

Spreadsheet Bubbles: Showing Contextually Relevant Data During Formula Editing

*Nidhi Kakulawaram
John Zamfirescu-Pereira
Björn Hartmann, Ed.
Aditya Parameswaran, Ed.*

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2021-145

<http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-145.html>

May 21, 2021



Copyright © 2021, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Acknowledgement

I would first like to thank Associate Professor Bjorn Hartmann and Assistant Professor Aditya Parameswaran for their continuous guidance and support throughout this project. None of this would be possible without them. I would also like to thank PhD Candidate, J.D. Zamfirescu, who worked alongside me and graciously offered his expertise. He always offered great advice and made himself accessible to me whenever possible. While this project was done completely remotely due to the COVID-19 Pandemic, these three made this unexpected transition seamless for me.

Spreadsheet Bubbles: Showing Contextually Relevant Data During Formula Editing

by

Nidhi Kakulawaram

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley



Committee in charge:

Associate Professor Björn Hartmann, Chair

Assistant Professor Aditya Parameswaran

Spring 2021

The thesis of Nidhi Kakulawaram, titled Spreadsheet Bubbles: Showing Contextually Relevant Data During Formula Editing, is approved:

Chair		Bjoern Hartmann	Date	5/19/2021
		Aditya Parameswaran	Date	5/20/2021

University of California, Berkeley

Spreadsheet Bubbles: Showing Contextually Relevant Data During Formula Editing

Copyright 2021
by
Nidhi Kakulawaram

Abstract

Spreadsheet Bubbles: Showing Contextually Relevant Data During Formula Editing

by

Nidhi Kakulawaram

Master of Science in Computer Science

University of California, Berkeley

Associate Professor Björn Hartmann, Chair

Spreadsheets are one of the most common data management and analysis tools used today. In recent years, there has been focus on the extensiveness of spreadsheet errors with 90-95% of spreadsheets containing significant errors [9] [2]. We present Spreadsheet Bubbles, a novel user interface that extends the research on error prevention by proposing a mechanism that reduces their incidence. Through uncovering hidden data and enabling formula confirmation, Spreadsheet Bubbles' goals are to improve user understanding during the spreadsheet creation and manipulation process, and to ultimately reduce errors while increasing efficiency. Our tool succinctly presents data that is often hidden to enable the user to more easily confirm the correctness of their logic and formulae. In addition to making hidden data visible, we also make the hidden state of intermediary results visible. Nested formulae become increasingly difficult to understand and to debug because the user often struggles to isolate which function in the larger formulae is causing the incorrect results. Through Spreadsheet Bubbles' display of intermediary results, the user is able to visualize and validate formula fragments. The essence of our augmented user interface is making visually distal but logically relevant data proximate to reduce cognitive burden on the user.

Contents

Contents	i
List of Figures	ii
1 Introduction	1
2 Background: Spreadsheet Formulae	3
3 Related Work	5
3.1 Empirical Data on Spreadsheet Errors	5
3.2 Techniques to Reduce Spreadsheet Errors	6
4 Formative Study: Uncovering Opportunities for Design	8
5 Spreadsheet Bubbles Interaction Design	11
6 Implementation	15
6.1 Implementation Break-Down	15
6.2 Limitations	17
7 Discussion and Future Work	19
8 Conclusion	20
Bibliography	21

List of Figures

2.1	Spreadsheet Example of Student Test Scores	3
4.1	Weather Spreadsheet used for Pilot Study	8
4.2	User populates column A with all the US States from the filter-demo sheet	9
4.3	User uses COUNTIF to filter by the state, year, and natural disaster to count how many entries fit the requirements	9
4.4	User looks up which state had the biggest increase in storms	9
5.1	Design Mockup: Clickable column identifiers	11
5.2	Design Mockup: Populated Range View highlighting range selected	12
5.3	Design Mockup: Filter View with all predicates toggled on and showing unselected data	13
5.4	Design Mockup: Filter View hiding unselected data and predicate 3 toggled off	13
5.5	Lookups highlight and condensation feature	14
6.1	Implementation Diagram	15
6.2	Column Identifiers pop-up when a separate sheet is being referenced in the formula bar	16
6.3	Implemented Range View - highlighting selected data and showing context for what is not selected	16
6.4	Implemented Filter View - highlighting selected data and showing context for what is not selected	17

Acknowledgments

I would first like to thank Associate Professor Björn Hartmann and Assistant Professor Aditya Parameswaran for their continuous guidance and support throughout this project. None of this would be possible without them. I would also like to thank PhD Candidate, J.D. Zamfirescu, who worked alongside me and graciously offered his expertise. He always offered great advice and made himself accessible to me whenever possible. While this project was done completely remotely due to the COVID-19 Pandemic, these three made this unexpected transition seamless for me.

Chapter 1

Introduction

Spreadsheets are commonly used worldwide both in corporations and academia to manage and analyze data. In the financial sector, approximately 71% of organizations utilize spreadsheets as their main data management tool [4]. However, spreadsheets have a significant error rate, with 90-95% of spreadsheets containing errors and more than 1% of spreadsheet formula cells having errors [2] [8]. Recent work has audited spreadsheets to obtain error rates and to build a taxonomies of error types[2] [9]. Some examples of error classes include *reference errors* where users refer to the wrong set of data and *copy/paste errors* where a formula is wrong due to an inaccurate use of copy/paste.

Interestingly, a copy/paste error led one of the costliest corporate spreadsheet errors to date. JP Morgan Chase Co.’s London Whale Incident cost the company over 6 Billion dollars [14]. There are many other similar cases where spreadsheet errors had large financial consequences. Thus, it is vital to mitigate these spreadsheet errors. The goal of our research is to reduce formula errors and increase user efficiency; we introduce, Spreadsheet Bubbles, for this purpose.

While we originally thought to build a debugging or testing tool, we noticed that there were many prevention and subsequent debugging tactics already proposed [5] [12] [7]. Prior work mainly aims to reduce spreadsheet errors through changing how users organize their spreadsheets [5] or through post-hoc analyses that locate likely errors [12]. In contrast, Spreadsheet Bubbles aim to reduce errors while users are actively engaged in constructing and editing formulae. It does so by showing on-demand, interactive visualizations of the *working set* of data referenced or manipulated by a formula, while the formula is being edited. Thus, users can immediately see relevant data and results. We hypothesize that the display of the working set data is especially useful when the referenced data is not currently on screen, e.g., because a spreadsheet is large or has multiple sub-sheets.

Spreadsheet Bubbles draws inspiration from Code Bubbles [1], a User Interface that presents the current working set of code fragments in editable “bubbles”. Code Bubbles is especially helpful for programmers working with larger code bases when they are navigating between different files, classes, and functions. Developers can see and edit all relevant fragments of code on one interface; thus, offloading their working memory and saving time.

Similarly, Spreadsheet Bubbles provides context for formula formation and manipulation in Spreadsheets by making visually distal data but logically relevant data proximate. By showing the working set of data and subsequent formula results, the users can understand spreadsheet formulae and recognize errors more easily. We hypothesize that this ultimately reduces the cognitive burden on the user.

In addition to visually showing hidden data and indices, Spreadsheet Bubbles also presents intermediary results. From our formative study, we observed that users struggled to verify formulae that contained intermediary results. When functions become nested in the formula bar, the final results are dependant on intermediary results being correct. For example, if there is a Filter function nested within a Count function, the user needs to verify that the Filter function results first in order to verify that Count is being applied to the intended set of data.

We designed a novel User Interface, Spreadsheet Bubbles, augmented onto spreadsheets to visually pop-up “bubbles” to succinctly show hidden data and states with the goals of increasing user understanding, mitigating errors, and improving efficiency (Figures 5.1, 5.2, 5.3, 5.4).

Chapter 2

Background: Spreadsheet Formulae

Spreadsheet programming is done through formulae operating on ranges of data. Grid-like cells that make columns and rows contain the data. Formulae contain functions that manipulate or aggregate the data. A few examples of popular spreadsheet functions are SUM, COUNT, and FILTER. Through analyzing spreadsheet functions, we were able to distill functions into three different types:

- **Range Aggregation** (e.g. SUM, AVG, MAX, MIN or COUNT) - Range aggregation functions are mathematical functions used to retrieve information or aggregate on a range of cells.
- **Filtered Range** (e.g. FILTER, COUNTIF or SUMIF) - Filtered Range functions are functions that contain conditional statements used include or exclude certain cells in a specified range. For example, if the user wanted to count the number of students that passed test 4 in the example spreadsheet (Figure 2.1), they would utilize this formula: `=COUNTIF(E2:E9, E2:E9 > 70)`.

	A	B	C	D	E
1	Students	Test 1	Test 2	Test 3	Test 4
2	John	39	65	25	77
3	Joe	98	99	78	58
4	David	76	76	98	96
5	Lisa	87	88	92	67
6	Amy	56	85	59	87
7	Kim	68	58	69	96
8	Greg	99	78	87	89
9	Mark	76	81	77	74

Figure 2.1: Spreadsheet Example of Student Test Scores

- **Lookups** (e.g. VLOOKUP, HLOOKUP or INDEX) - For lookups, the user is trying to get an entry in a column or row with a specific search key. If the user is trying to retrieve Greg's test 4 score, they could use this formula: **=VLOOKUP("Greg", A2:E9, 5)**. This formula looks for a row with the entry "Greg" and gets the 5th element in that row.

Chapter 3

Related Work

3.1 Empirical Data on Spreadsheet Errors

According to several studies, approximately 90-95% of spreadsheets contain errors. In addition, spreadsheet formula cells have an error rate more than 1% [2] [8]. These errors can have very tangible and costly consequences. A survey of financial professionals found that 71% of organizations heavily depend on spreadsheets for a majority of their data management [4]. With current error rates, cases like the JP Morgan Chase's London Whale Incident are not uncommon. A spreadsheet copy/paste error cost the financial giant more than \$6 Billion [14].

Due to the frequency of errors, researchers began proposing taxonomies to categorize how users were making errors in the first place. Panko and Halverson [9] distinguished qualitative errors and quantitative errors. The former refers to errors that do not produce incorrect results but are risky practices that could lead to future errors. An example of a qualitative error is hard coding in numbers instead of utilizing cell references. If data is changed in the future, the formula would display previous calculations instead of current ones. Conversely, quantitative errors do lead to incorrect results and they can be further categorized into: logical errors, which arise from selecting the incorrect function or incorrectly using a function; mechanical errors, which arise from typing or pointing errors; omission errors, which arise from misinterpretation of the situation to be modeled. Spreadsheet bubbles is built to address quantitative errors so users can efficiently and easily recognize incorrect results.

A fundamental distinction in the human error literature is between “slips” and “mistakes” — in slips, the user has the correct mental model of the task but makes some error during the execution of a correct plan, e.g., a typo. In mistakes, the user has an incorrect mental model that leads them to an incorrect plan [10]. So, omission errors from the Panko/Halverson would be considered a mistake while mechanical errors would be considered a slip.

Errors in Operational Sheets [2] aims to take it a step further than error classification by analyzing the frequency of various types of errors. The paper audits and analyzes more than 50 spreadsheets from various sources with no more than 5 spreadsheets coming from the same

Error Type	Wrong Results	Poor Practices
Hard-Coding	31 (11% of total)	151 (74.8% of total)
Reference	137 (48.8%)	22 (10.9%)
Logic	89 (31.7%)	17 (8.4%)
Copy/Paste	14 (5%)	3 (1.5%)
Omission	7 (2.5%)	6 (3.0%)
Data Input	3 (1.1%)	3 (1.5%)
Totals	281	202

Table 3.1: **Instances of Errors and their Results** - While Spreadsheet Bubbles can prevent many error types through its visualizations, reference errors are especially targeted. A user can easily see what is included in their selection and what is not with the Range View. Table reproduced from Clarke et al., Errors in Operational Sheets [5].

source. They used a combination of auditing software and manual analysis to propose an updated taxonomy that has 7 classifications: logic errors, reference errors, placing numbers in a formula, copy/paste error, data input error, and omission error. Hard-coding errors were the most common occurring with 37.7% of the instances followed by reference errors at 32.9% of the instances and logic errors with 21.9% of the instances. The remaining four error types had minimal occurrences. For each of the seven error types, the paper also distinguishes how many of the errors led to incorrect results and how many were considered errors due to being poor practice. Their results can be seen in Table 3.1. Out of all errors that resulted in incorrect outcomes, reference errors made up almost 50% of the wrong results in the study. Spreadsheet Bubbles addresses this by making reference data visually present on the user's screen so they can verify that their reference is correct. This is especially helpful when referring to data on another sheet.

3.2 Techniques to Reduce Spreadsheet Errors

Due to the pervasiveness of spreadsheet errors, researchers in the field have proposed error reduction techniques [12]. Spreadsheet engineering is one of these techniques [5]. It adopts software engineering principles as a framework for spreadsheet programming best practices. Others argued for increase in standardized testing similar to coding test cases [11]. Panko[7] advocates that testing should account for 25-40% of spreadsheet development time. Spreadsheet auditing software is another proposed error reduction technique [12]. Most error reduction techniques involve debugging after the fact or refining spreadsheet interaction for prevention. In contrast, Spreadsheet Bubbles aims to reduce errors and increase understanding *during* the programming of spreadsheets.

Code Bubbles principles and techniques inspired our approach to increase efficiency and

mitigate spreadsheet errors [1]. Code bubbles is a user interface that presents the current working set of code fragments in “bubbles”. Developers spend significant time navigating between different files and folders. Code Bubbles allows all relevant functions and definitions to be on one interface as editable fragments. Similarly, Spreadsheet Bubbles populates the working set of data from anywhere in the spreadsheet in “bubbles”. This reduces the cognitive burden on the user to remember sheet and column names. In addition, it increases efficiency by presenting data the user would otherwise have to scroll or switch tabs for.

Current User Interface techniques exist in commercial spreadsheets to assist the user in visually confirming their formulas. For example, popular spreadsheets, like Excel and Google Sheets, color-code terms in a formula to show where the formula references are in the current spreadsheet view. Users can utilize this to visually confirm that their range selection is correct. However, it is only possible to view color coded cell ranges in the user’s current screen. Spreadsheet Bubbles would serve as an extension of this type of visualization to include hidden data and state. Separately, Microsoft has a tool coined the Workbook Relational Diagram that allows users to visualize spreadsheet dependencies from a high level using pointers [3]. Users can leverage this tool to create an interactive, graphical map of workbook dependencies in order to further understand and debug their workbook.

Chapter 4

Formative Study: Uncovering Opportunities for Design

We initiated the design process with a small ($N = 2$) pilot study. Inspired by spreadsheet computational tasks from *Understanding Data Analysis Workflows on Spreadsheets*, [13] the study encompassed common spreadsheet tasks like aggregation, search, manipulation, conditional statements, filter, sheet creation, and lookup. The study utilized a Weather spreadsheet that tracked extreme weather events over decades documenting location, date, duration, and more.

Task: “By state, how many more severe storms were there in 2010-2015 compared to 1980-1985? Then, find the state that had the largest increase in severe storms from the time period 1980-1985 to the time period 2010-2015”

One way to complete this task is to create a column of all US states (*Figure 4.2*), a column of the number of storms from 2010-2015 in each respective state (*Figure 4.3*), a column of the number of storms from 1980-1985 in each respective state, and then lastly, a column finding the difference between columns 2 and 3. Then, for part two of the task, the user had to look up the maximum difference in column 4 and identify which state this occurred at (*Figure 4.4*). These tasks had the users utilize sheet referencing, range selection,

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Disaster Number	IA Program Declared	PA Program Declared	HM Program Declared	State	Declaration Date	Disaster Type	Incident Type	Title	Incident Begin Date	Incident End Date	Disaster Close Out Date	Place Code	Declared County/Area	
2	4298	No	No	Yes	Yes	SD	2/1/2017	DR	Severe Storm(s)	SEVERE WINTER STORM	12/24/2016	12/26/2016	11860	Cheyenne River Indian Reservation	
3	4298	No	No	Yes	Yes	SD	2/1/2017	DR	Severe Storm(s)	SEVERE WINTER STORM	12/24/2016	12/26/2016	49700	Oglala Sioux Tribe of the Pine Ridge Reservation	
4	4298	No	No	Yes	Yes	SD	2/1/2017	DR	Severe Storm(s)	SEVERE WINTER STORM	12/24/2016	12/26/2016	99019	Butte (County)	
5	4298	No	No	Yes	Yes	SD	2/1/2017	DR	Severe Storm(s)	SEVERE WINTER STORM	12/24/2016	12/26/2016	99025	Clark (County)	
6	4298	No	No	Yes	Yes	SD	2/1/2017	DR	Severe Storm(s)	SEVERE WINTER STORM	12/24/2016	12/26/2016	99029	Codington (County)	
7	4298	No	No	Yes	Yes	SD	2/1/2017	DR	Severe Storm(s)	SEVERE WINTER STORM	12/24/2016	12/26/2016	99037	Dey (County)	
8	4298	No	No	Yes	Yes	SD	2/1/2017	DR	Severe Storm(s)	SEVERE WINTER STORM	12/24/2016	12/26/2016	99039	Deuel (County)	
9	4298	No	No	Yes	Yes	SD	2/1/2017	DR	Severe Storm(s)	SEVERE WINTER STORM	12/24/2016	12/26/2016	99041	Dewey (County)	
10	4298	No	No	Yes	Yes	SD	2/1/2017	DR	Severe Storm(s)	SEVERE WINTER STORM	12/24/2016	12/26/2016	99045	Edmunds (County)	
11	4298	No	No	Yes	Yes	SD	2/1/2017	DR	Severe Storm(s)	SEVERE WINTER STORM	12/24/2016	12/26/2016	99047	Fall River (County)	
12	4298	No	No	Yes	Yes	SD	2/1/2017	DR	Severe Storm(s)	SEVERE WINTER STORM	12/24/2016	12/26/2016	99049	Faulk (County)	
13	4298	No	No	Yes	Yes	SD	2/1/2017	DR	Severe Storm(s)	SEVERE WINTER STORM	12/24/2016	12/26/2016	99051	Grant (County)	
14	4298	No	No	Yes	Yes	SD	2/1/2017	DR	Severe Storm(s)	SEVERE WINTER STORM	12/24/2016	12/26/2016	99055	Haakon (County)	
15	4298	No	No	Yes	Yes	SD	2/1/2017	DR	Severe Storm(s)	SEVERE WINTER STORM	12/24/2016	12/26/2016	99057	Hamlin (County)	

Figure 4.1: Weather Spreadsheet used for Pilot Study

CHAPTER 4. FORMATIVE STUDY: UNCOVERING OPPORTUNITIES FOR DESIGN

A2 fx =SORT(UNIQUE('filter-demo'!F2:F))				
	A	B	C	D
1	State			
2	AK			
3	AL			
4	AR			
5	AS			
6	AZ			
7	CA			
8	CO			
9	CT			
10	DC			
11	DE			
12	FL			
13	FM			
14	GA			
15	GU			

Figure 4.2: User populates column A with all the US States from the filter-demo sheet

B2 fx =COUNTIFS('filter-demo'!\$F:\$F, "="&\$A2, 'filter-demo'!\$I:\$I, "=Severe Storm(s)", 'filter-demo'!\$K:\$K, ">=01/01/2010")									
	A	B	C	D	E	F	G	H	I
1	State	2010-2015							
2	AK	16							
3	AL	200							
4	AR	159							
5	AS	5							
6	AZ	22							
7	CA	6							
8	CO	0							
9	CT	42							
10	DC	2							

Figure 4.3: User uses COUNTIF to filter by the state, year, and natural disaster to count how many entries fit the requirements

E2 fx =INDEX(A1:A60, MATCH(VALUE(MAX(D2:D60)), D:D, 0), 1)					
	A	B	C	D	E
1	State	2010-2015	1980-1985	Difference	
2	AK	16	18	-2	OK
3	AL	200	108	92	
4	AR	159	93	66	
5	AS	5	2	3	
6	AZ	22	28	-6	
7	CA	6	46	-40	
8	CO	0	14	-14	
9	CT	42	5	37	
10	DC	2	4	-2	

Figure 4.4: User looks up which state had the biggest increase in storms

aggregation, filter, conditional formulae, and look ups. During the study, we observed the following user pain points and errors.

- When dealing with nested functions, it is difficult to verify intermediate results. In our study, the users had a Filter function nested in a Count function. They had to isolate the intermediary results for filter to verify that Count is being applied to the intended set of data. As functions and operations get more complicated, specifically with nested formulae, intermediary results become necessary to validate.
- Range selection and sheet referencing was another pain point. The users were often clicking and scrolling between various sheets and columns to select the intended set of data. There were errors in range selection that ultimately led to inaccurate results.
- The user rarely got the formula correct and the desired result on their first attempt. Often, the user had to debug their formula and consult outside resources to rectify the issue. There were two main types of formula issues we observed: syntax issues - e.g. misspelling a sheet name or not following expected formula format and logical issues - e.g. choosing an incorrect function or misunderstanding the question.

From these observations, we prioritized displaying intermediary results. This would make it easier for the users to isolate each function in order to ensure they achieved the intended result. To assist the user in debugging and understanding a filtered range, predicates are able to be toggled on and off. As previously mentioned, users frequently navigated between sheets to retrieve column and row information. This process takes extra time for the user and depends on their working memory to get data referencing in the original sheet correct. We decided to summarize and display the data referenced, regardless of where the data is located.

Chapter 5

Spreadsheet Bubbles Interaction Design

With Spreadsheet Bubbles, we set out to augment a spreadsheet with explicit support for each type of function. So, three function views were tailored to target pain points associated with each function type.

Each view is located within an ephemeral bubble that is visually positioned to indicate which portion of the formula the view is meant for. The bubble will disappear a few seconds after the cursor moves away unless pinned by the user to stay. Our aim is to aid the user in quickly identify syntactical errors and understand their logical errors to ease the process of spreadsheet creation and manipulation.

Most formulas include a range on which a function is enacted on. As the user is typing to potentially refer to data on another sheet, we provide clickable column identifiers so the user can easily select a range without having to go back and forth. This view can be seen in *figure 5.1*. Then, once the user completes typing the range, a range view is displayed with

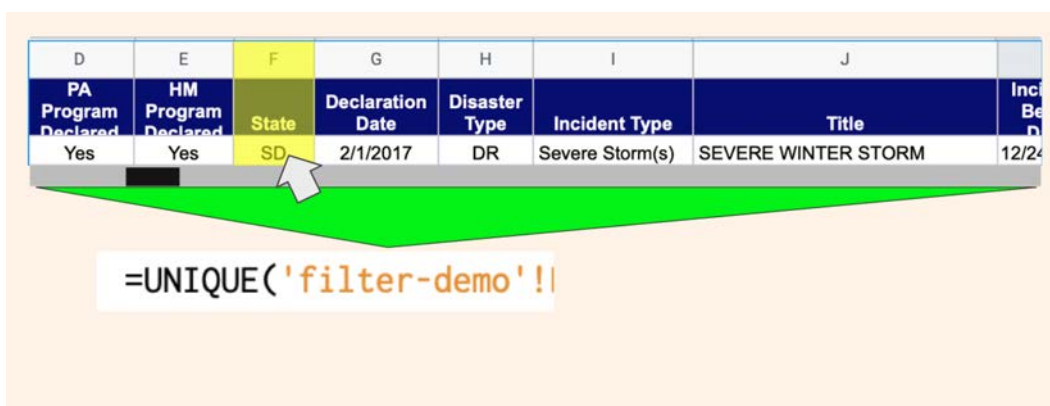


Figure 5.1: Design Mockup: Clickable column identifiers

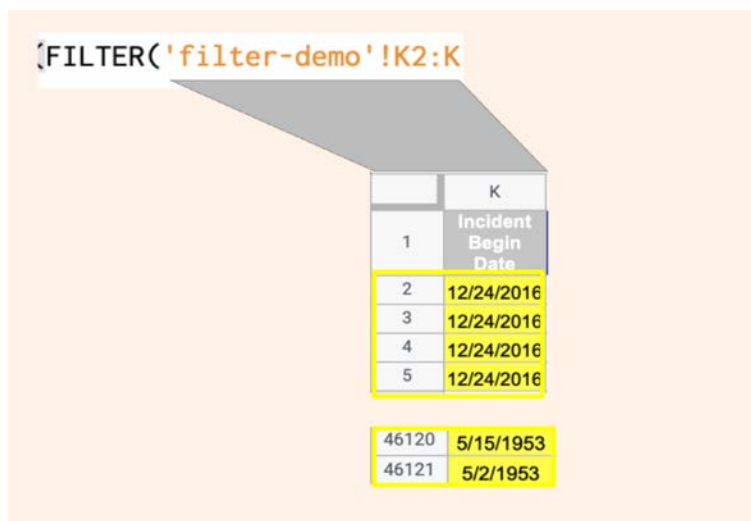


Figure 5.2: Design Mockup: Populated Range View highlighting range selected

the selected range highlighted (e.g. `Sheet3!A1:C20`) and shows relevant context for what was not selected. The range view (*figure 5.2*) is helpful so the user can verify what they selected in the highlighted range but also shows the outer boundaries of what the user did *not* select.

Next, we created a view for filter functions that contain conditional statements to include only certain cells in a specified range. The results of nested Filter functions (e.g. `MAX(FILTER(...))`, `AVG(FILTER(...))`) and conditional computation like `SUMIF` and `COUNTIF` are difficult to confirm correctness. There is hidden filtering of the range and the user is left with a number. The user must make sure the filtered results are correct to ensure the mathematical calculations are done on the right set of data.

For filtering ranges, we created a filter view that can display intermediary results. So, even if the filter is nested or hidden by a conditional computation, the user is able to validate the filter output on the specified range. The filter view is able to show selected results as well as context for what was not selected. Similar to the Range View, selected cells will be highlighted yellow and cells *not* selected are left as is. As an additional feature, the user is able to toggle on/off whether they want unselected cells to be displayed. In Figure 5.3 the user shows unselected data and highlights selected data, and in Figure 5.4 the user hides unselected data and only displays Filter function's results.

To assist in the debugging and/or the exploration process for the user, the filter conditions are able to be individually toggled on and off. In Figure 5.4, Predicate 3 is toggled off at the top. So, the third predicate that filters column I to only contain "Severe Storm(s)" is no longer applied on the range.

Lastly, we discuss the view for lookup functions like `VLOOKUP`, `HLOOKUP` and `INDEX`. Lookup functions find a row or column with a specified search key and then return an indexed element from that selected row or column. For lookups, we enable highlighting to

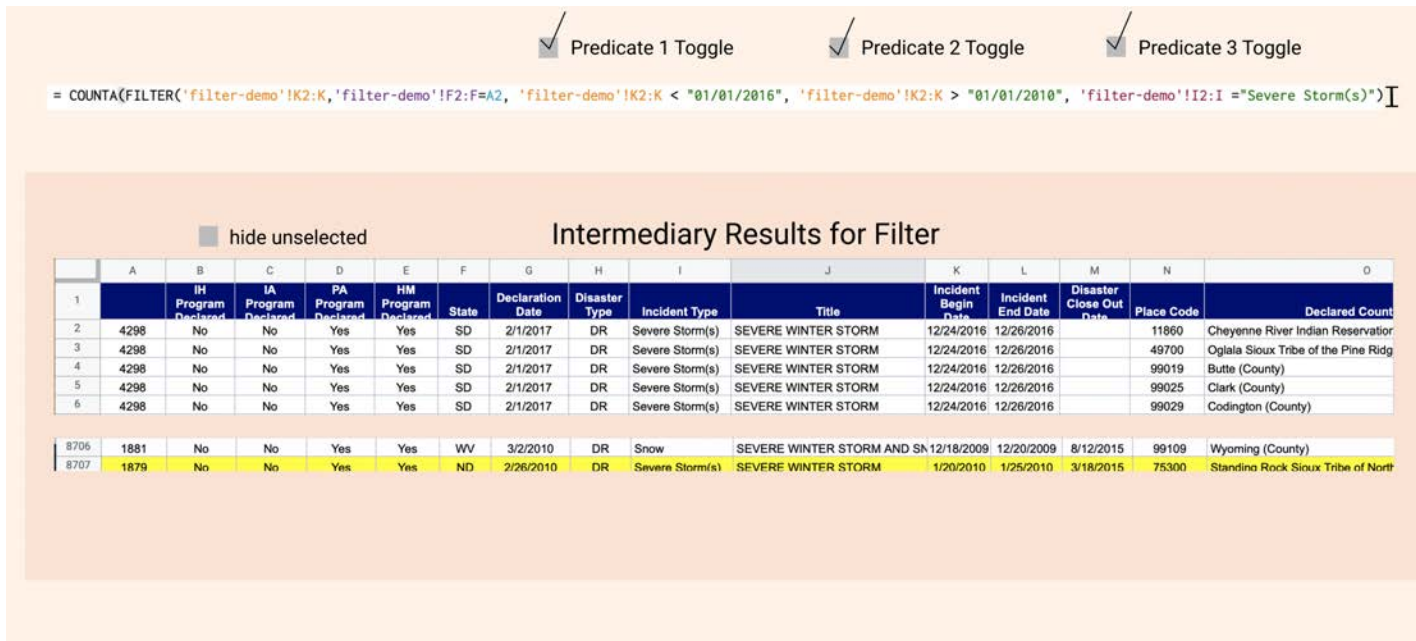


Figure 5.3: Design Mockup: Filter View with all predicates toggled on and showing unselected data

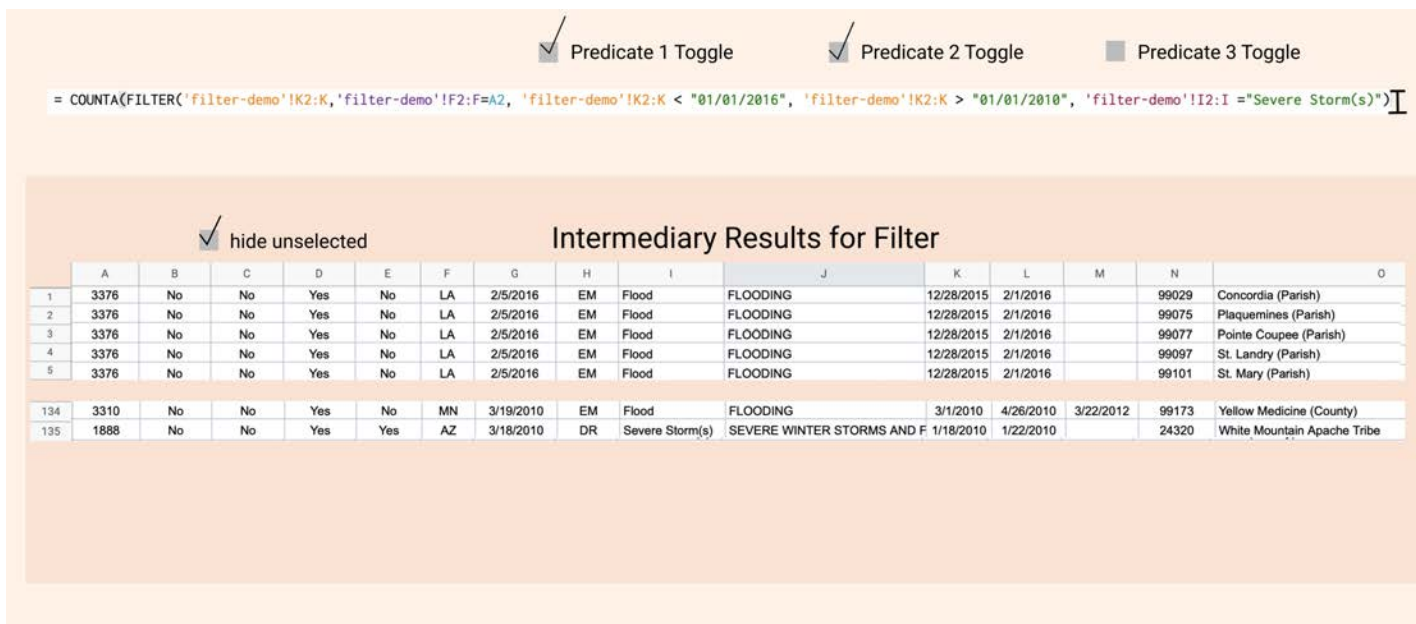


Figure 5.4: Design Mockup: Filter View hiding unselected data and predicate 3 toggled off

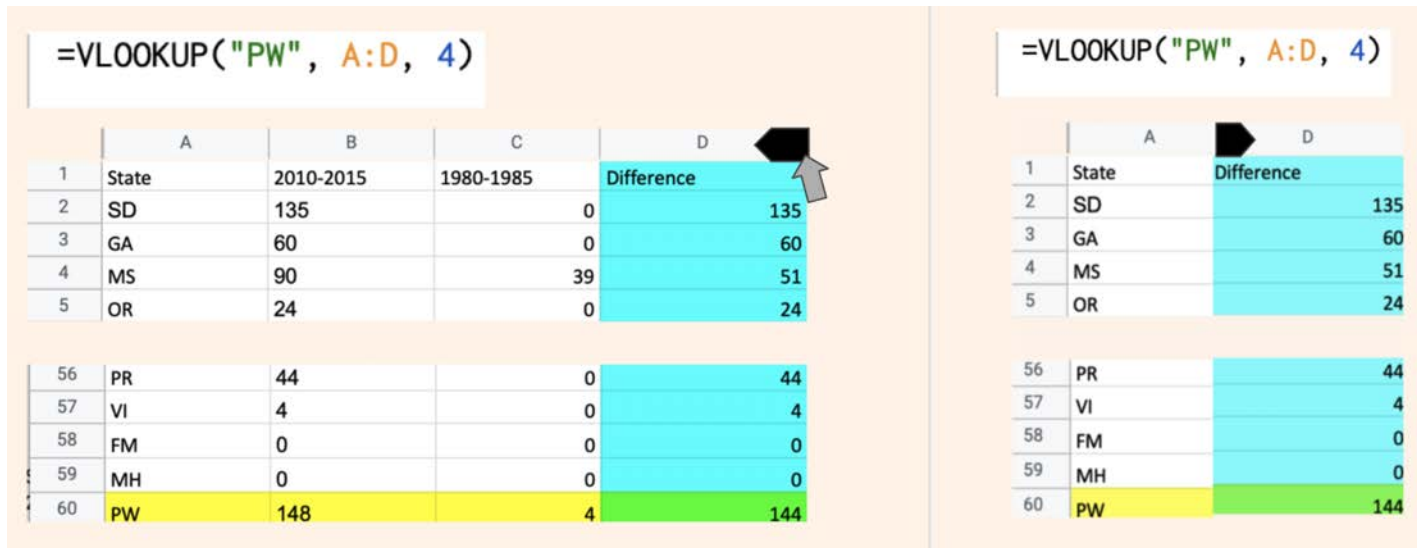


Figure 5.5: Lookups highlight and condensation feature

show exactly which row and column is being selected and allow sheet condensation to make verification as easy as possible for the user. For example, in the Figure 5.5, the search key is "PW" and the element in the 4th index of that row, 144, is to be returned.

Chapter 6

Implementation

6.1 Implementation Break-Down

We built Spreadsheet Bubbles on top of Luckysheet, a configurable, open source spreadsheet available online[6]. When the user edits a formula or selects a new cell, we parse the formula text to discern if and which view to present and what the bubble must contain. We then utilize HTML to display the Bubbles and its' contents.

If the user is in the process of referring to data from another sheet, Spreadsheet Bubbles populates all of the column names from that sheet in a bubble above the formula bar as shown in Figure 6.2. We do this by parsing the formula bar text every time the user presses a key to see whether a bubble must pop up. Generally, we pop up bubbles if the user types an exclamation point, a comma, or an end parenthesis. When the user types an exclamation point, they are referring to data in another sheet, so Spreadsheet Bubbles pops up the column identifier bubble for the sheet referenced. For example, if the user types in

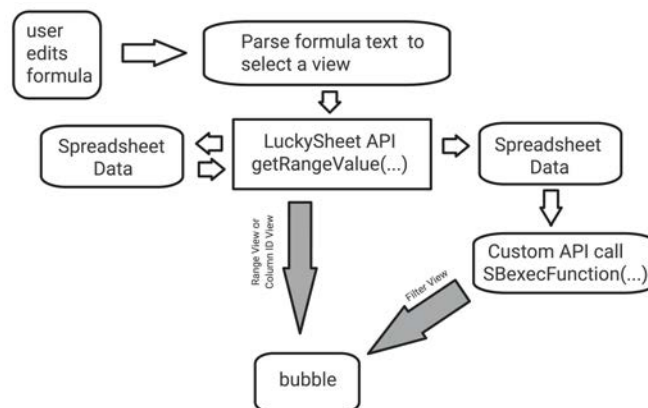


Figure 6.1: Implementation Diagram

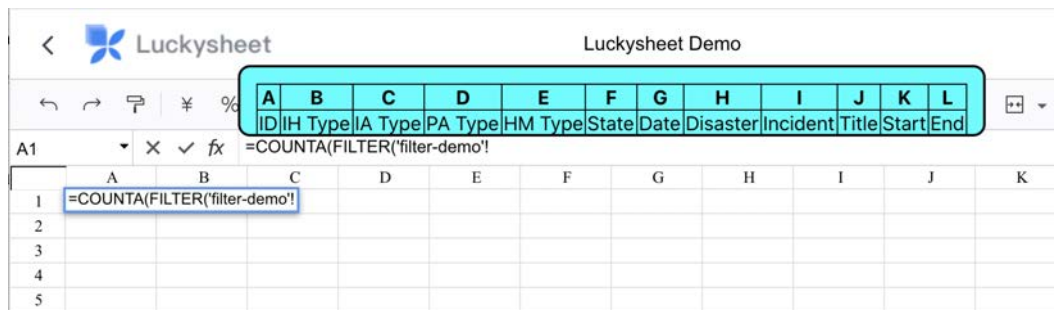


Figure 6.2: Column Identifiers pop-up when a separate sheet is being referenced in the formula bar

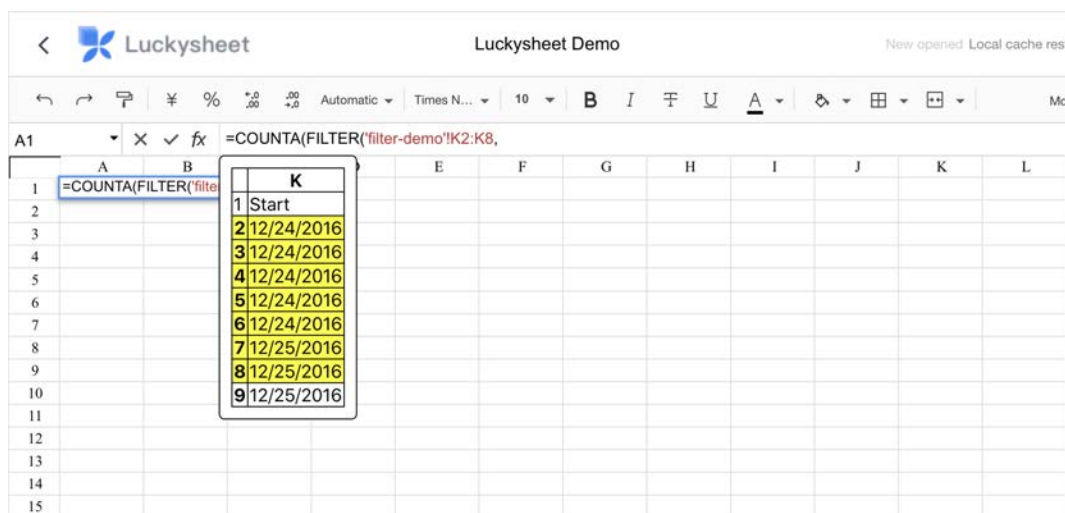


Figure 6.3: Implemented Range View - highlighting selected data and showing context for what is not selected

”COUNT(Sheet1!”, Spreadsheet Bubbles recognizes that the user will be referring to Sheet1 data and pulls column names from that sheet to make range selection easier (Figure 6.2).

In our original design (Figure 5.1), we meant for the column identifier view to be clickable, so that the column reference would automatically populate in the formula bar. However, our implementation at the moment is not clickable.

For Range Views, Spreadsheet Bubbles parses the formula bar to see if the last character type was either a comma or end parenthesis. If the user typed one of those two characters, the tool searches to see if a range was typed before the exclamation point or end parenthesis. Then, a bubble is populated with the range the user typed. To retrieve data from anywhere in the spreadsheet to display in the bubble, we utilized the `getRangeValue(...)` function

The screenshot shows the Luckysheet application interface. At the top, there's a navigation bar with the Luckysheet logo and 'Luckysheet Demo'. Below it is a toolbar with various icons for undo, redo, print, and other spreadsheet functions. The main area displays a spreadsheet with a filter view bubble. The formula bar at the top shows the formula: `=COUNTA(FILTER(filter-demo!K2:K8, 'filter-demo'!I2:I8 = 'SevereStorm(s)'))`. The spreadsheet data is as follows:

ID	IH Type	IA Type	PA Type	HM Type	State	Date	Disaster	Incident	Title	Start	End
24298	No	No	Yes	Yes	SD	2/1/2017	DR	SevereStorm(s)	SEVEREWINTERSTORM	12/24/2016	12/26/2016
34298	No	No	Yes	Yes	SD	2/1/2017	DR	Snow	SEVEREWINTERSTORM SNOW	12/24/2016	12/26/2016
44298	No	No	Yes	Yes	SD	2/1/2017	DR	SevereStorm(s)	SEVEREWINTERSTORM	12/24/2016	12/26/2016
54298	No	No	Yes	Yes	SD	2/1/2017	DR	SevereStorm(s)	SEVEREWINTERSTORM	12/24/2016	12/26/2016
64298	No	No	Yes	Yes	SD	2/1/2017	DR	SevereStorm(s)	SEVEREWINTERSTORM	12/27/2016	12/28/2016
74298	No	No	Yes	Yes	SD	2/1/2017	DR	Flood	FLOODING	12/29/2016	12/30/2016
84298	No	No	Yes	Yes	SD	2/1/2017	DR	SevereStorm(s)	SEVEREWINTERSTORM	1/1/2017	1/1/2017
94298	No	No	Yes	Yes	SD	2/1/2017	DR	Tornado	SEVERE WEATHER	1/1/2017	1/1/2017

Below the table, there is a tooltip explaining the FILTER function: "condition2 [Option] Additional rows or columns containing boolean values 'TRUE' or 'FALSE' indicating whether the corresponding row or column in 'range' should pass through 'FILTER'. Can also contain array formula expressions which evaluate to such rows or columns."

Figure 6.4: Implemented Filter View - highlighting selected data and showing context for what is not selected

from the Luckysheet API. Once the spreadsheet data is received, we display the range in the bubble highlighting the selected data. We include the surrounding unselected data as context, but do not highlight the cells to indicate that the cells are not a part of the selected range (Figure 6.3).

If the parsed formula text detects a filter formula like SUMIF, COUNTIF, or FILTER, the tool is directed to populate a filter view bubble. We use the `getRangeValue(...)` function from the Luckysheet API to retrieve the relevant spreadsheet data for the specified range. Unfortunately, the Luckysheet API does not expose their function execution to developers. So, we edited the files `functionImplementation.js` and `formula.js` in the source code to enable ample access to spreadsheet data and under the hood functions. Specifically, we created an accessible function called `SBexefunction` in `formula.js` to be able to execute the filter function on any range of data with any conditions we specify. Then, we had to allow the functions to return more than one entry in `functionImplementation.js` since the source code originally only allowed single dimensional arrays to be returned.

After retrieving the filtered data from our `SBexefunction`, Spreadsheet Bubbles displays it, highlights it, and surrounds it with context (Figure 6.4). Essentially, if a cell was selected in accordance to the filter conditions, we highlight the selected data and include the cells above and below it to show context with unselected data.

6.2 Limitations

Due to time constraints and the complexity of working with Luckysheet, not all features from our design in Chapter 5 were able to be fully implemented. We were unable to position the bubble underneath the specific part of the formula it is depicting. So, at the moment, the bubbles are stagnant. In addition, while the clickable column identifier above the formula

bar from Figure 5.1 appears, the user cannot click columns in the bubble to populate column names in the formula bar. Lastly, the Filter View does not currently have options to hide unselected data and toggle specific predicates on and off like in Figure 5.4.

Chapter 7

Discussion and Future Work

Spreadsheet Bubbles augments an enhanced User Interface onto existing Spreadsheet solutions. In the future, we see Spreadsheet Bubbles possibly existing as a Google Chrome extension working on Google Sheets. Once Spreadsheet Bubbles is able to be fully implemented, a user study is important to objectively test the error prevention and efficiency increase Spreadsheet Bubbles can achieve.

In the user study, the users would have two similarly difficult tasks on two separate Spreadsheets. The tasks and Spreadsheets must be different so solving one problem does not inform the other. For each task, we would observe the number of times the user goes back and forth between sheets, the length of time it takes to solve the problem, and pain points the user ran into. We would also ask users afterwards how Spreadsheet Bubbles changed their workflow and overall Spreadsheet manipulation experience. This user study would ideally be done with non-programmers that have experience in working with Spreadsheets. Based on the user study results, we would iterate on Spreadsheet Bubbles' design.

In the process of visualizing hidden data and state in spreadsheets, we uncovered more design questions. A computer screen has limited real estate and we have yet to figure out how to best optimize the space bubbles take up. In addition, due to the bubbles' limited size, we must efficiently summarize the spreadsheet data to convey the most salient information in our bubble.

Formula manipulation and creation in Spreadsheets can very quickly get complex and difficult to debug. More formula support is necessary, especially since Spreadsheets do not often give detailed, helpful feedback. In general, assistance and mechanisms of aide in the process of Spreadsheet creation and manipulation need to be explored more.

Chapter 8

Conclusion

We present Spreadsheet Bubbles, an enhanced User Interface, augmented onto spreadsheets with the goal of making the spreadsheet workflow more efficient and less error prone. Spreadsheet Bubbles uncovers the working set of hidden state and data to reduce cognitive burden on the user; Thus, users are able to visually confirm their logic with ease, and save time and effort by avoiding going back and forth between data sets.

Spreadsheets are here to stay. They are used in every industry for countless different purposes from financial modeling to inventory tracking to event planning. We must continue innovating and rethinking within this space. Spreadsheet Bubbles is only the start of leveraging an augmented User Interface to enhance the Spreadsheet user experience.

Bibliography

- [1] Andrew Bragdon et al. “Code bubbles: Rethinking the user interface paradigm of integrated development environments”. In: vol. 1. Jan. 2010, pp. 455–464. DOI: 10.1145/1806799.1806866.
- [2] Steve Clarke et al. “Errors in Operational Spreadsheets”. In: Jan. 2011, pp. 236–247. DOI: 10.4018/978-1-60960-577-3.ch011.
- [3] *Compare workbooks using Spreadsheet Inquire*. URL: <https://support.microsoft.com/en-us/office/compare-workbooks-using-spreadsheet-inquire-ebaf3d62-2af5-4cb1-af7d-e958cc5fad42>.
- [4] *Future of Financial Reporting Survey 2017 - FSN - The Modern Finance Forum*. June 2019. URL: <https://fsn.co.uk/research-papers/future-of-financial-reporting-survey-2017/>.
- [5] Thomas Grossman. “Spreadsheet Engineering: A Research Framework”. In: (Nov. 2007).
- [6] *Luckysheet*. Jan. 2021. URL: <https://mengshukeji.github.io/LuckysheetDocs/guide/>.
- [7] Raymond Panko. “Recommended Practices for Spreadsheet Testing”. In: (Jan. 2008).
- [8] Raymond Panko. “Revisiting the Panko-Halverson Taxonomy of Spreadsheet Errors”. In: (Oct. 2008).
- [9] Raymond R Panko and RP Halverson. “Spreadsheets on trial: A survey of research on spreadsheet risks”. In: *Proceedings of HICSS-29: 29th Hawaii International Conference on System Sciences*. Vol. 2. IEEE. 1996, pp. 326–335.
- [10] James Reason. *Human error*. Cambridge University Press, 1990.
- [11] Gregg Rothermel et al. “A Methodology for Testing Spreadsheets”. In: *ACM Trans. Softw. Eng. Methodol.* 10 (Jan. 2001), pp. 110–147. DOI: 10.1145/366378.366385.
- [12] Simon Thorne and D. Ball. “A Review of Spreadsheet Error Reduction Techniques”. In: *Communications of the Association for Information Systems* 25 (Jan. 2009), pp. 413–424. DOI: 10.17705/1CAIS.02534.
- [13] Pingjing Yang et al. “Understanding Data Analysis Workflows on Spreadsheets: Roadblocks and Opportunities”. In: (2020).

- [14] Arwin Zeissler, Daisuke Ikeda, and Andrew Metrick. “JPMorgan Chase London Whale A: Risky Business”. In: *Journal of Financial Crises* 1.2 (2019).