be found in a much broader population. We are careful to consider it formative and preliminary. Nevertheless, for thoroughness, we list some of the threats to validity.

Internal Validity. Our goal was to assess conceptual understanding of cloud file-sharing behavior, but several factors could have interfered with that goal:

Branching logic effects For the two-part scenarios, the second video shown depended on the participant’s answer to the first question. This may have inadvertently reinforced incorrect mental models by confirming the participant’s earlier misconception.

Surface-level guessing Some participants may have answered correctly through deduction or pattern-matching without fully understanding the underlying concept, especially in the Parsons-style programming tasks.

Mistakes vs. misinterpretation Wrong answers could have resulted from task complexity rather than a true misunderstanding of the semantics.

Inconsistent engagement duration Some participants took significantly longer than the average of all participants. These extended durations may reflect interruptions or disengagement, which could reduce attention or continuity of reasoning between scenario questions, potentially distorting their understanding and therefore response accuracy.

Lack of validation We have not conducted any studies that ensure that our semantic model, its visualizations, and proposed tools (section 8), will actually aid users. We leave this for future work.

External Validity. There are numerous reasons why our work may not generalize:

Lack of ecological validity Participants were asked to simulate interactions with tools like Google Docs and Google Mail in their heads, rather than actually using the interfaces. In real environments, affordances, auto-suggestions, or UI constraints might help or hinder performance differently.

Sample representativeness While our sample includes participants across a range of ages, it may not fully represent the populations most susceptible to real-world CFS errors, such as older adults or less technically experienced users. However, given that our sample skews younger, we anticipate that including a broader age range might result in lower overall performance.

Device and platform variation We instructed participants to imagine using their preferred system (Mac or Windows), but did not control for actual experience or prior habits, which may influence reasoning.

Limited real-world stakes Participants were performing hypothetical tasks for compensation, not trying to actually share files or meet a goal. This may reduce their focus and effort invested in careful decision-making.

10 Discussion

The core contribution of our work is the central analogy, connecting CFS to programming operations. We not only see that there are clear parallels between aliasing, mutation, and copying, but that the problems end-users suffer from have parallels in difficulties that programmers also face. This suggests that tools analogous to those that help programmers may also help end-users. At the very least, this connection provides some clarity about what is happening, reducing the large number of applications and the variation in their interfaces to a small number of key semantic primitives—which is often a necessary precursor to a scientific treatment.

Interface Confusion. As we and the prior research show, users already suffer from semantic difficulties. We suspect these only get worse as more applications blur the bright line between the desktop and the cloud. For example, Microsoft Word’s desktop and online versions offer nearly identical interfaces, yet differ significantly in how they synchronize changes and track file state across devices. This surface-level consistency conceals deep semantic variation, making it difficult for users to build accurate mental models of what their actions actually do.

Helping Users. A valuable tool in STEM education is the concept inventory [26]. This is a multiple-choice instrument where every question has one correct answer and several wrong answers, where the wrong answers are each crisply tied to a *specific* misconception. This has the benefit that if a user chooses the wrong answer, they can be given feedback tied to that particular misconception, thus hopefully helping fix their underlying misunderstanding. While the average

end-user likely has neither the time nor inclination to use such an instrument, there are settings where it could be useful: e.g., in training staff within an organization, especially if they have to deal with sensitive documents.

Traces. The traces our models can generate, such as the one shown in section 8.3, are not necessarily ready to be consumed directly by an end-user. Here, we have focused on presenting the formal object. User-testing it and translating it into an end-user-friendly presentation remains an open problem for human-computer interaction research. However, we feel it is important that such work should be based on ground truth, which is what we have provided.

Growing the Model. Our semantic model can be extended in several directions. Our lived experience and study responses suggest features that would benefit both analysis and user-facing tools: support for fine-grained permission levels (view/comment/edit), representations of device power state (e.g., unsynced edits made offline), and the inclusion of alternative platforms such as Dropbox, which have distinct sync and versioning semantics. These extensions would allow the semantics to support broader categories of user confusion and better reflect real-world diversity in tooling.

Alternate Models. Our semantics (section 8) is based on first-order logic. It is possible that other models could also be effective. For instance, separation logic [47] helps users reason about sharing and non-sharing of values on the heap, which relates to our work using the central analogy. It is also possible that ownership [5] is relevant here, and validated work to explain ownership [6] may therefore also be useful.

Broader Populations. Our findings also raise concern for more vulnerable populations. Previous work has emphasized the value of systems that support situated reasoning for older adults and those experiencing cognitive decline (section 3). Our model and tools could provide a foundation for such systems: ones that can explain, simulate, or constrain user actions in ways aligned with their intent, rather than requiring them to guess what a platform will do.

The study conducted by McDonald and Mentis demonstrates that people experiencing memory loss can make informed decisions about their personal cybersecurity when systems allow them to reason about risks in context [35]. Participants were better able to choose appropriate safety settings—such as those for email, online banking, and password management—when they could visualize the implications of different actions through concrete scenarios. This ability to assign risk meaningfully to particular activities motivated the use of security features and shaped shared decision-making within caregiving partnerships. Our work parallels this emphasis on situated reasoning: by providing an interface to explore traceable, semantically explicit models of cloud file-sharing, we aim to support similar forms of understanding and planning.

More broadly, much of the human-computer interaction literature on dementia emphasizes the importance of technologies that support social connection, communication, and sharing [18, 31, 35]. These are also the domains in which misunderstandings about CFS can lead to confusion, data loss, or worse, that may be especially consequential for users with cognitive impairments. Our findings underscore the need to design systems that not only function securely but also surface their underlying behaviors in ways that are intelligible and support shared reasoning.

Looking Forward. This paper is the first step in a larger project on whether knowledge about misconceptions in programming can be applied to improve the design of end-user tools. This paper presents the architecture of studies towards this goal: check whether the end-user task reflects similar errors to a programming task (our user study); formalize a notional machine of how the end-user tool works (our model); use the notional machine to try to identify the specific misconceptions in the end-user domain (future work); develop tool supports after the notional machine has been validated (future work). Additional steps include exploring ways that current user-interfaces might contribute to creating semantic misconceptions and how situational and contextual factors might affect the general application of the semantics. Addressing these questions and expanding the work to the remaining scenarios in fig. 1 are fertile spaces for future work.

Acknowledgments

We thank our non-technical family and friends who test-drove the study and let us probe their understanding of cloud file-sharing. We thank Diana Freed, Tim Nelson, and Siddhartha Prasad for their help with the content of this paper. We appreciate the feedback from the reviewers, which significantly improved our presentation. This research was partly supported by US NSF grant DGE-2335625.

A Scenario-to-Semantics Mapping

Table 3. Tracing scenarios: Program pseudocode and Forge semantics steps.

Scenario & Narrative	Program Pseudocode	Forge Semantics Steps
Menu: Alice creates a burger menu in Google Docs and shares it live with Joe. Joe then edits in place (salads).	<pre> # Joe edits the shared list. function Joe(receivedDoc): receivedDoc[0] := "Salad X" receivedDoc[1] := "Salad Y" receivedDoc[2] := "Salad Z" end # Alice's steps: menu = ['Burger A', 'Burger B', 'Burger C'] Joe(menu) print("Alice sees:", menu) # ["Salad X", ...]</pre>	<pre> // Create Item0 atom in Alice's Drive. createFile[Alice, AliceDrive] // Share Item0 with Joe. shareItem[Alice, Item0, Joe] // Create Email0 atom. createEmail[Alice] // Set Joe as a recipient. setRecipients[Alice, Email0, Joe] // Add a link to Item0 to Email0. addLink[Alice, Item0, Email0] // Send Alice's Email0 to Joe. sendEmail[Alice, Email0] // Joe edits the shared document. editFile[Joe, Item0]</pre>
Logo: Frank emails Kate a PPT with a square logo. Kate changes her copy to a circle. Frank later edits his original to a hexagon. Finally, Kate re-downloads the original logo attachment.	<pre> coroutine Frank(): localFile = { "shape": "■" } # Create attachment, which is a copy. attachment = localFile.copy() Kate(attachment) print("Frank sees:", localFile) # ■ localFile["shape"] := "●" Kate() coroutine Kate(optional: receivedFile): # Download the attachment locally. localCopy = receivedFile.copy() localCopy["shape"] := "●" Frank() # Re-download the original attachment. redownload = receivedFile.copy() print("Kate sees:", redownload) # ■ end Frank()</pre>	<pre> // Create Item0 atom on Frank's computer. createFile[Frank, FrankComputer] // Create Email0 atom. createEmail[Frank] // Set Kate as a recipient. setRecipients[Frank, Email0, Kate] // Create Item1 atom attached to Email0. attachFile[Frank, Item0, Email0] // Send Frank's Email0 to Kate. sendEmail[Frank, Email0] // Create Item2 atom on Kate's computer. // (same_content as Item0 and Item1) downloadFileAttachment[Kate, Email0] // Kate edits her local copy (Item2). editFile[Kate, Item2] // Frank edits his original file (Item0). editFile[Frank, Item0]</pre>
Farm: Marie shares a Google Doc (cow) with Aaron. Aaron downloads it locally as Word and changes to pig. Marie then changes the shared Doc to chicken.	<pre> coroutine Marie(): onlineDoc = { "animal": "🐮" } # Share the live Google Doc with Aaron. Aaron(onlineDoc) # Edit the Google Doc. onlineDoc["animal"] := "🐷" print("Marie sees:", onlineDoc) # 🐷 Aaron() end coroutine Aaron(optional: receivedDoc): # Download local (Word) copy of Google Doc. localCopy = receivedDoc.copy() localCopy["animal"] := "🐷" Marie() print("Aaron sees:", localCopy) # 🐷 end Marie()</pre>	<pre> // Create Item0 atom on Marie's Drive. createFile[Marie, MarieDrive] // Share Item0 with Aaron. shareItem[Marie, Item0, Aaron] // Create Item1 atom on Aaron's computer. // (same_content as Item0) downloadDriveFile[Aaron, Item0] // Aaron edits his local copy (Item1). editFile[Aaron, Item1] // Marie edits the shared Doc file (Item0). editFile[Marie, Item0]</pre>

Table 4. Programming scenarios: Program pseudocode and Forge semantics steps.

Scenario & Narrative	Program Pseudocode	Forge Semantics Steps
Brochure: Olivia creates a Word brochure locally, uploads to Google Docs, shares live with Charlie, who then edits.	<pre> function Charlie(receivedDoc): # Edit the brochure in Google Docs. receivedDoc["data"] := "Edited" end # Olivia's steps: # Create the brochure locally in Word. localFile = { "data": "original" } # Upload, creating a shared Google Docs. uploadedDoc = localBrochure.copy() # Email Charlie the Google Docs link. Charlie(uploadedDoc) print("Olivia sees:", uploadedDoc) # Edited </pre>	<pre> // Create Item0 atom on Olivia's computer. createFile[Olivia, OliviaComputer] // Upload Item0 to Drive, creating Item1. // (same_content as Item0) uploadFileToDrive[Olivia, Item0] // Share Item1 with Charlie. shareItem[Olivia, Item1, Charlie] // Create Email0 atom. createEmail[Olivia] // Add link to Item1 to Email0. addLink[Olivia, Item1, Email0] // Send Olivia's Email0 to Charlie. setRecipients[Olivia, Email0, Charlie] sendEmail[Olivia, Email0] // Charlie edits the shared document. editFile[Charlie, Item1] </pre>
Sensitive: David and Erica exchange only isolated email attachments of a sensitive file, round-trip.	<pre> function Erica(receivedFile): # Download the file from David's email. localCopy = receivedFile.copy() # Edit the file. localCopy["data"] := "Edited" # Create attachment, which is a copy. outgoingAttachment = localCopy.copy() # Send the attachment to David. return outgoingAttachment end # David's steps: localFile = { "data": "Original" } # Create attachment, which is a copy. attachment = localFile.copy() # Send the attachment to Erica. returnedAttachment = Erica(attachment) # Download Erica's edited file from her email. downloadedEdits = returnedAttachment.copy() print("David sees:", downloadedEdits) # Edited </pre>	<pre> // Create Item0 atom on David's computer. createFile[David, DavidComputer] // Create Email0 atom. createEmail[David] // Create Item1 atom attached to Email0. attachFile[David, Item0, Email0] // Send David's Email to Erica. setRecipients[David, Email0, Erica] sendEmail[David, Email0] // Create Item2 atom on Erica's computer. // (same_content as Item0 and Item1) downloadFileAttachment[Erica, Email0] // Erica edits her local copy (Item2). editFile[Erica, Item2] // Creates Email1 atom. createEmail[Erica] // Create Item3 atom attached to Email1. attachFile[Erica, Item2, Email1] // Send Erica's Email to David. setRecipients[Erica, Email1, David] sendEmail[Erica, Email1] // Create Item4 atom on David's computer. // (same_content as Item2 and Item3) downloadFileAttachment[David, Email1] </pre>

References

- [1] Devdatta Akhawe, Adam Barth, Peifung E. Lam, John Mitchell, and Dawn Song. 2010. Towards a Formal Foundation of Web Security. In *2010 23rd IEEE Computer Security Foundations Symposium*. IEEE Computer Society, 290–304. doi:10.1109/CSF.2010.27
- [2] Alloy. 2023. Alloy Analyzer Website. <https://alloytools.org>. Accessed May 4, 2023.
- [3] Ofer Bergman, Tamar Israeli, and Steve Whittaker. 2020. The scalability of different file-sharing methods. *J. Assoc. Inf. Sci. Technol.* 71, 12 (Nov. 2020), 1424–1438. doi:10.1002/asi.24350
- [4] Robert Capra, Emily Vardell, and Kathy Brennan. 2014. File synchronization and sharing: User practices and challenges. *Proceedings of the American Society for Information Science and Technology* 51, 1 (2014), 1–10. doi:10.1002/meet.2014.14505101059
- [5] David G. Clarke, John M. Potter, and James Noble. 1998. Ownership Types for Flexible Alias Protection. In *Proceedings of the 13th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA '98)*. Association for Computing Machinery, New York, NY, USA, 48–64. doi:10.1145/286936.286947
- [6] Will Crichton, Gavin Gray, and Shriram Krishnamurthi. 2023. A Grounded Conceptual Model for Ownership Types in Rust. *Proc. ACM Program. Lang.* 7, OOPSLA2, Article 265 (Oct. 2023), 29 pages. doi:10.1145/3622841
- [7] Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: Simplified data processing on large clusters. In *Operating System Design and Implementation*.
- [8] Paul Denny, Andrew Luxton-Reilly, and Beth Simon. 2008. Evaluating a new exam question: Parsons problems. In *Proceedings of the Fourth International Workshop on Computing Education Research (Sydney, Australia) (ICER '08)*. Association for Computing Machinery, New York, NY, USA, 113–124. doi:10.1145/1404520.1404532
- [9] Prasun Dewan and HongHai Shen. 1998. Flexible meta access-control for collaborative applications. In *Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work (Seattle, Washington, USA) (CSCW '98)*. Association for Computing Machinery, New York, NY, USA, 247–256. doi:10.1145/289444.289499
- [10] Javier Diaz and Ivana Harari. 2016. Are Google Office Applications Easy for Seniors?: Usability Studies with 120 Elderly Users. In *HCI International 2016 – Posters' Extended Abstracts*, Constantine Stephanidis (Ed.). Springer International Publishing, 420–425.
- [11] Amer Diwan, William M. Waite, and Michele H. Jackson. 2004. PL-detector: a system for teaching programming language concepts. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education (Norfolk, Virginia, USA) (SIGCSE '04)*. Association for Computing Machinery, New York, NY, USA, 80–84. doi:10.1145/971300.971330
- [12] B. D. Douglas, P. J. Ewell, and M. Brauer. 2023. Data quality in online human-subjects research: Comparisons between MTurk, Prolific, CloudResearch, Qualtrics, and SONA. *PLOS ONE* 18, 3 (2023), e0279720. doi:10.1371/journal.pone.0279720
- [13] Benedict Du Boulay, Tim O'Shea, and John Monk. 1981. The Black Box inside the Glass Box: Presenting Computing Concepts to Novices. *International Journal of man-machine studies* 14, 3 (1981), 237–249. <https://www.sciencedirect.com/science/article/pii/S0020737381800569>
- [14] Hermann Ebbinghaus. 1964. *Memory: A Contribution to Experimental Psychology*. Dover, New York. Translated by H. A. Ruger and C. E. Bussenius. Original work published 1885.
- [15] Clarence A. Ellis, Simon J. Gibbs, and Gail Rein. 1991. Groupware: some issues and experiences. *Commun. ACM* 34, 1 (Jan. 1991), 39–58. doi:10.1145/99977.99987
- [16] Kathi Fisler, Shriram Krishnamurthi, and Preston Tunnell Wilson. 2017. Assessing and Teaching Scope, Mutation, and Aliasing in Upper-Level Undergraduates. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (Seattle, Washington, USA) (SIGCSE '17)*. Association for Computing Machinery, New York, NY, USA, 213–218. doi:10.1145/3017680.3017777
- [17] Ann E. Fleury. 1991. Parameter passing: the rules the students construct. In *Proceedings of the Twenty-Second SIGCSE Technical Symposium on Computer Science Education (San Antonio, Texas, USA) (SIGCSE '91)*. Association for Computing Machinery, New York, NY, USA, 283–286. doi:10.1145/107004.107066
- [18] Sarah Foley, Daniel Welsh, Nadia Pantidi, Kellie Morrissey, Tom Nappey, and John McCarthy. 2019. Printer Pals: Experience-Centered Design to Support Agency for People with Dementia. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (Glasgow, Scotland UK) (CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–13. doi:10.1145/3290605.3300634
- [19] Max Fowler, David H. Smith IV, Mohammed Hassan, Seth Poulsen, Matthew West, and Craig Zilles. 2022. Reevaluating the relationship between explaining, tracing, and writing skills in CS1 in a replication study. *Computer Science Education* 32, 3 (2022), 355–383. doi:10.1080/08993408.2022.2079866
- [20] Z. Ge, D.R. Figueiredo, Sharad Jaiswal, J. Kurose, and D. Towsley. 2003. Modeling peer-peer file sharing systems. In *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No.03CH37428)*, Vol. 3. 2188–2198 vol.3. doi:10.1109/INFCOM.2003.1209239
- [21] Martin Georgiev, Subodh Iyengar, Suman Jana, Rishita Anubhai, Dan Boneh, and Vitaly Shmatikov. 2012. The most dangerous code in the world: validating SSL certificates in non-browser software. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security (Raleigh, North Carolina, USA) (CCS '12)*. Association for Computing Machinery, New York, NY, USA, 38–49. doi:10.1145/2382196.2382204
- [22] Ken Goldman, Paul Gross, Cinda Heeren, Geoffrey Herman, Lisa Kaczmarczyk, Michael C. Loui, and Craig Zilles. 2008. Identifying important and difficult concepts in introductory computing courses using a delphi process. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education (Portland, OR, USA) (SIGCSE '08)*. Association for Computing Machinery, New York, NY, USA, 256–260. doi:10.1145/1352135.1352226
- [23] Arjun Guha, Claudiu Saftoiu, and Shriram Krishnamurthi. 2010. The Essence of JavaScript. In *European Conference on Object-Oriented Programming*.
- [24] Sarah Henderson. 2011. Document duplication: How users (struggle to) manage file copies and versions. *Proceedings of the American Society for Information Science and Technology* 48, 1 (2011), 1–10. doi:10.1002/meet.2011.14504801013
- [25] Cormac Herley. 2009. So long, and no thanks for the externalities: the rational rejection of security advice by users. In *Proceedings of the 2009 Workshop on New Security Paradigms Workshop (Oxford, United Kingdom) (NSPW '09)*. Association for Computing Machinery, New York, NY, USA, 133–144. doi:10.1145/1719030.1719050
- [26] David Hestenes, Malcolm Wells, and Gregg Swackhamer. 1992. Force Concept Inventory. *The Physics Teacher* 30 (March 1992), 141–158.
- [27] Mat Honan. 2012. How Apple and Amazon Security Flaws Led to My Epic Hacking. *Wired* (6 August 2012). <https://www.wired.com/2012/08/apple-amazon-mat-honan-hacking/> Accessed: 2025-04-20.
- [28] Fawzi Fayeze Ishtaiwa and Ibtehal Mahmoud Aburezeq. 2015. The impact of Google Docs on student collaboration: A UAE case study. *Learning, Culture and Social Interaction* 7 (2015), 85–96. doi:10.1016/j.lcsi.2015.07.004
- [29] Young-Wook Jung, Youn-kyung Lim, and Myung-suk Kim. 2017. Possibilities and Limitations of Online Document Tools for Design Collaboration: The Case of Google Docs. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing (Portland, Oregon, USA) (CSCW '17)*. Association for Computing Machinery, New York, NY, USA, 1096–1108. doi:10.1145/2998181.2998297

- [30] Shriram Krishnamurthi and Kathi Fisler. 2019. Programming Paradigms and Beyond. In *The Cambridge Handbook of Computing Education Research*, Sally Fincher and Anthony Robins (Eds.). Cambridge University Press.
- [31] Amanda Lazar, Caroline Edasis, and Anne Marie Piper. 2017. A Critical Lens on Dementia and Design in HCI. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (Denver, Colorado, USA) (CHI '17). Association for Computing Machinery, New York, NY, USA, 2175–2188. doi:10.1145/3025453.3025522
- [32] 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* (Montreal QC, Canada) (CHI '18). Association for Computing Machinery, New York, NY, USA, 1–12. doi:10.1145/3173574.3173692
- [33] Kuang-Chen Lu and Shriram Krishnamurthi. 2024. Identifying and Correcting Programming Language Behavior Misconceptions. *Proc. ACM Program. Lang.* 8, OOPSLA1, Article 106 (April 2024), 28 pages. doi:10.1145/3649823
- [34] Kuang-Chen Lu, Shriram Krishnamurthi, Kathi Fisler, and Ethel Tshukudu. 2023. What Happens When Students Switch (Functional) Languages (Experience Report). *Proc. ACM Program. Lang.* 7, ICFP, Article 215 (Aug. 2023), 17 pages. doi:10.1145/3607857
- [35] Nora McDonald and Helena M. Mentis. 2021. Building for 'We': Safety Settings for Couples with Memory Concerns. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) (CHI '21). Association for Computing Machinery, New York, NY, USA, Article 554, 11 pages. doi:10.1145/3411764.3445071
- [36] Michael Nebeling, Matthias Geel, Oleksiy Syrotkin, and Moira C. Norrie. 2015. MUBox: Multi-User Aware Personal Cloud Storage. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (Seoul, Republic of Korea) (CHI '15). Association for Computing Machinery, New York, NY, USA, 1855–1864. doi:10.1145/2702123.2702233
- [37] Tim Nelson, Ben Greenman, Siddhartha Prasad, Tristan Dyer, Ethan Bove, Qianfan Chen, Charles Cutting, Thomas Del Vecchio, Sidney LeVine, Julianne Rudner, Ben Ryjikov, Alexander Varga, Andrew Wagner, Luke West, and Shriram Krishnamurthi. 2024. Forge: A Tool and Language for Teaching Formal Methods. *Proceedings of the ACM on Programming Languages* 8, OOPSLA1 (2024), 613–641.
- [38] Dale Parsons and Patricia Haden. 2006. Parson's programming puzzles: a fun and effective learning tool for first programming courses. In *Proceedings of the 8th Australasian Conference on Computing Education - Volume 52* (Hobart, Australia) (ACE '06). Australian Computer Society, Inc., AUS, 157–163.
- [39] David A. Patterson. 2008. Technical perspective: the data center is the computer. *Commun. ACM* 51, 1 (2008), 105–105.
- [40] Eyal Peer, David Rothschild, Andrew Gordon, Lior Everdy, Jörn Grah, and Jesse Chandler. 2022. Data quality of platforms and panels for online behavioral research. *Behavior Research Methods* 54 (2022), 1643–1662. doi:10.3758/s13428-021-01694-3
- [41] Justin Pombrio and Shriram Krishnamurthi. 2014. Resugaring: Lifting Evaluation Sequences through Syntactic Sugar. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*.
- [42] Justin Pombrio, Shriram Krishnamurthi, and Kathi Fisler. 2017. Teaching Programming Languages by Experimental and Adversarial Thinking. In *2nd Summit on Advances in Programming Languages (SNAPL 2017) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 71)*, Benjamin S. Lerner, Rastislav Bodík, and Shriram Krishnamurthi (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 13:1–13:9. doi:10.4230/LIPIcs.SNAPL.2017.13
- [43] Siddhartha Prasad, Ben Greenman, Tim Nelson, and Shriram Krishnamurthi. 2025. Lightweight Diagramming for Lightweight Formal Methods: A Grounded Language Design. doi:10.4230/LIPIcs.ECOOP.2025.26
- [44] Prolific. 2025. Prolific. <https://www.prolific.com>. London, UK. Accessed April 2025..
- [45] Qualtrics. 2025. Qualtrics. <https://www.qualtrics.com>. Provo, Utah, USA. Accessed April 2025..
- [46] Hannah Quay-de la Vallee, James M. Walsh, William Zimrin, Kathi Fisler, and Shriram Krishnamurthi. 2013. Usable security as a static-analysis problem: modeling and reasoning about user permissions in social-sharing systems. In *Proceedings of the 2013 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software* (Indianapolis, Indiana, USA) (Onward! 2013). Association for Computing Machinery, New York, NY, USA, 1–16. doi:10.1145/2509578.2509589
- [47] J.C. Reynolds. 2002. Separation logic: a logic for shared mutable data structures. In *Proceedings 17th Annual IEEE Symposium on Logic in Computer Science*. 55–74. doi:10.1109/LICS.2002.1029817
- [48] Sam Saarinen, Shriram Krishnamurthi, Kathi Fisler, and Preston Tunnell Wilson. 2019. Harnessing the Wisdom of the Classes: Class-sourcing and Machine Learning for Assessment Instrument Generation. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) (SIGCSE '19). Association for Computing Machinery, New York, NY, USA, 606–612. doi:10.1145/3287324.3287504
- [49] Alaa Sadik. 2017. Students' acceptance of file sharing systems as a tool for sharing course materials: The case of Google Drive. *Education and Information Technologies* 22 (2017), 2455–2470. <https://api.semanticscholar.org/CorpusID:35743602>
- [50] Jorma Sajaniemi, Marja Kuittinen, and Taina Tikansalo. 2008. A study of the development of students' visualizations of program state during an elementary object-oriented programming course. *J. Educ. Resour. Comput.* 7, 4, Article 3 (Jan. 2008), 31 pages. doi:10.1145/1316450.1316453
- [51] Michael Siebers and Ute Schmid. 2019. Please delete that! Why should I? *KI - Künstl. Intell.* 33, 1 (March 2019), 35–44.
- [52] Juha Sorva. 2013. Notional machines and introductory programming education. *ACM Trans. Comput. Educ.* 13, 2, Article 8 (July 2013), 31 pages. doi:10.1145/2483710.2483713
- [53] Juha Sorva, Ville Karavirta, and Lauri Malmi. 2013. A Review of Generic Program Visualization Systems for Introductory Programming Education. *ACM Trans. Comput. Educ.* 13, 4, Article 15 (Nov. 2013), 64 pages. doi:10.1145/2490822
- [54] Kelly S. Steelman, Kay L. Tislar, Leo C. Ureel, and Charles Wallace. 2016. Breaking Digital Barriers: A Social-Cognitive Approach to Improving Digital Literacy in Older Adults. In *HCI International 2016 – Posters' Extended Abstracts*, Constantine Stephanidis (Ed.). Springer International Publishing, Cham, 445–450.
- [55] Filip Strömback, Pontus Haglund, Aseel Berglund, and Erik Berglund. 2023. The Progression of Students' Ability to Work With Scope, Parameter Passing and Aliasing. In *Proceedings of the 25th Australasian Computing Education Conference* (Melbourne, VIC, Australia) (ACE '23). Association for Computing Machinery, New York, NY, USA, 39–48. doi:10.1145/3576123.3576128
- [56] Xin Tan and Yongbeom Kim. 2015. User acceptance of SaaS-based collaboration tools: a case of Google Docs. *J. Enterp. Inf. Manag.* 28 (2015), 423–442. <https://api.semanticscholar.org/CorpusID:29712512>
- [57] Stephen Volda, W. Keith Edwards, Mark W. Newman, Rebecca E. Grinter, and Nicolas Ducheneaut. 2006. Share and share alike: exploring the user interface affordances of file sharing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Montréal, Québec, Canada) (CHI '06). Association for Computing Machinery, New York, NY, USA, 221–230. doi:10.1145/1124772.1124806
- [58] Tara Whalen, Diana Smetters, and Elizabeth F. Churchill. 2006. User experiences with sharing and access control. In *CHI '06 Extended Abstracts on Human Factors in Computing Systems* (Montréal, Québec, Canada) (CHI EA '06). Association for Computing Machinery, New

- York, NY, USA, 1517–1522. doi:[10.1145/1125451.1125729](https://doi.org/10.1145/1125451.1125729)
- [59] Tara Whalen, Elaine Toms, and James Blustein. 2008. File Sharing and Group Information Management. *Personal Information Management: PIM* (2008).
- [60] Soobin Yim, Mark Warschauer, and Binbin Zheng. 2016. Google Docs in the Classroom: A District-wide Case Study. *Teachers College Record* 118, 9 (2016), 1–32. doi:[10.1177/016146811611800903](https://doi.org/10.1177/016146811611800903) arXiv:<https://doi.org/10.1177/016146811611800903>
- [61] Rui Zhi, Min Chi, Tiffany Barnes, and Thomas W. Price. 2019. Evaluating the Effectiveness of Parsons Problems for Block-based Programming. In *Proceedings of the 2019 ACM Conference on International Computing Education Research* (Toronto ON, Canada) (ICER '19). Association for Computing Machinery, New York, NY, USA, 51–59. doi:[10.1145/3291279.3339419](https://doi.org/10.1145/3291279.3339419)
- [62] Li Zhou and M.H. Ammar. 2003. A file-centric model for peer-to-peer file sharing systems. In *11th IEEE International Conference on Network Protocols, 2003. Proceedings.* 28–37. doi:[10.1109/ICNP.2003.1249754](https://doi.org/10.1109/ICNP.2003.1249754)

Received 2025-04-25; accepted 2025-08-11