FxD: a functional debugger for dysfunctional spreadsheets

Ian Drosos[1], Nicholas Wilson[1], Andrew D. Gordon[12], Sruti Srinivasa Ragavan[3], Jack Williams[1]
[1] *Microsoft Research, Cambridge, UK*; [2] *University of Edinburgh, UK*; [3] *Indian Institute of Technology, Kanpur, India*
[1] {*t-iandrosos, nicholas.wilson, adg, jack.williams*}*@microsoft.com;* [3] *srutis@cse.iitk.ac.in*

***Abstract*—Recent enhancements to the spreadsheet formula language and intelligent spreadsheet interfaces allow spreadsheet users to build more complex spreadsheets in systematic ways (e.g., via functional abstractions). However, users have been slow to adopt such features, partly due to the absence of corresponding improvements in tools such as editors and debuggers.**

In this paper, we present FxD, a novel spreadsheet debugging interface, which provides structured information needed for spreadsheet users to debug formulas in systematic ways through affordances such as the ability to step into the execution of dependencies and provide contextual information to users based on the current context.

An in-vitro, within-subject (n=12) experiment revealed that, even though using FxD did not lead to faster debugging, participants reported qualitative improvements (e.g., feelings of efficiency and capability) when debugging with it. Further, participants were more satisfied with the amount of information provided by FxD and felt that it would enhance their existing debugging workflows. Our results have implications for the design of debuggers for spreadsheets and for functional programming languages in general.

***Index Terms*—spreadsheets, debugging, end-user programming, functional programming**

I. Introduction

Spreadsheets are one of the most widely used programming languages [1], [2]. Spreadsheet users are end-user programmers who program using the spreadsheet formula language, a functional programming language.

As with any programming activity, programming in spreadsheets also involves activities such as debugging, testing, maintenance and design [3]. This reasoning lies at the root of a large body of work in end-user programming and end-user software engineering; examples include fault isolation [4], spreadsheet refactoring [5] and spreadsheet comprehension [6]. Several language extensions have also been introduced over the years to help users manage complexity via abstractions. For example, arrays [7] and custom data types [8] provide new data structures to store and access information, and sheet-defined functions [9] and LAMBDA [10] allow functional abstractions.

Unfortunately, these language improvements have not been accompanied by improvement in spreadsheet development tools in commercial spreadsheets. If any, such tool improvements are predominantly focused on authoring spreadsheets (e.g., suggesting formulas) and overlook other equally important activities such as debugging or testing. Indeed, a recent study of spreadsheet users found users pointing out the need for better spreadsheet development tools, including better syntax highlighting in editors, better debuggers, and even package management systems [2]. Multiple studies in the past have also indicated these limitations, particularly the challenges of debugging spreadsheets [11]–[13].

Building on these past findings, we set out to build a debugger for spreadsheet formulas. We call it FxD. The idea of debugging spreadsheets is by no means new, and nor is ours the first spreadsheet debugger; all commercial spreadsheet packages come with some debugger capabilities. Still, users face various challenges debugging, and the existing tools have remained largely unchanged in decades and user research on spreadsheet debugging is sparse. Therefore, we asked the question: *what would a spreadsheet debugger look like, if we took a principled, user-centric approach to design one?*

To answer the above question, we reviewed the literature on spreadsheet debugging, as well as debuggers in the functional programming paradigm (Section II). Based on this review:

- We implement FxD, which embodies a new interaction style for debugging spreadsheet formulas: it provides live debugging and a feature-rich set of navigation tools for users to inspect formulas by exploring the steps in formula execution, discerning connections between steps through contextual coloring, and by inspecting data and formula context (Section III).
- We evaluate FxD through a user study: we found that FxD provides the right amount of information for users to debug spreadsheet issues, made participants feel more efficient and capable, integrates with the spreadsheet user's debugging workflow, reduced the need for users to turn to web searches when comprehending unfamiliar and complex formulas, and even assisted them in authoring new formulas. We collected feedback on FxD features to improve them for future debuggers (Sections V and VI).

II. Related work

Debugging research is too vast to describe in this section; [14] offers a detailed review of the literature. Here, we discuss only those works relevant to spreadsheet debugging.

A. Spreadsheet errors and their management

Spreadsheets are widely used [15] but continue to be notoriously error-prone [16], [17]; this has motivated a large body of research on spreadsheet errors.

1) Spreadsheet error incidence and persistence: Early studies in the area sought to qualify, categorize and quantify spreadsheet errors [16]–[19]. In summary, these studies have consistently found that over 90% of operational spreadsheets audited in various domains and contexts contained errors, and that they can sometimes have catastrophic consequences [20]. By way of understanding the underlying cause for these errors, there are also several taxonomies of spreadsheet errors [17]–[19]. Studies also reveal that spreadsheet users' abilities to find errors in their or others' spreadsheets are alarmingly low [21], [22], and that several factors (e.g., presentation of spreadsheets, user expertise, collaboration) affect their error finding performance [23]–[26]. Inspired by such findings, there are several efforts to manage spreadsheet errors.

2) Automated error prevention and detection techniques: One body of work aims to automatically detect, correct, and prevent various classes of spreadsheet errors which includes detecting and correcting unit errors in formulas [27], [28], or missing data [29] to detect errors and recommend fixes. Some others only detect errors and flag them to users; examples are CheckCell and ExceLint that detect anomalous cells in ranges [30], [31]. Finally, there are several error prevention efforts, mostly in the form of enabling users to adopt solid engineering practices. Approaches include writing assertions, specifications and test cases, refactoring, and documentation [5], [32]–[37].

B. Empirical studies of spreadsheet debugging

Debugging is defined as the detection, location, and elimination of errors in programs [38]. Among these three subactivities in debugging, most works described above only focus on error detection. Users then, manually or aided by debuggers, locate and fix errors, and often verify the fix and add a test case to prevent the recurrence of the bug.

Following this definition, Brown and Gould [39], and subsequently Hendry and Green [40], observed that spreadsheet users, when creating spreadsheets, adopted a variety of mechanisms to aid future debugging. For example, they broke down long calculations into multiple cells, so that the comprehension of formulas as well as inspecting intermediate results are convenient. In another study, Kankuzi and Sajaniemi [41] studied the mental models of spreadsheet users and found both domain-related and spreadsheet-related concepts (e.g., cell references, functions) in participant verbalizations during debugging tasks. Both these studies indicate the need to facilitate finding both domain-related and specific spreadsheet-related information during debugging.

Studies by Grigoreanu et al. [11], Ruthruff and Burnett [42] Chen and Chan [43], and Ragavan et al. [44] explicitly list information needs of spreadsheet users, including for debugging, and highlight the challenges for users in obtaining this information. We drew heavily from all these studies when designing FxD, with a specific focus on alleviating the barriers and providing the salient pieces of information needed for comprehending and debugging formulas.

C. Tools for spreadsheet debugging

However, ours is not the first tool for spreadsheet debugging; many other researchers have attacked the problem. The most common approach is fault localization where a combination of program slicing and constraint satisfaction (here, constraints are the expected and actual values of cells) is used to identify potentially erroneous cells. The technique has been used in goal-based debugging [45], interval-based fault isolation [46], mutation-based debugging [47], and constraint-based debugging [48]. In [49], Exquisite uses an AI-based approach. The commercial audit tool PerfectXL [50] is based on the PhD research of Hermans [51]. In general, the output of all these tools is a set of possible cells that are potentially erroneous. The user then investigates the recommendations and implements the fix. An exception is [45], where the user is also offered potential fixes rather than just the potential bug location.

Our work is complementary and distinct from these automated debugging approaches. Specifically, we do not take an automated approach to debugging: this follows [52]'s finding that spreadsheet users tend to over-rely on automated debugging tools and suffer productivity losses (such as when the tool is limited or offers incorrect suggests). Instead, FxD presents to users relevant information that will guide users' systematic debugging. The belief is that such aids will help end user programmers cultivate essential debugging skills [53].

D. Debugging functional programs

Spreadsheets are code [1], and spreadsheet programs written in the form of formulas are functional programs. We draw from the prior works on debugging functional programs to guide our design. In particular, we used trace-based debugging, where a user can transparently see the steps in the evaluation of a functional program, or its individual statements [54]. A challenge in trace debugging is that the traces can be too long and too broad, for long (e.g., too many computations in a formula) and nested expressions (e.g., formula refers to another formula, function invokes another function), respectively. We drew upon recent work by John Whitington [55] to simplify program traces, to manage both the depth and the breadth of the traces, as we will describe later. Other approaches to debugging functional programs include the use of REPL, forward and backward tracing (observing computation steps from output to expression and vice versa, respectively) and various approaches for stepping through calculations; we refer the reader to [56] for a summary of prior research in the area, in particular for program tracing and provenance [57], [58].

III. FxD Design and Implementation

A. Design Goals

Drawing from Eisenstadt's principles for debuggers [59] and the limitations of prior work [56], we aim to meet the following design goals in FxD:

G1 Allow full functionality at all times,

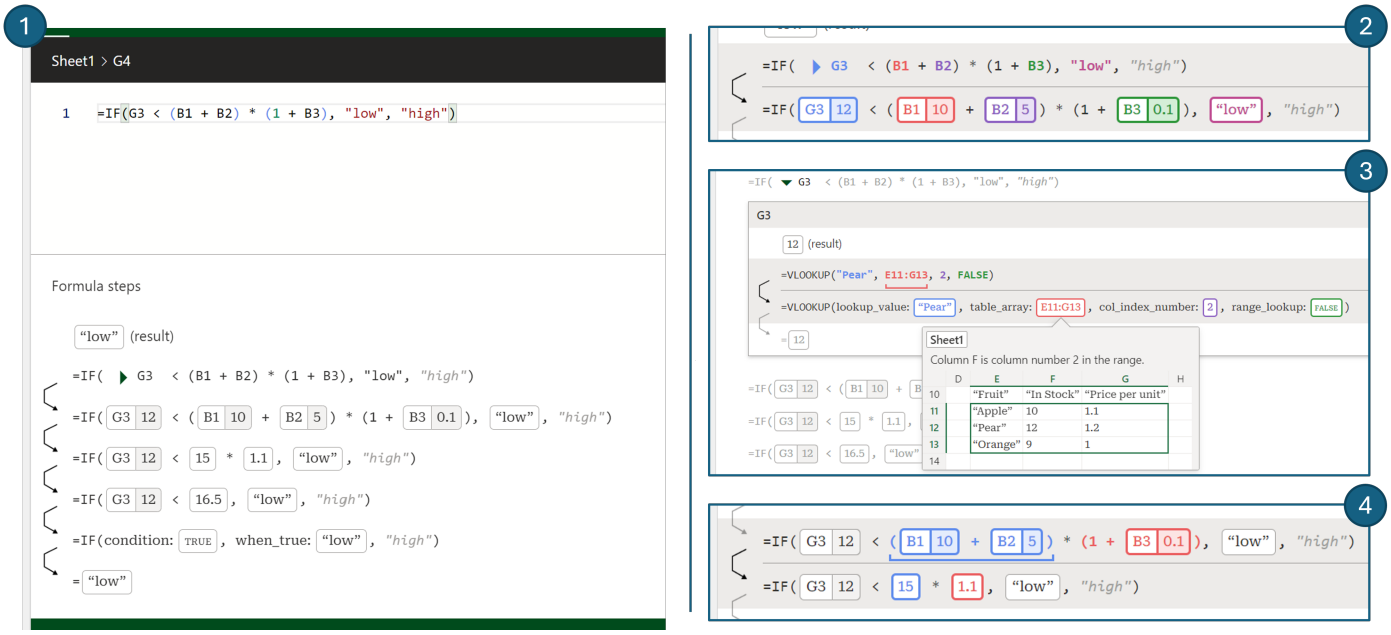

Fig. 1. FxD in action. Image 1 shows the full debugger window. Images 2-4 show aspects of the debugger in use.

TABLE I
INFORMATION NEEDS DURING FORMULA DEBUGGING.

User need	*Source*	*In FxD?*
Formula evaluation to check correctness	[44], [56], [60]	YES
Debug the source of an error	[44]	YES
Why is it this value not what I expected?	[44], [60]	YES
What is different between two formulas?	[44]	PARTIAL
What are the precedents of this formula?	[44], [60]	YES
What are the dependents of this cell?	[44], [60], [61]	NO

- G2 Allow inspection of any evaluable expression and not just the variables, where variables are formulas in cells,
- G3 Provide a variety of navigation tools at different levels of granularity,
- G4 Allow live debugging, without having to re-initialize the debugger on program edits,
- G5 Provide information users need when debugging spreadsheet behaviors (here formulas, Table I), and
- G6 Minimize cognitive loads, especially with complex formulas and long calculation chains.

B. Realization of design goals

To realize the design goals, we implement FxD with four central features, with each feature being directly addressed in our evaluation (Section IV). Figure 1 presents FxD and these features in-use, and we now describe each feature in turn.

1) ALWAYS-ON DEBUGGING*:* The FxD pane presents an editable formula at the top, with the corresponding execution steps displayed below (Figure 1.1). The execution steps are automatically visible to the user, without initialization G1. Additionally, the execution steps reactively update as the user edits the formula G4. As a user modifies potentially problematic code they can immediately inspect the results, aiding fault localization and verification.

2) FORMULA TRACING*:* FxD shows the steps in the evaluation of the formula being debugged; this is called tracing in the functional debugging literature [54]. We considered two design options: 1) show one execution step at a time (similar to step-through debugging), or 2) show all steps at once. The former is less efficient, while the latter can lead to information overload. We selected the latter to allow the user to quickly navigate to a step they are interested in G3. To lower the cognitive burden of complex formulas due to this information overload, FxD does not present every atomic evaluation step G6 [56], [62]. Instead, a single step in FxD can evaluate multiple expressions, either by chaining evaluation of operators of equal precedence, or by evaluating expressions in parallel subtrees. Further, FxD will collapse the evaluation of precedent formulas, indicated by a triangle, and expanding the triangle will reveal the formula trace as a recursive card (Figure 1.3).

3) SUB-FORMULA COLORING*:* To deal with the information overload that arises from showing all steps in the trace, as well as evaluating multiple expressions in a single step, FxD provides additional affordances for users to inspect the affected expression at each step of the trace G6. When hovering over a step, FxD will highlight the evaluated expressions in the step and use the same color to indicate the result; each expression is assigned a different color (Figure 1.2 & 4). This paradigm borrows from reference highlighting, commonly used in spreadsheets. The highlighting introduced through hovering can be "pinned" by selecting the arrow in the left margin. When there are many simultaneous reductions in a step it can be difficult to relate a particular expression

to its value, even with coloring. FxD will underline the original expression when hovering over an intermediate value, as shown in Figure 1.4.

4) INFORMATION INSPECTOR*:* FxD allows users to inspect all precedents of the formula or expression they are debugging G2. Additionally, FxD provides a range of affordances that offer debugging information to the user G5.

As discussed, precedent formulas can be inspected by expanding the trace card. Furthermore, FxD integrates reference provenance using "pills". A value that is derived from another cell is annotated with the cell reference, as shown in Figure 1.2 and the pill for 12, obtained by evaluating G3.

Ranges are an integral part of formulas and FxD implements a range preview, inspired by spreadsheet bubbles [63]. When hovering over a range a preview of the grid and surrounding context is shown to the user (Figure 1.3). Range previews are aware of the execution context; when the range appears as the table argument to a lookup function, the lookup column is labeled in the preview.

FxD also supports a variety of tooltips that describe functions and their parameters, or allow the inspection of array values which are normally truncated in the trace. Finally, when a formula evaluates to an error code, FxD indicates the step in which the error first occurs (Figure 2).

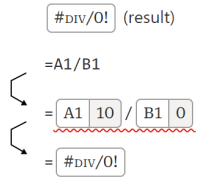

Fig. 2. FxD indicates subformulas that evaluate to an error

C. FxD *implementation*

We implement FxD as a JavaScript web add-in for Microsoft Excel. The interface is implemented using React and the debugger communicates with the spreadsheet using `office-js`. Formula evaluation is implemented using an instrumented interpreter, written in JavaScript. Every subformula in the program is assigned a unique label, and the interpreter populates a map from labels to values as it evaluates. The debugging interface uses this map and the abstract syntax tree (AST) to construct the steps shown to the user. Steps are allocated based on distance from leaves in the tree, for example, in Figure 1 the first step replaces the precedent cells with their values. In some cases, multiple levels in the tree are combined, such as sequences of additions.

D. FxD *limitations*

As FxD is a research prototype, there are certain technical limitations encountered during our evaluation.

Excel formulas are localized; however in FxD they are only presented in English, adding cognitive overhead for users who write formulas in other languages. FxD has tooltips for inspecting array values and ranges, however these tooltips are large for complex values. A further limitation with arrays is that many spreadsheet operations are vectorized such that they are automatically applied pointwise to elements in the array. Whilst FxD supports inspecting the input and output arrays to vectorized operations, there is no specific support for inspecting a particular point-wise operation. In Section VI we discuss other limitations discovered by participants associated with our design goals and features.

IV. EVALUATION: IN-LAB COMPARATIVE STUDY

To determine the effectiveness of the design of FxD we ran an in-lab comparative study to answer (1) if the features of FxD were useful for assisting users in debugging formulas, (2) if FxD provided adequate information to users to do so, and (3) if FxD fit within existing user workflows for debugging.

A. Participants

We recruited Excel users (n=12, 4 women, 0 non-binary, 8 men) via a social media posting on LinkedIn and email to spreadsheet users from past studies. They self-reported usage of at least 1 version of Microsoft Excel. All participants reported a lot of experience with spreadsheets, but with varying levels of usage (4 basic usage, 3 some advanced features, 5 many advanced features). The most commonly reported usage of spreadsheets was tracking (e.g., budgeting) and analyzing data. Finally, participants had varying levels of programming experience (4 having never programmed, 4 ranging between basic through advanced experience, and 4 programming professionally now or in the past).

B. Tasks

We considered two existing datasets, named A and B. For each, we prepared a spreadsheet containing descriptions of three tasks on the data. We seeded each task with a faulty formula. To obtain each faulty formula, we first prepared a formula to complete each task correctly, and then inserted one or more bugs. The type of formulas and bugs selected were sourced from existing posts on the Microsoft Tech Community forum [64], where the original poster presented a formula causing unexpected results and forum helpers provided verified fixes. We also took spreadsheet challenges posted online by spreadsheet influencers, which raised a problem that many viewers solved with their own formulas which had potential issues. We then adapted the root cause of the bug to our new datasets. The formulas involved advanced functions like VLOOKUP, FILTER, LET, IF, and typically computed a whole column (that is, the formula applies to each row in a table), but no LAMBDA functions were used.

We asked participants to localize the faults in each of the faulty formulas in the spreadsheets. If time remained, we asked them to fix the issue by modifying the formula.

Dataset A contains information on vehicles relocated in the Chicago, IL area.[1] We designed three tasks using this dataset:

- A1 Depending on the license `Plate` number, determine if a vehicle can enter on even or odd days (e.g., a plate ending in 1 can enter on odd days since 1 is odd).
- A2 For a specific `Plate` number, get the correct `State Code` from the secondary table.

[1]https://www.kaggle.com/datasets/chicago/chicago-relocated-vehicles

A3 For cars `Relocated to` and `from` the same street, was the car relocated to more than 10 street numbers away (e.g., 1 Main St. to 9 Main St. is FALSE)?

Dataset B contains property rental data in San Francisco. [2] We designed three tasks using this dataset:

B1 Find acceptable houses based on `Bedrooms` [at least 1], `Bathroom` to `Bedroom` ratio [over 0.5], `Property Type` [Apartments only], and `Room Type` [Entire home/apt only].

B2 Based on budget and planned stay, which places can you rent based on `Price` and `Minimum nights required`? Then get the 4 rentals that are the most northerly, southernly, easterly, and westerly places.

B3 For each `Property Type` in the table, calculate each average price.

C. Protocol

Participants were assigned A and B datasets through a counterbalanced design, such that half the participants received A then B, and the other half received B then A. Within these groups, participants were further balanced into two groups that determined if the first set of tasks would be completed with FxD (1 - FxD first) or only using default Excel (2 - Excel first). In summary, we had four evenly distributed sets of participants: {A first, B first} × {FxD first, Excel first}.

Participants completed three tasks in their first condition. They had 7 minutes per task to read the task, explore the spreadsheet, and debug the formula. If the participant believed they found the cause of the issue, they explained the cause to the experimenter who confirmed if they were correct. If they failed to complete the task within the limit, the task was recorded as incorrect. If any time was left, the participant was asked to fix the formula to meet the parameters of the task.

At the end of each task, we asked participants to complete a questionnaire to rate aspects of their experience with FxD or with Excel, based on the condition. After participants completed three tasks in their first condition, they would switch conditions and complete the remaining three tasks with the other dataset using the same protocol.

Before participants used FxD to complete their set of tasks, they completed a short tutorial introducing them to FxD. After all tasks were completed, we presented a final questionnaire and interviewed them about their experience debugging the formulas with both FxD and Excel.

D. Questionnaires and interview

After each task, participants rated how often they felt efficient, capable, confused, and frustrated while debugging. In the FxD condition, participants rated their agreement on if each of the 4 features of FxD was helpful in debugging the formula found in the task. They then rated if it was easy to use FxD or Excel and find bugs based on their current condition. Participants also rated the amount of information shown during the task by FxD and Excel.

[2]https://www.kaggle.com/datasets/karthikbhandary2/property-rentals

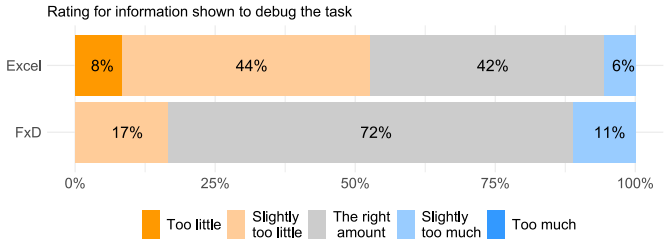

Fig. 3. Participant ratings for the amount of information shown to debug the task in each condition.

After all tasks were complete, they were presented with a final questionnaire that asked how likely they were to use FxD in the future to debug their spreadsheets and how likely they were to recommend FxD to co-workers. Additionally, they selected spreadsheet tasks they believed FxD could help with during their normal workflow.

Finally, participants were interviewed about their debugging experience with Excel and FxD. This interview covered topics of how FxD would fit within their daily workflow with spreadsheets, improvements FxD could provide their workflow, and feedback on improvements for FxD and Excel.

E. Study limitations

FxD was evaluated against a current default installation of Excel, without other debugging tools or add-ons that may have assisted participants in debugging their spreadsheets. Two of our participants noted that they use custom debugging tools in their normal debugging workflow but were unable to leverage them in the study. However, participants were able to use web search for unfamiliar formulas, which is a common strategy our participants employed. Participants noted that the time limit of seven minutes per task made it difficult to both debug and learn to leverage FxD's features. All participants reported they had a lot of experience with Excel, so our findings cannot be applied to novice debugging. Finally, participants used Microsoft Teams' screen sharing feature to control the experimenter's Excel application to complete tasks and use FxD, introducing input delay while debugging each task.

V. Quantitative Results

A. Task completion and timing

Fisher's exact test did not find a significant difference between using FxD and Excel for task completion (where completion is finding and describing the fault to the experimenter), nor did it find a significant difference in individual task performance. Overall, participants in both FxD and Excel conditions completed 21/36 tasks. Further, Fisher's exact test did not find a significant difference between conditions for fixing the formulas, with participants in the FxD condition having only 1 more task fixed than the Excel condition. A T-test did not find a significant difference between the time taken for completed tasks between conditions.

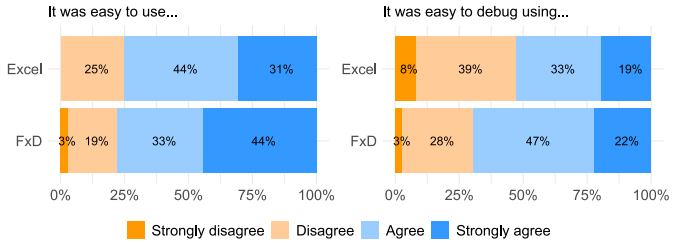

Fig. 4. Participant ratings for ease of use (left) and debugging (right) in each condition.

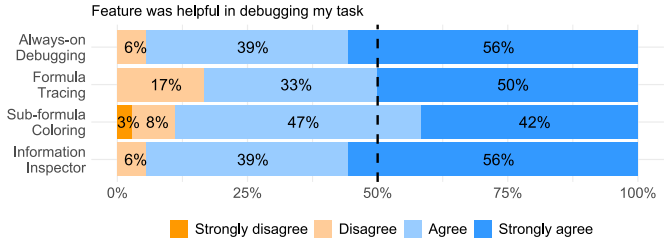

Fig. 5. Participants ratings for agreement that each feature of FxD was helpful for debugging each task in the FxD condition.

B. *Amount of information shown by* FxD *and Excel*

Participants rated the amount of information shown for each task by FxD (Fig. 3) with a median of "Just the right amount" (3, sd= 0.53) vs "Slightly too little" (2, sd= 0.73) for Excel. A Wilcoxon signed-rank test identified a significant difference between these conditions (S=895, p=0.0017). This might mean that FxD's features provides users with the level of information needed to debug issues in spreadsheets, which participants elaborate on in Section VI.

C. *Ease of use and finding bugs with* FxD *and Excel*

Participants in both conditions "Agreed" (median= 3, [FxD avg= 3.19, sd= 0.86], [Excel avg= 3.06, sd= 0.75]) that it was easy to use FxD and Excel during the tasks and "Agreed" (median= 3, [FxD avg= 2.89, sd= 0.78], [Excel avg= 2.64, sd= 0.90]) that it was easy to find bugs in each condition (Fig. 4). A Wilcoxon signed-rank test did not identify a significant difference between conditions for these categories.

D. FxD*'s helpfulness for debugging*

In the FxD condition participants were asked to rate each feature's helpfulness for debugging for each task by rating their agreement to the statement "I found this feature helpful during my debugging task" (Fig. 5). Participants reported a median of "Strongly agree" (4, sd= 0.61) for ALWAYS-ON DEBUGGING and INFORMATION INSPECTOR, between "Agree" and "Strongly agree" (3.5, sd= 0.76) for FORMULA TRACING, and "Agree" (3, sd= 0.74) for SUB-FORMULA COLORING.

Participants were asked if FxD would be helpful for various debugging and comprehension tasks they see in their normal workflows. 12/12 participants said FxD is helpful for fixing errors in spreadsheets, 10/12 said that FxD is helpful for comprehending spreadsheets and formulas received from others, 10/12 said that FxD is helpful for comprehending their own spreadsheets they had written in the past, and 11/12 said that FxD is helpful for writing new formulas. Participants wrote in a free-text option that they think FxD would be useful for complex 'Lambda' formula development, learning new formula syntax, and reading documentation for functions.

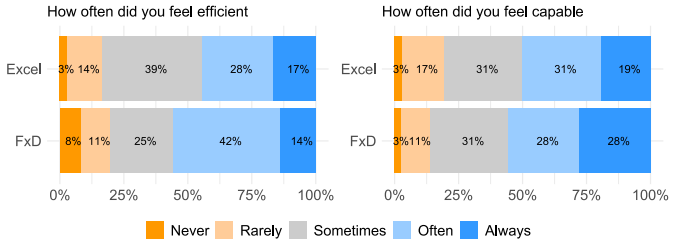

Fig. 6. Participant reported affect for felt efficiency (left) and capability (right) after each task for both conditions.

E. *Participant affect while debugging*

Participants rated how efficient, capable, frustrated, and confused they felt while using FxD and Excel to debug each task (Fig. 6). Participants rated they felt more often efficient with FxD (median= 4 (Often), sd= 1.13) vs Excel (median= 3 (Sometimes), sd= 1.02) and more often capable with FxD (median= 4 (Often)) vs Excel (median= 3.5 (between Sometimes and Often)). However, participants rated similar levels of confusion (median= 2 (Rarely)) and frustration (median= 2 (Rarely)) between conditions. A Wilcoxon signed-rank test failed to identify a significant difference between each condition and reported frequency of affect.

F. *Likely use and recommendation of* FxD

As a measure of user satisfaction we asked participants how likely they were to use and recommend FxD if a productionized version of the tool was released to the public. 4 participants said they were "Likely" (3) to use FxD and 8 said they were "Very likely" (4) to use FxD. The same results occurred for whether participants would also recommend FxD to co-workers who worked with spreadsheets.

VI. QUALITATIVE FEEDBACK

Through think-alouds during the tasks and interviews, we collected impressions and feedback on the features of FxD and how FxD fits within debugging workflows.

A. FxD *features feedback*

Participants spoke about the usefulness of the features provided by FxD (descriptions of each found in Section III) and gave feedback on how to improve the user experience, with P9 saying FxD's features were "really helpful" and that using FxD was "very intuitive".

ALWAYS-ON DEBUGGING (III-B1): Participants noted the usefulness of this feature for authoring formulas. P6 said that if you used FxD to write formulas and inspected the debugging steps shown, "we probably don't need to debug it, because

you would have written it in the correct manner the first time anyway." Participants offered feedback on how to improve this feature. Some participants wanted to edit values within the steps instead of on the spreadsheet or in the formula editor above the FORMULA TRACING steps (P1, 4). This could "make life easier, rather than moving between two different interfaces" for users during the debugging process (P4). P7 wanted to debug certain parts of a formula, rather than the entire cell's contents, since it could "make it easier to focus on bits of the formula that you're working on". P7 imagined an interaction of highlighting part of the formula, which would be evaluated in steps and shown in FXD.

FORMULA TRACING (III-B2): Our participants saw this feature as helpful to find what went wrong in each formula. P4 said that this feature enables them to "see what value is coming in through the intermediate steps and why I'm not getting the right output". P7 thought that having preceding steps to compare with the current step "makes it easy to connect the dots". P11 stated that these steps help visualize the error for users by marking exactly what step created the error.

However, participants thought that these steps can cause "information overload" (P1) for some formulas using 'LET', which can leave users feeling "daunted by the debugger" (P7). This issue occurs because each 'LET' binding gets a step in FXD, thus formulas with many declared variables introduce many steps. P8 said that when you have a logically complex formula, the steps might be "just too long' which "might make it very hard to compare across the steps". One solution that participants imagined was being able to view the data at a higher level of abstraction (P1, 7) by allowing steps to be "collapsible" (P1). A related affordance participants (P1, 7, 8, 11) described was to allow stepping through the steps one at a time, instead of all at once.

SUB-FORMULA COLORING (III-B3): P8 said that for many of the cases they had seen in the study this feature was "the most helpful". P11 noted during the study that when faced with a more complex formula, this feature gets "very useful" for navigation. P11 elaborated that these types of visualizations are "helpful for the programmer to keep track, because it's a lot of working memory that you need to hold and this is very cognitively demanding" and stated that "these kind of small visual cues to keep track is really helpful" for longer formulas. Feedback on this feature related to extending the highlighting beyond the formula steps. For example, P3 wanted to see the highlighting within the grid itself similar to how the current Excel formula bar does. P7 wanted the editable formula above the execution steps to get the same highlighting as you explore the steps so you can connect elements in each step with the original formula being debugged.

INFORMATION INSPECTOR (III-B4): Participants appreciated the ability to preview ranges with this feature. P11 said that large spreadsheets can require a lot of scrolling "up and down, left and right", but that FXD can help as "you can see in-context the other data around that cell. In that sense, you don't need to scroll". Others also described INFORMATION INSPECTOR as minimizing context switches and navigations between the spreadsheet and the formula section (P8), or between the spreadsheet and web search (P11).

Participants also gave feedback on how to improve this feature. For the table previews provided, P5 said they wanted the choice to have table headings instead of column letters in the preview to see the column context. P1 wanted the array preview to be expandable in case they wanted to see it "in more detail". Some participants wanted to see the data types expected for a specific reference or formula parameters since it could help users who might not know why a specific value is not working with their formula. Some participants said that the INFORMATION INSPECTOR could also enable navigation to the spreadsheet by being taken to the column or cells they click in the inspector, which would help users quickly jump to a cell to edit it (P1) or bring them to the correct spreadsheet in multi-sheet workbooks (P5).

B. Debugging workflows

Participants spoke about their current debugging workflows in Excel and how the debugger might improve them.

P7 starts their debugging workflow by first "eyeballing the formula" at a higher level and seeing if there are obvious issues "like missing dollar signs" (P7). However, a strategy participants reported and performed during the Excel condition is to break down complex formulas step-by-step manually to try to comprehend what was going wrong with the formula (P5, 8, 10, 12). P8 would "write formulas all over the place and try to just make sure each step is correct and then put them together.", and that with FXD and FORMULA TRACING, "you help me do this process".

Participants then discussed how they would use FXD as part of their spreadsheet debugging workflow, which P3 said was representative of their current workflow - "this (FORMULA TRACING) is the first part that I noticed and I'm already thinking awesome, that's what I'm doing in my head anyways" (P3). All participants were generally positive about adopting FXD into their workflow, with P2 noting that FXD "is a game changer". Participants thought FXD would be "very integrated" with their workflow (P9) and even replace other features in Excel, with P6 saying they would switch off "the formula bar on the top" in favor of FXD.

Participants stated that FXD was useful for comprehending formulas (P1, 2, 5, 7, 10-12), particularly during collaboration or sharing of spreadsheets. P1 said that they would recommend FXD to others since "most people have a lot of difficulty in understanding their formulas, I think it really gives a view into how the formula is working that people can learn from." P12 said FXD was useful for spreadsheets received from others since "it could be useful to have an interface that also explains the formulas you're not familiar with." Others (P1, 9, 11) also echoed the same sentiment, mentioning that FXD lets one "explore what somebody else's spreadsheet is doing" (P1) and gives "really good scaffolding to understand other's formulas" (P7). P7 also expected that FXD could be useful in helping them understand past formulas since they "probably will have forgotten" what they did after time away from that spreadsheet.

FXD provided affordances that participants said they would use to debug complex formulas and errors that might occur with them (P1, 5, 7, 9-11), which can help end-user programmers "try to do the task without needing programming, but also try to eliminate the error in complicated syntax" (P11). P1 stated that for debugging complex formulas like a 'LAMBDA' function, "there just aren't very good tools available" and that they would "love to have a better, more built-in way to do that sort of thing". Even simple features like underlining an evaluation step in red (similar to text editing environments marking misspelled words) when it evaluated into an error was helpful for large and complex formulas.

FXD was also seen as useful for creating and editing formulas (P5-7). P7 noted that FXD allowed them to make sure their newly created formula "does the steps in the right way that I was intending". P6 said that FXD facilitated more lightweight edits, since the user could "edit the formula in the editor without having to write back to the grid until you're confident with the answer", thus serving as a "test environment" for formulas that are "very calculation intensive".

However, participants also reported the need for more time to learn and get comfortable, and even attributed their performance on some tasks to this unfamiliarity (P1, 2, 7, 10, 11). For example, P1 said "I feel like I'm at a disadvantage because I'm still kind of new to the tool...but I think it would have (helped)." This might mean that careful consideration to the first-use tutorial is needed to educate users on how to leverage FXD's features to debug.

VII. Discussion: End-user debugger needs

Participant feedback described several important design principles for debuggers aimed at end-users that future debuggers should consider.

While most participants believed FXD showed useful information, several of our participants warned of information overload that could occur for more complex formulas they saw in the study. For example, task A1 contained a LET formula that led to some participants struggling with the amount of trace information shown in FXD. Participants expressed a need to start at a higher abstraction level first and then do a deeper dive into the details of a formula when needed. Presenting large formula execution traces on the onset of debugging a formula, or any functional program, might overwhelm a user. Thus, we believe end-user debuggers must consider the complexity of the formula being debugged and ease users into the debugging process to prevent information overload. One ability to assist in navigating larger traces is the ability to pin execution steps. FXD's affordance for pinning steps was seen as helpful to assist in preventing users from "losing their place" when scrolling through longer execution traces if they found an interesting step but wanted to continue to explore further steps while debugging.

Our participants also noted a need for explainability within formula debuggers. While FXD provides general context and a short natural language explanation of functions being debugged in a formula, more can be done to meet this need. This includes helping users understand what terminology used in parameters might mean. Other participants wanted to be told what types of values were expected in certain functions, to help them quickly hone in on issues like when a number is expected but a string is given, which can cause errors.

Our participants wanted a seamless connection between the debugger and the spreadsheet. Some participants wanted edits of values within the debugger steps to impact the spreadsheet itself, rather than just fixing the formula in the formula edit portion of FXD. However, this freedom can impact downstream formulas that depend on the result of the formula being debugged. These downstream effects must be made clear to users so they can be confident that their changes to one formula do not have unforeseen consequences. They also wanted the ability to click elements within FXD and be brought to the location of that element in the spreadsheet, instead of just peeking at the context which the INFORMATION INSPECTOR feature affords. This would also assist in navigating more complex spreadsheet situations, for example workbooks with multiple sheets that cross-reference each other. Users need highly integrated debuggers that fit within their spreadsheet workflows and bring them the flexibility to explore and modify the steps of the formula wherever they desire.

Finally, there is a user need to audit formulas written by non-humans, as some participants wanted AI-assistance in finding and fixing bugs they found in their spreadsheets. However, AI-generated code is not without bugs itself. Even as AI-generated formulas become commonplace, debuggers can still play a role in the auditing and verifying of generated formulas, especially as these new AI-affordances allow users to generate complex formulas that are difficult to comprehend without extra information.

VIII. Conclusion

Spreadsheet users need to be able to debug formulas to fix issues within spreadsheets they have written or have received from external sources. Existing affordances in Excel do not provide the level of information to do this satisfactorily for these users. Thus, we identified and designed 4 features for formula debugging in spreadsheets and implemented them as FXD, which adheres to design goals that provide live debugging and a feature-rich set of navigation tools to enable formula inspection. We evaluated FXD through an in-lab user study and found it provided a sufficient amount of information for users to debug spreadsheet issues, made participants feel more efficient and capable, and fit with in the spreadsheet user's workflow for debugging formulas. The feedback received from our participants shows a need for spreadsheet debuggers that provide the right amount of information to debug issues and provide adequate amounts of formula explanability, but do so without causing information overload to the user. Our findings are useful for the implementing of end-user spreadsheet debugging interactions and may even inform designs for understanding, auditing, and debugging AI-generated formulas.

References

[1] F. Hermans, B. Jansen, S. Roy, E. Aivaloglou, A. Swidan, and D. Hoepelman, "Spreadsheets are code: An overview of software engineering approaches applied to spreadsheets," in *FOSE@SANER*. IEEE Computer Society, 2016, pp. 56–65.

[2] A. Sarkar, S. S. Ragavan, J. Williams, and A. D. Gordon, "End-user encounters with lambda abstraction in spreadsheets: Apollo's bow or achilles' heel?" in *2022 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2022, pp. 1–11.

[3] A. J. Ko, R. Abraham, L. Beckwith, A. Blackwell, M. Burnett, M. Erwig, C. Scaffidi, J. Lawrance, H. Lieberman, B. Myers, M. B. Rosson, G. Rothermel, M. Shaw, and S. Wiedenbeck, "The state of the art in end-user software engineering," *ACM Comput. Surv.*, vol. 43, no. 3, apr 2011. [Online]. Available: https://doi.org/10.1145/1922649.1922658

[4] J. Ruthruff, E. Creswick, M. Burnett, C. Cook, S. Prabhakararao, M. Fisher, and M. Main, "End-user software visualizations for fault localization," in *Proceedings of the 2003 ACM Symposium on Software Visualization*, ser. SoftVis '03. New York, NY, USA: Association for Computing Machinery, 2003, p. 123–132. [Online]. Available: https://doi.org/10.1145/774833.774851

[5] F. Hermans, M. Pinzger, and A. Deursen, "Detecting and refactoring code smells in spreadsheet formulas," *Empirical Softw. Engg.*, vol. 20, no. 2, p. 549–575, apr 2015. [Online]. Available: https://doi.org/10.1007/s10664-013-9296-2

[6] S. Srinivasa Ragavan, A. Sarkar, and A. D. Gordon, "Spreadsheet comprehension: Guesswork, giving up and going back to the author," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, ser. CHI '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3411764.3445634

[7] Microsoft, "Create an array formula," 2022, accessed: 2023-05-09. [Online]. Available: https://support.microsoft.com/en-gb/office/create-an-array-formula-e43e12e0-afc6-4a12-bc7f-48361075954d

[8] ——, "Create a data type (power query)," 2022, accessed: 2023-05-09. [Online]. Available: https://support.microsoft.com/en-us/office/create-a-data-type-power-query-a465a3b7-3d37-4eb1-a59c-bd3163315308

[9] S. P. Jones, A. Blackwell, and M. Burnett, "A user-centred approach to functions in excel," *SIGPLAN Not.*, vol. 38, no. 9, p. 165–176, aug 2003. [Online]. Available: https://doi.org/10.1145/944746.944721

[10] Microsoft, "Announcing LAMBDA: Turn Excel formulas into custom functions," 2022, accessed: 2023-05-09. [Online]. Available: https://techcommunity.microsoft.com/t5/excel-blog/announcing-lambda-turn-excel-formulas-into-custom-functions/ba-p/1925546

[11] V. Grigoreanu, M. Burnett, S. Wiedenbeck, J. Cao, K. Rector, and I. Kwan, "End-user debugging strategies: A sensemaking perspective," *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 19, no. 1, pp. 1–28, 2012.

[12] S. Srinivasa Ragavan, A. Sarkar, and A. D. Gordon, "Spreadsheet comprehension: Guesswork, giving up and going back to the author," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–21.

[13] C. Chambers and C. Scaffidi, "Struggling to excel: A field study of challenges faced by spreadsheet users," in *2010 IEEE Symposium on Visual Languages and Human-Centric Computing*. IEEE, 2010, pp. 187–194.

[14] R. Caballero, A. Riesco, and J. Silva, "A survey of algorithmic debugging," *ACM Computing Surveys (CSUR)*, vol. 50, no. 4, pp. 1–35, 2017.

[15] B. A. Nardi, *A small matter of programming: perspectives on end user computing*. MIT press, 1993.

[16] R. R. Panko, "Spreadsheet errors: What we know. what we think we can do," *arXiv preprint arXiv:0802.3457*, 2008.

[17] E. Dobell, S. Herold, and J. Buckley, "Spreadsheet error types and their prevalence in a healthcare context," *Journal of Organizational and End User Computing (JOEUC)*, vol. 30, no. 2, pp. 20–42, 2018.

[18] R. R. Panko and S. Aurigemma, "Revising the Panko–Halverson taxonomy of spreadsheet errors," *Decision Support Systems*, vol. 49, no. 2, pp. 235–244, 2010.

[19] K. Rajalingham, D. R. Chadwick, and B. Knight, "Classification of spreadsheet errors," *arXiv preprint arXiv:0805.4224*, 2008.

[20] P. O'Beirne, F. Hermans, T. Cheng, and M. P. Campbell, "Spreadsheet horror stories," 2022, accessed: 2023-11-05. [Online]. Available: https://eusprig.org/research-info/horror-stories/

[21] D. F. Galletta, D. Abraham, M. El Louadi, W. Lekse, Y. A. Pollalis, and J. L. Sampler, "An empirical study of spreadsheet error-finding performance," *Accounting, Management and Information Technologies*, vol. 3, no. 2, pp. 79–95, 1993.

[22] R. R. Panko and R. H. Sprague Jr, "Hitting the wall: errors in developing and code inspecting a simple spreadsheet model," *Decision Support Systems*, vol. 22, no. 4, pp. 337–353, 1998.

[23] B. J. Reithel, D. L. Nichols, and R. K. Robinson, "An experimental investigation of the effects of size, format, and errors on spreadsheet reliability perception," *Journal of Computer Information Systems*, vol. 36, no. 3, pp. 54–64, 1996.

[24] D. F. Galletta, K. S. Hartzel, S. E. Johnson, J. L. Joseph, and S. Rustagi, "Spreadsheet presentation and error detection: An experimental study," *Journal of Management Information Systems*, vol. 13, no. 3, pp. 45–63, 1996.

[25] T. Chintakovid, S. Wiedenbeck, M. Burnett, and V. Grigoreanu, "Pair collaboration in end-user debugging," in *Visual Languages and Human-Centric Computing (VL/HCC'06)*. IEEE, 2006, pp. 3–10.

[26] R. McKeever, K. McDaid, and B. Bishop, "Can named ranges improve the debugging performance of novice spreadsheet users?" in *PPIG*, 2009, p. 15.

[27] C. Chambers and M. Erwig, "Automatic detection of dimension errors in spreadsheets," *Journal of Visual Languages & Computing*, vol. 20, no. 4, pp. 269–283, 2009.

[28] J. Williams, C. Negreanu, A. D. Gordon, and A. Sarkar, "Understanding and inferring units in spreadsheets," in *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2020, pp. 1–9.

[29] C. Negreanu, A. Karaoglu, J. Williams, S. Chen, D. Fabian, A. Gordon, and C.-Y. Lin, "Rows from many sources: Enriching row completions from wikidata with a pre-trained language model," in *Companion Proceedings of the Web Conference 2022*, 2022, pp. 1272–1280.

[30] D. W. Barowy, D. Gochev, and E. D. Berger, "Checkcell: Data debugging for spreadsheets," *ACM SIGPLAN Notices*, vol. 49, no. 10, pp. 507–523, 2014.

[31] D. W. Barowy, E. D. Berger, and B. Zorn, "ExceLint: automatically finding spreadsheet formula errors," *Proceedings of the ACM on Programming Languages*, vol. 2, no. OOPSLA, pp. 1–26, 2018.

[32] J.-C. Bals, F. Christ, G. Engels, and M. Erwig, "Classsheets-model-based, object-oriented design of spreadsheet applications." *J. Object Technol.*, vol. 6, no. 9, pp. 383–398, 2007.

[33] R. Abraham, M. Erwig, S. Kollmansberger, and E. Seifert, "Visual specifications of correct spreadsheets," in *2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)*. IEEE, 2005, pp. 189–196.

[34] M. Burnett, C. Cook, O. Pendse, G. Rothermel, J. Summet, and C. Wallace, "End-user software engineering with assertions in the spreadsheet paradigm," in *25th International Conference on Software Engineering, 2003. Proceedings.* IEEE, 2003, pp. 93–103.

[35] S. Badame and D. Dig, "Refactoring meets spreadsheet formulas," in *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 2012, pp. 399–409.

[36] M. Fisher, M. Cao, G. Rothermel, C. R. Cook, and M. M. Burnett, "Automated test case generation for spreadsheets," in *Proceedings of the 24th International Conference on Software Engineering*, 2002, pp. 141–153.

[37] D. Canteiro and J. Cunha, "Spreadsheetdoc: An excel add-in for documenting spreadsheets," in *Proceedings of the 6th National Symposium of Informatics*, 2015.

[38] IEEE, "ISO/IEC/IEEE international standard - systems and software engineering–vocabulary," *ISO/IEC/IEEE 24765:2017(E)*, pp. 1–541, 2017.

[39] P. S. Brown and J. D. Gould, "An experimental study of people creating spreadsheets," *ACM Transactions on Information Systems (TOIS)*, vol. 5, no. 3, pp. 258–272, 1987.

[40] D. G. Hendry and T. R. Green, "Creating, comprehending and explaining spreadsheets: a cognitive interpretation of what discretionary users think of the spreadsheet model," *International Journal of Human-Computer Studies*, vol. 40, no. 6, pp. 1033–1065, 1994.

[41] B. Kankuzi and J. Sajaniemi, "A mental model perspective for tool development and paradigm shift in spreadsheets," *International Journal of Human-Computer Studies*, vol. 86, pp. 149–163, 2016.

[42] J. R. Ruthruff and M. Burnett, "Six challenges in supporting end-user debugging," *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4, pp. 1–6, 2005.

[43] Y. Chen and H. C. Chan, "Visual checking of spreadsheets," *arXiv preprint arXiv:0805.2189*, 2008.

[44] S. Srinivasa Ragavan, A. Sarkar, and A. D. Gordon, "Spreadsheet comprehension: Guesswork, giving up and going back to the author," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, ser. CHI '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3411764.3445634

[45] R. Abraham and M. Erwig, "Goaldebug: A spreadsheet debugger for end users," in *29th International Conference on Software Engineering (ICSE'07)*. IEEE, 2007, pp. 251–260.

[46] Y. Ayalew and R. Mittermeir, "Spreadsheet debugging," *arXiv preprint arXiv:0801.4280*, 2008.

[47] B. Hofer and F. Wotawa, "Mutation-based spreadsheet debugging," in *2013 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2013, pp. 132–137.

[48] R. Abreu, A. Riboira, and F. Wotawa, "Constraint-based debugging of spreadsheets." in *CIbSE*. Citeseer, 2012, pp. 1–14.

[49] T. Schmitz and D. Jannach, "An AI-based interactive tool for spreadsheet debugging," in *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2017, pp. 333–334.

[50] S. Schalkwijk, F. Hermans, M. van der Ven, and H. Duits, "Auditing spreadsheets: With or without a tool?" in *Proc. 16th EuSpRIG Conf. Spreadsheet Risk Management*, 2015, pp. 29–46. [Online]. Available: https://arxiv.org/abs/1603.02261

[51] F. Hermans, "Analyzing and visualizing spreadsheets," Ph.D. dissertation, Delft University of Technology, 2012.

[52] A. Mukhtar, B. Hofer, D. Jannach, and F. Wotawa, "Spreadsheet debugging: The perils of tool over-reliance," *Journal of Systems and Software*, vol. 184, p. 111119, 2022.

[53] T. Lowe, "Debugging: The key to unlocking the mind of a novice programmer?" in *2019 IEEE Frontiers in Education Conference (FIE)*, 2019, pp. 1–9.

[54] S. Kamin, "A debugging environment for functional programming in Centaur," Ph.D. dissertation, INRIA, 1990.

[55] J. Whitington and T. Ridge, "Direct interpretation of functional programs for debugging," *arXiv preprint arXiv:1905.06545*, 2019.

[56] J. Whitington, "Debugging functional programs by interpretation," Ph.D. dissertation, University of Leicester, 2020. [Online]. Available: https://figshare.le.ac.uk/articles/thesis/Debugging_Functional_Programs_by_Interpretation/12696617

[57] R. Perera, U. A. Acar, J. Cheney, and P. B. Levy, "Functional programs that explain their work," *SIGPLAN Not.*, vol. 47, no. 9, p. 365–376, sep 2012. [Online]. Available: https://doi.org/10.1145/2398856.2364579

[58] U. A. Acar, A. Ahmed, J. Cheney, and R. Perera, "A core calculus for provenance," *J. Comput. Secur.*, vol. 21, no. 6, p. 919–969, nov 2013.

[59] M. Eisenstadt, "My hairiest bug war stories," *Commun. ACM*, vol. 40, no. 4, pp. 30–37, 1997. [Online]. Available: https://doi.org/10.1145/248448.248456

[60] A. Kohlhase, M. Kohlhase, and A. S. Guseva, "Context in spreadsheet comprehension," in *SEMS@ICSE*, 2015.

[61] C. Kissinger, M. Burnett, S. Stumpf, N. Subrahmaniyan, L. Beckwith, S. Yang, and M. B. Rosson, "Supporting end-user debugging: What do users want to know?" in *Proceedings of the Working Conference on Advanced Visual Interfaces*, ser. AVI '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 135–142. [Online]. Available: https://doi.org/10.1145/1133265.1133293

[62] D. Bajaj, M. Erwig, and D. Fedorin, "A visual notation for succinct program traces," *Journal of Computer Languages*, vol. 75, p. 101199, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2590118423000096

[63] N. Kakulawaram and J. Zamfirescu-Pereira, "Spreadsheet bubbles: Showing contextually relevant data during formula editing," Master's thesis, EECS Department, University of California, Berkeley, May 2021. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-145.html

[64] Microsoft, "Microsoft tech community," 2022, accessed: 2022-08-01. [Online]. Available: https://techcommunity.microsoft.com/