Understanding and Supporting Debugging Workflows in Multiverse Analysis

Ken Gu
kenqgu@cs.washington.edu
University of Washington
Seattle, Washington, USA

Eunice Jun
emjun@cs.washington.edu
University of Washington
Seattle, Washington, USA

Tim Althoff
althoff@cs.washington.edu
University of Washington
Seattle, Washington, USA

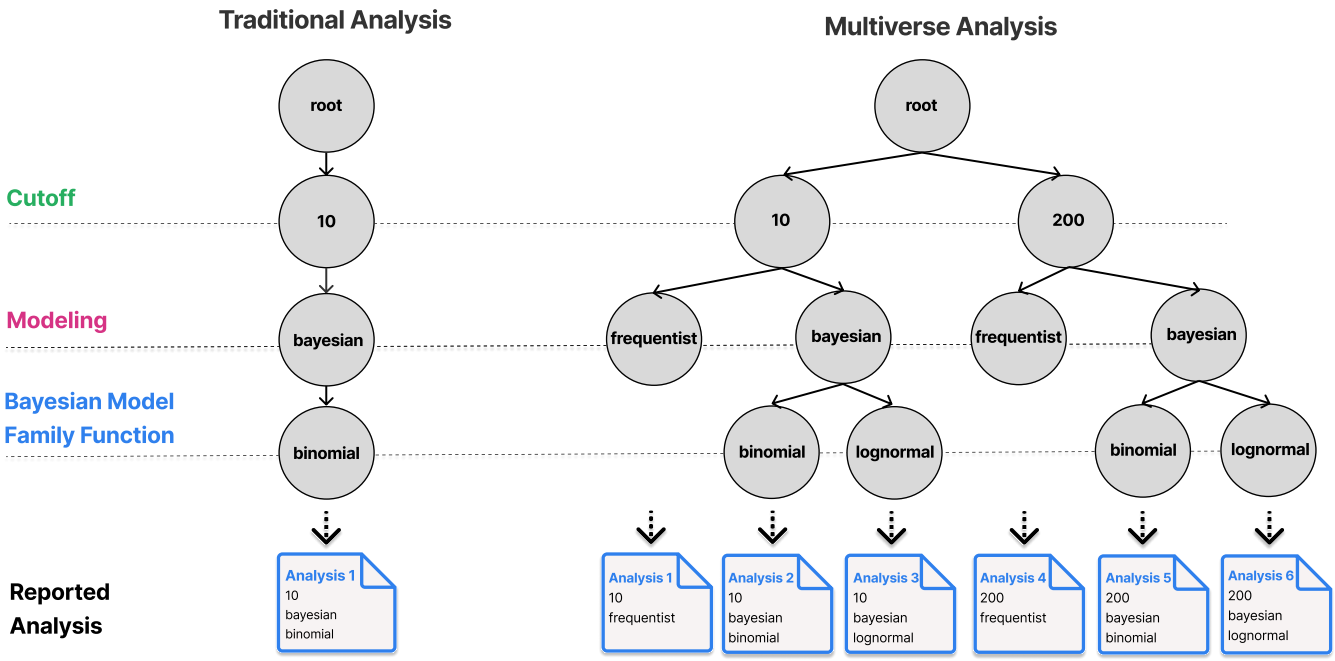

Figure 1: Overview of Multiverse Analysis. In traditional analyses, an analyst may consider multiple decisions in their analysis—data filter cutoff, statistical modeling approach (e.g., frequentist, Bayesian), and Bayesian family function (e.g., binomial, lognormal). Traditionally, analysts may conduct multiple analyses with different decision choices but ultimately report only one combination of decisions (a "universe"). In contrast, in multiverse analyses, analysts consider, conduct, and report all reasonable combinations of decisions.

ABSTRACT

Multiverse analysis—a paradigm for statistical analysis that considers all combinations of reasonable analysis choices in parallel—promises to improve transparency and reproducibility. Although recent tools help analysts specify multiverse analyses, they remain difficult to use in practice. In this work, we identify debugging as a key barrier due to the latency from running analyses to detecting bugs and the scale of metadata processing needed to diagnose a bug. To address these challenges, we prototype a command-line interface tool, Multiverse Debugger, which helps diagnose bugs in the multiverse and propagate fixes. In a qualitative lab study (n=13), we use Multiverse Debugger as a probe to develop a model of debugging workflows and identify specific challenges, including difficulty in understanding the multiverse's composition. We conclude with design implications for future multiverse analysis authoring systems.

CHI '23, April 23–28, 2023, Hamburg, Germany

ACM ISBN 978-1-4503-9421-5/23/04…$15.00
https://doi.org/10.1145/3544548.3581099

CCS CONCEPTS

• **Human-centered computing** → **User studies**; *Interactive systems and tools*; • **Software and its engineering** → Development frameworks and environments.

KEYWORDS

Multiverse analysis, statistical analysis, debugging, workflows, analysis authoring

ACM Reference Format:
Ken Gu, Eunice Jun, and Tim Althoff. 2023. Understanding and Supporting Debugging Workflows in Multiverse Analysis. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23), April 23–28, 2023, Hamburg, Germany*. ACM, New York, NY, USA, 19 pages. https://doi.org/10.1145/3544548.3581099

1 INTRODUCTION

Even when trained analysts are given the same analysis task and dataset, they make different, sometimes conflicting, conclusions [17, 44, 50, 51]. While it is not expected that different analysts when given only a dataset and a broadly defined task are to arrive at the exact same results, the level of variability is surprising. These divergences may even contribute to reproducibility crises across scientific disciplines [9, 42]. How could this be? Researchers believe that the flexibility in analytical choices (e.g., data filtering, statistical modeling approach, model parameters) is a key contributor. For example, analysts leverage their unique belief systems, domain knowledge, expertise, understanding of the problem, and exploratory results to justify their analytical decisions [29, 37]. Additionally, analysts only report the result of one set of analysis decisions despite exploring multiple combinations.

As a response to these problems, prior work has proposed multiverse analysis [52, 55] as a promising solution. Multiverse analysis, in contrast to traditional analysis, is a statistical analysis paradigm that involves considering, specifying, and reporting results from all combinations of key decision options (Figure 1 right). Multiverse analysis reveals how sometimes arbitrary decisions may affect an analysis conclusion. Moreover, by documenting and accounting for all reasonable decision options, multiverse analysis, and related approaches such as sensitivity analysis, improve transparency and robustness of statistical analyses and could prevent future reproducibility crises.

Despite the many benefits of multiverse analysis, authoring a multiverse analysis remains challenging. Authoring multiverses is difficult because analysts must explicitly enumerate decisions and the options for those decisions, write programs that generate additional programs or scripts for each individual combination of options, compare and jointly interpret statistical results across all combinations of decision choices, and iteratively debug and refine all the above. Recent work in the HCI community and beyond provide tools to ease some of the challenges in the authoring process: Boba [38], `multiverse` [48], rdfanalysis [24]. However, multiverse analysis remains difficult to adopt for many analysts. What are authoring challenges that, if addressed, could lower the barriers to authoring multiverse analyses? Prior work [48] and our own correspondences with multiverse tool developers and multiverse practitioners have identified debugging as a central challenge.

In this work, we target multiverse debugging as a key challenge. Based on prior work [48], our experiences, and with correspondences with multiverse practitioners and tool developers, we develop an initial model of debugging workflows in multiverse analysis (Figure 3). We find that analysts tend to focus on debugging a single analysis at a time (a "universe"). Even debugging a single universe script is time-consuming due to the need to triage and fix code. The scale of multiverse analyses, which can be on the order of tens of thousands of universes [37], exacerbates this problem and introduces additional cognitive burdens, such as keeping track of how many unique errors there are, which set of universes these correspond to, and what portion of analyses are buggy. Based on our initial workflow model, we identify three unique challenges of debugging in the multiverse paradigm:

Challenge 1 — Detecting bugs takes a long time during the slow execution of a multiverse (Figure 3D1).
Challenge 2 — Diagnosing the source of a bug to a specific decision choice or set of choices (i.e., singular universe) is hard amongst thousands of universes (Figure 3D2).
Challenge 3 — After fixing a bug in a single universe (Figure 3D3), the analyst needs to remember changes and understand how to propagate them to the rest of the multiverse (Figure 3D4), which increases cognitive load and creates opportunities for error.

Although existing debugging tools and workflows help analysts fix a bug in a specific universe, determining what universe to focus on and subsequently propagating one universe's changes to other universes that share the same error, remain under-supported.

To address these initial challenges, we prototype a debugging tool, Multiverse Debugger (Section 3). Multiverse Debugger is a command-line interface (CLI) tool that extends Boba [38], an an existing open-source tool that has already been employed in a large real-world study [49]. Multiverse Debugger has three key features, each of which addresses a challenge: (i) execution of a a significantly smaller set of decision choice combinations to facilitate fast iteration (Challenge 1) (ii) aggregation of error messages across a multiverse analysis (Challenge 2), and (iii) propagation of edits made to the rest of the multiverse (Challenge 3).

Using this tool as a probe, we conduct a qualitative lab study with 13 analysts to explore multiverse debugging in greater depth (Section 4). This lab study confirms Challenge 1 and Challenge 2 and we find Multiverse Debugger's features benefit analysts in diagnosing multiverse error messages and quickly detecting bugs. We observe that Challenge 3 is not a central concern to analysts as, prior to propagating bug fixes, analysts already struggle with understanding the composition of the multiverse (i.e., the multiverse analysis tree in Figure 1), which is critical in their efforts to diagnose multiverse error messages. We also observe analysts, inspired by Multiverse Debugger, favor selective execution of a subset of universes in the debugging process, which current tools do not yet support.

Based on these findings (Section 5), we update and extend our model of the multiverse debugging workflow and associated challenges (Figure 6). In addition, we discuss (Section 6) a set of design implications that include helping analysts better understand the composition of the multiverse and supporting analysts in navigating their multiverse analysis.

This paper contributes the following:

(1) Findings from a qualitative lab study that reveal open challenges in multiverse debugging,
(2) A publicly available open-source prototype of Multiverse Debugger that addresses some of these challenges and lab

study results that evaluate to what degree our prototype's features can alleviate them [1],
(3) A model of the key operational steps in multiverse debugging workflows and associated challenges, and
(4) A set of design implications for how to better support debugging for multiverse analysis authoring.

2 BACKGROUND AND RELATED WORK

2.1 Debugging in Software Engineering

Debugging is challenging and time-consuming. In prior works aimed to understand debugging in software engineering, developers reported spending 20% to 60% of their time debugging [10]. This has been later confirmed in a study analyzing real-world developer debugging sessions [8].

A central challenge in debugging is the "large temporal or spatial chasms between the root cause and the symptom" [34]. Based on a prior lab study, researchers detailed the mechanisms of debugging as involving the processes of searching, relating, and collecting information of perceived relevance, in which the development environment plays a central role in influencing developers' perceptions [35]. In other studies on general software development, it was discovered that a significant amount of mental effort is spent in understanding how a program works via searching relevant software artifacts, and inspecting source code/documentation [40, 57].

With this understanding, multiverse debugging is likely to exacerbate the problems of traditional debugging workflows. There are more analyses to work with, more meta-data per analysis in the form of associated decision options which can affect the presence of bugs, and shared relationships between the collection of scripts that need to be considered. All this information, if not presented well, can make the process of collecting and relating relevant information significantly harder. We contribute the first user study to explore and model debugging behavior and challenges in the context of a multiverse analysis workflow.

2.2 Multiverse Analysis

Multiverse analysis [52, 55] aims to have the analyst consider all reasonable decisions and combinations of decision options a-priori while then conducting and reporting all considered analyses. "Reasonable" here means actions with firm theoretical and statistical support [53]. Moreover, a decision in the multiverse paradigm is any decision an analyst may consider in an analysis. These decisions (e.g., Cutoff, Modelling, and Bayesian Model Family Function in Figure 1) are wide-ranging and can cover data collection and wrangling, statistical modeling, inference, and evaluation. For each decision, there are decisions options, defined as the alternatives that the specific decision could take.

Because multiverse analysis considers all reasonable combinations of decision options, there is a combinatorial explosion in the number of universes as more decisions are involved. For example, a multiverse of 5 decisions each with 4 options would result in $4^5 = 1024$ universes. Prior work has estimated that multiverses in practice contain between hundreds and hundreds of thousands of individual analyses [37].

[1]The code for our prototype is publicly available at https://github.com/behavioral-data/multiverse-tooling.

As multiverse analysis has gained recognition and adoption [16, 18, 19, 30, 45, 46], associated workflows, tools, and visualization techniques have been developed [20, 24, 25, 38, 48]. Recent work on multiverse authoring has identified debugging as an important, unaddressed challenge [48]. The present work extends this prior work by contributing the first user study and first prototype specifically focused on the unique debugging challenges that the multiverse paradigm presents.

2.3 Tools for Multiverse Analysis

Traditionally, analysts must consider hundreds if not thousands of universes if they were to perform a multiverse analysis. This can result in a large set of mostly similar universe scripts which, with so many variations, is difficult to maintain [32]. On the other hand, an analyst can write a complex series of control flow logic in one large script [56] but this makes it hard to selectively run an individual universe. Multiverse authoring tools make it easier to specify a multiverse analysis and execute it. These tools simplify specifying decisions by introducing special syntax to specify decisions, decision options, and constraints between decision options in one central file. In these general multiverse authoring tools, namely, Boba [38] and `multiverse` [48], a common authoring workflow is observed (Figure 2). Analysts specify their multiverse in a central multiverse specification containing different code snippets for different decision options (Figure 2A). Afterwards, the multiverse specification is compiled into universes (Figure 2B). A universe contains an instantiation of decisions' options and compilation also produces a specification summary enumerating each universe's decision options (Figure 2C). After the compilation step, the universes are executed which each produces an error message (Figure 2D) and other outputs (Figure 2E).

While largely following the authoring workflow in Figure 2, `multiverse` aims to support the iterative workflow of a computation notebook. It is a R package that works in RMarkdown notebooks. To specify decisions, execute the universes, and gather results, analysts call `multiverse` methods. The notebook acts as the multiverse specification. The compilation is implicitly performed under the hood when universes are executed.

Meanwhile, in Boba, the multiverse specification is one central *template* file. Boba places specific domain-specific language (DSL) directives that indicate how different chunks of code fit together. This has the benefit of being programming language agnostic, treating non-multiverse code as raw strings. Nevertheless, because of these directives, the *template* file is not executable and cannot leverage any of the advanced debugging features in modern interactive development environments (IDEs). Boba provides additional command-line commands to compile and run the multiverse. Analysts run `boba compile` to compile their multiverse specification. To execute the multiverse after compilation, Boba provides the command `boba run` to execute a range of or all the universes. When executed with Boba, each universe's standard output messages and standard error messages are saved to a corresponding output file and can be gathered in a CSV file.

However, other than collecting error messages as entries in a table, both tools do not provide any additional support for multiverse debugging workflows. Our work extends the authoring framework

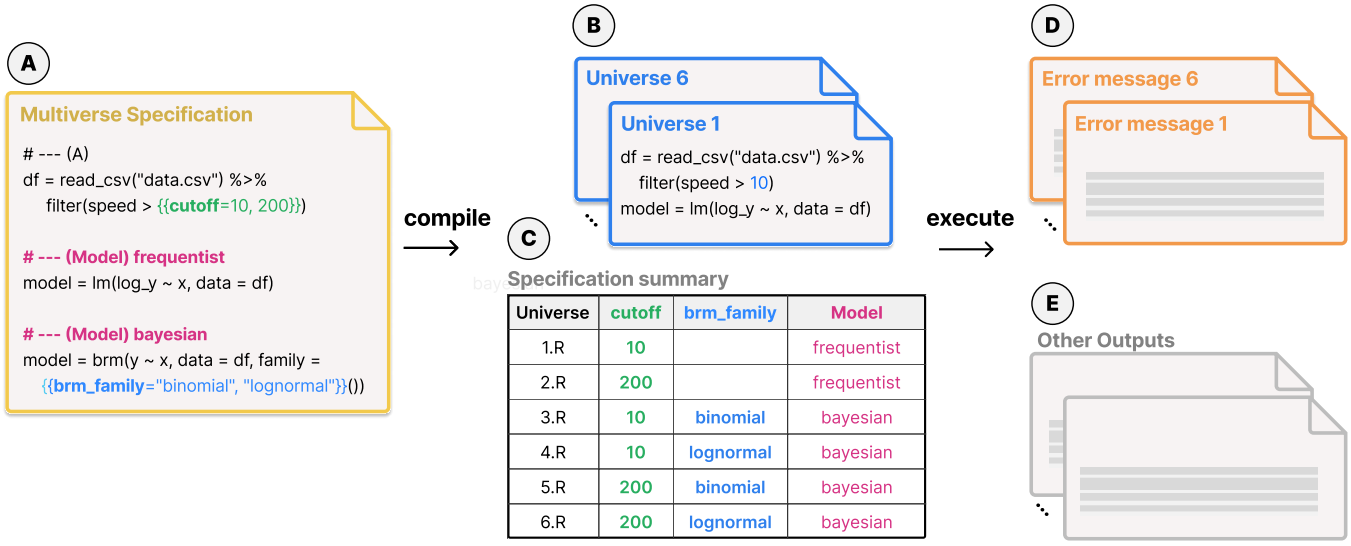

Figure 2: Multiverse Authoring Process An analyst starts out by writing a multiverse specification (A). Afterwards, the analyst compiles the multiverse specification into individual universes (B), which are enumerated in a specification summary (C). The specification summary indicates what the decisions are for a given universe. Next, when the universes are executed, each universe generates an error message (D) and other outputs (E) such as a model fit summary, model predictions etc. While this example shows Boba's domain specific language, other tools follow a similar process.

of existing tools to study and alleviate the challenges encountered during multiverse debugging. In this paper, we focus on studying debugging workflows with Boba, as it is widely researched in the research community [13, 23, 41, 50, 54]. One advantage of Boba is that it is programming language agnostic, allowing multiverse analysis to reach a greater audience.

2.4 Debugging is a Challenge in Multiverse Authoring

Based on prior work [20, 37, 38, 48], our experiences, and initial correspondences with multiverse practitioners and multiverse tool developers (see Appendix A), we hypothesize an initial debugging workflow (Figure 3). The workflow model is a first attempt to understand debugging in multiverse analysis and contrasts with the multiverse authoring workflow that is currently supported through existing tools (Figure 3A). After specifying and compiling a multiverse specification, the analyst executes the universes which produce error messages (Figure 3D1). Next, the analyst tries to diagnose the cause of the bug, which leads them to a single buggy universe to target (Figure 3D2). This step often involves examining multiple universes that share an error message. Once the analyst is working with an individual universe, they address the bug and make edits along the way, the same as they would when debugging a single script (Figure 3D3). After, they abstract and propagate the specific changes in the universe back to the higher-level multiverse specification (Figure 3D4). Finally, after the edits are propagated to form a new multiverse specification, it is compiled (Figure 3D5). This iterative debugging cycle typically repeats multiple times until all bugs are addressed.

This workflow model suggests the following three challenges to debugging a multiverse analysis.

Challenge 1 - Detecting bugs is slow. During the execution of the multiverse (Figure 3D1), the order of execution of the universes is arbitrary. Therefore, to discover a bug that occurs in a select few universes, hundreds or thousands of universes may need to be executed before the buggy universe is encountered. Even with running universes in parallel, this process can be time-consuming and drastically slows down the debugging cycle.

Challenge 2 - Sifting through error messages and multiverse artifacts to diagnose a bug is difficult at scale. In the process of diagnosing an error from running the multiverse (Figure 3D2), an analyst potentially needs to navigate through many error messages, many universes, and the specification summary and relate these sources to understand the shared decision options of an error. It is infeasible for an analyst to inspect hundreds of files (or a single file that combines these) and looking at a significantly smaller subset may not fully isolate the shared decision options of an error and divert focus from the true source of a bug. We note that a multiverse that does not lead to any error messages is not necessarily bug-free. For example, a poorly specified model formula may not be statistically sound but may not raise any error messages. However, many bugs exhibit themselves as error messages and that is the primary way analysts debug in our experience.

Challenge 3 - Abstracting and propagating universe edits to the multiverse increases cognitive load. In the procedure to abstract and propagate universe edits to the multiverse specification (Figure 3D4), the analyst needs to remember all their edits and locate where to place them in the multiverse specification.

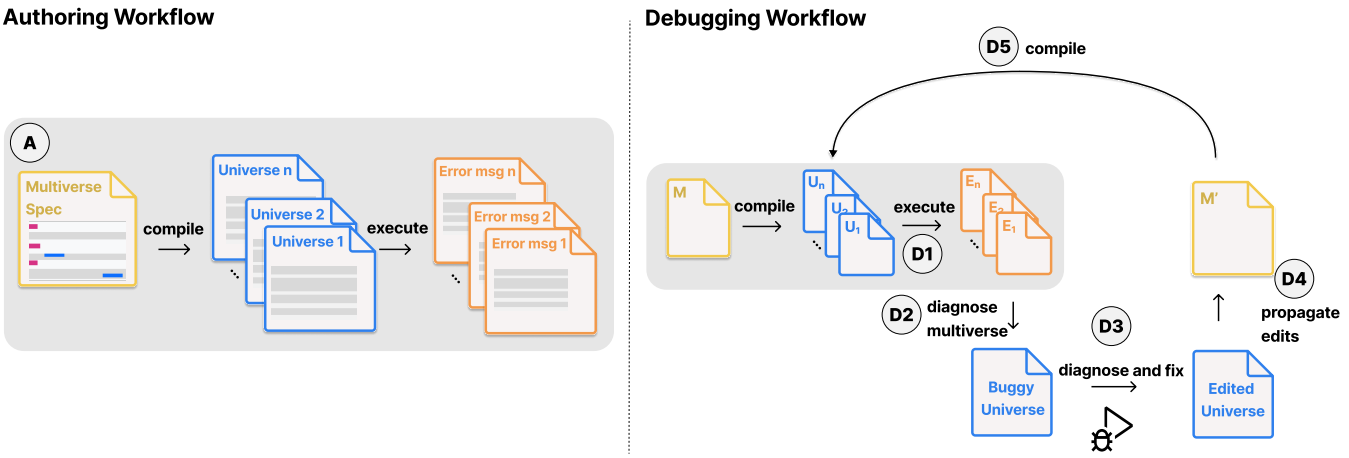

Figure 3: Authoring vs Debugging Workflow. Existing tools focus on the authoring process in a multiverse workflow (A) comprising of a multiverse specification, compiling the specification to universes, and executing the universes. However, there is an entire debugging workflow pertinent to the multiverse paradigm that is not well understood, presents challenges, and lack support from existing tools. We hypothesize the following debugging workflow. First, an analyst executes universes which can generate error messages (D1). Here, detecting errors quickly is challenging because executing all universes is time consuming and a universe containing error prone code may not be executed until hundreds or thousands of others have been executed already. Next, the analyst diagnoses what decision or set of decisions caused the errors (D2) which guides them to focus on one buggy universe. This step is challenging because an analyst needs to synthesize information from a myriad of sources (i.e., the multiverse specification, universes, error messages, and the specification summary) which only gets worse as the multiverse scales. Now, debugging at the universe level, the analyst diagnoses and fixes their bug in the typical single script debugging paradigm and is free to use debugging tools they are most comfortable with (D3). Once the fixes are made at the universe level, the analyst then propagates the edits made to the universe back to the multiverse specification (D4). This step contains the challenge of remembering changes in the universe and where those changes propagate to in the overall multiverse specification. Finally, the analyst compiles the new specification (D5) and the cycle repeats. The gray area highlights shared workflow steps.

In complex multiverse specifications and universe edits that involve many changes, propagating universe edits induces additional cognitive demands, especially when the analyst must keep track of the associated decision options underlying the code they are propagating.

3 PROTOTYPE: MULTIVERSE DEBUGGER

To better understand multiverse debugging workflows and how to support them, we set out to build a prototype tool, Multiverse Debugger, to use as a probe in our subsequent lab study. We identify three design goals to support analysts in the multiverse debugging workflow (Figure 3). The goals correspond to addressing the three challenges identified in Section 2.4.

(1) *DG 1 - Reduce the time between executing universes and encountering error messages.* After compiling a multiverse specification, a tool should enable analysts to quickly observe error messages before committing to running the full multiverse. This is in the spirit of unit testing in which different components of the multiverse are rapidly tested before running the entire system. Quickly identifying error messages before executing an entire multiverse may reduce time spent authoring (buggy) multiverse analyses.
(2) *DG 2 - Give an overview of error messages and how they relate to specific decision options.* Running thousands of universes can lead to thousands of individual error messages. In addition, error messages may arise due to a combination of decision options, which the analyst did not test when writing the multiverse specification. Therefore, diagnosing the severity and frequency of an error message helps to identify which parts of the code may need to be updated (including adding or removing dependencies between decision options in the multiverse specification). To identify common bugs and distinguish among different kinds of bugs, summarizing the frequency of error messages and connecting them to specific decisions and decision options are likely to help analysts.
(3) *DG 3 - Support the abstraction and propagation of single universe bug fixes to a multiverse specification.* The context of a multiverse analysis adds new complexity to fixing bugs. An analyst may elect to debug error messages in a specific universe as opposed to the higher-level multiverse specification. This enables the analyst to take advantage of already familiar and idiosyncratic ways of debugging specific universe error messages. In the analyst's process of debugging a single universe, they can leverage an entire ecosystem of single script debugging tools that they may already be familiar with. Therefore, making the process of propagating

changes to individual universes to the higher-level multiverse specification easier, empowers the preferred single universe debugging workflow.

Based on these three design goals, we implement Multiverse Debugger with three core features: decision cover, error message aggregation, and universe-to-multiverse diff. The features of Multiverse Debugger are designed to be used after compiling a written multiverse specification. This prototype extends the Boba multiverse library [38] and each feature is exposed through the Boba command line interface.

While we implemented Multiverse Debugger on top of Boba, the challenges and design goals would largely exist for other multiverse authoring tools as well. Boba makes the decision to represent universes and error messages as individual files. While other tools may make different design decisions such as consolidating all these into a single file or object, this would still result in similar challenges of slow detection of bugs (Challenge 1) and difficulty of diagnosing error messages from a large number of universes (Challenge 2). These challenges are ubiquitous because of the combinatorial explosion of universes which is inherent in multiverse analysis' definition to run individual analyses corresponding to all combinations of decisions. Therefore, these challenges which motivate DG 1 and DG 2 persist no matter the choice to represent universes as individual files or some other format. Challenge 3 and DG 3, meanwhile, are more specific to a universe level workflow which is enabled by tools like Boba in which the universe is represented as a single file. However, the choice of whether a tool enables a universe level workflow or multiverse level workflow (in which individual universes are not easily editable) comes with its own trade-offs which we further describe in Section 6.3.

Both the error message aggregation and the universe-to-multiverse diff interfaces are implemented as web applications in Python. The frontend uses HTML, CSS and Bootstrap [43], and the backend uses Flask [47]. The universe-to-multiverse diff interface also uses the Monaco Editor library [39].

3.1 Accelerating Bug Discovery Through Minimum Cover Approximation

A key problem in executing universes with existing tools is the latency between executing universes and encountering error messages. Analysts may not encounter a universe that contains a specific decision option until hundreds or thousands of universes have already been run. decision cover can reduce the latency in detecting a bug (DG 1) by helping the analyst quickly identify all error messages corresponding to code in a specific decision option while running a much smaller subset of universes. In seven multiverses we tested, decision cover reduced the number of universes to run by over 98%. After the analyst runs decision cover (`boba run --cover`), decision cover calculates the reduced set of universes, executes them, and surfaces the error message aggregation interface (Section 3.2) to summarize the error messages encountered in the executed universes. The analyst can interact with this interface to promptly see the set of error messages caused by a bug in any decision option.

decision cover calculates an approximation to the minimal set of universes to run such that all decision options in the multiverse are "covered". The problem of finding the minimal set of universes reduces to the classic set cover problem [31] which is known to be NP-hard [1]. To encourage trying different universes during each decision cover run, we employ a heuristic approximation based on random sampling that is highly effective in practice. We describe the decision cover algorithm in detail in Appendix B.

Making sure each decision option is encountered corresponds to ensuring "condition coverage" in traditional software testing [12]. However, decision cover does not ensure "multiple-condition coverage" [12] which would require running all combinations of decision options (essentially the entire multiverse) and leads to the combinatorial explosion of execution time. Nevertheless, error messages raised by "multiple-condition coverage" but not "condition coverage" are rare and become more obvious after the errors decision cover raises are addressed.

3.2 Diagnosing Bugs via Error Message Aggregation

A core challenge in diagnosing a bug from error messages is that it is difficult to sift through the myriad of information sources (i.e., error messages, universe files, and the specification summary) to diagnose a bug to a set of decision options. Therefore, we design error message aggregation to aggregate this information automatically and give analysts an overview of error messages and how they relate to specific decision options (DG 2). error message aggregation supports two interactions: identifying groups of error messages and the scale of an error, and understanding the decisions that may cause an error.

When an analyst runs the error message aggregation command (`boba --error`), the program ingests error messages from executed universes and categorizes errors based on string similarity (to handle slight line number or other changes in the error traceback). String similarity is calculated using the `string grouper` Python library [11] and is based on the cosine similarity of vectors of TF-IDF values in which the terms are N-Grams. Afterwards, the information is presented in an interactive interface that includes the aforementioned interactions (Figure 4).

3.2.1 Identifying error groups and the scale of errors. The analyst can quickly identify the number of universes affected by each error in a summary panel on the left-hand side (see Figure 4A1-3). Each error group has a preview of the error text (Figure 4A2) and a badge indicating the number of universes affected (Figure 4A3). The panel also displays a progress bar indicating the progress of universes run so far and updates when the user refreshes the page (Figure 4A1). The summary panel gives the user a sense of how many errors occur relative to what universes have already been executed. In addition, the summary view of error groups helps the analyst assess a bug's frequency across the multiverse and subsequently prioritize errors.

3.2.2 Understanding the decisions that may cause an error. Once an analyst has selected an error to investigate from the summary panel, they can focus on the shared decision options that potentially isolate an error group via the center panel (Figure 4B1-3).

The center panel shows a traceback of the error message as well as the shared decisions options of all the universes that caused that

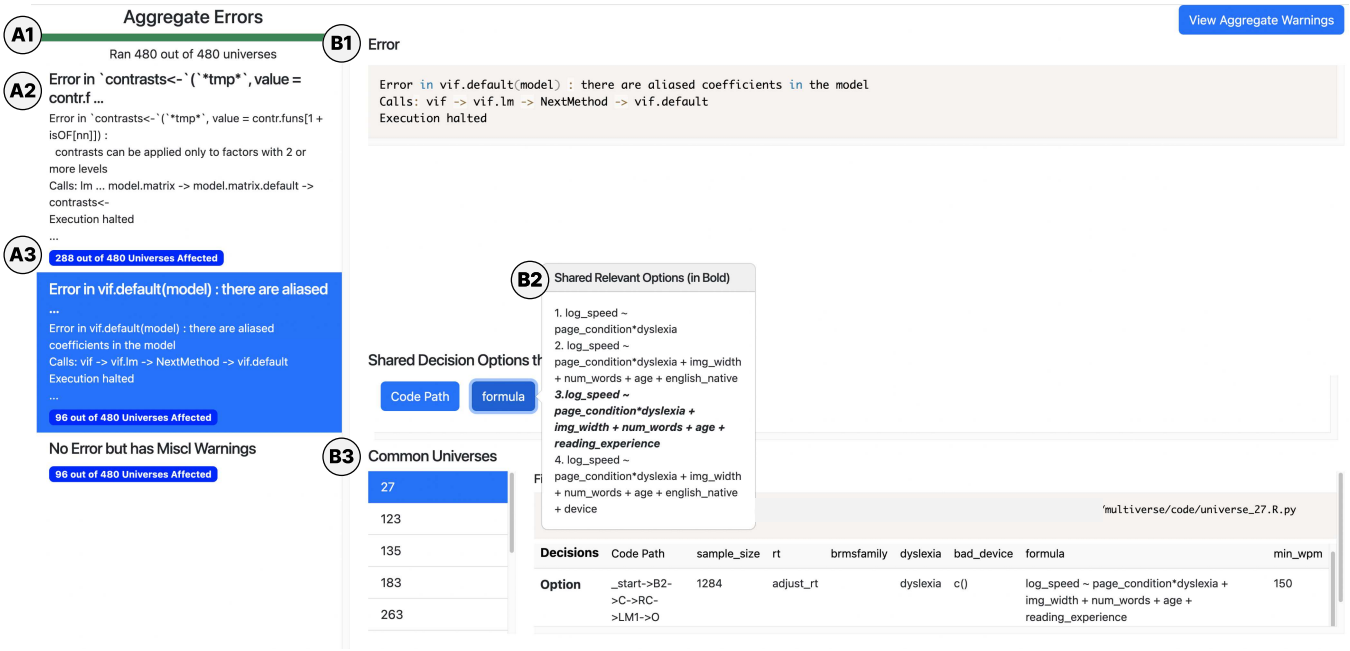

Figure 4: Error Message Aggregation Interface. The left side panel (A) contains a progress bar (A1) and unique groups of error messages in the universes ran so far. Each group contains a preview of the error message (A2), and the number of universes affected in each error aggregation (A3). The left side panel selects the error message to look into in the main panel (B). The main panel comprises of the full error message (B1), the decision options that are shared in the error aggregation (B2), and sample universes that contain the error message (B3). Without such a tool, analysts would have to manually inspect error messages (potentially hundreds or thousands of error messages) while cross-referencing universe entries in the specification summary. Not only is this action tedious but it is prone to missteps leading to poorly understood bugs. ERROR MESSAGE AGGREGATION seeks to address this challenge by automatically surfacing the information of all unique error messages and shared decisions in a grouped error.

error (Figure 4B2). Each decision that may cause the error is shown as a button to the analyst in which they can then click to see the shared decision options of universes that have this error message. Decisions that are most likely irrelevant to the error are removed to shift focus to the potential buggy decisions.

To determine whether a decision is irrelevant the following heuristic is used. If the error involves all options of a decision, then it is unlikely that anything in that decision is causing the error. If the error involves not all options in a decision, then there is a possibility that something specific to that option could affect the error. It must be noted, however, that the existing heuristic may not work if each option has an error that is identical. However, this scenario may be unlikely and it was never encountered throughout our entire study.

With a better understanding of the severity and decision options associated, the analyst can focus on a specific universe that has the selected error message to fix the specific bug. With an understanding of shared decision options in an error, the analyst may be able to better isolate where the error occurs and start with a more focused understanding of how the error may have occurred. Moreover, having a grasp on the isolated decision option that may cause an error provides a more semantically meaningful error message than a single script bug. With the additional information of an error message, the analyst can look for common universes that share the error at the bottom of the main panel (Figure 4B3) and begin focusing on one universe. Overall, the emphasis on outlining shared decisions across an error message can potentially help the analyst focus on a specific universe and the code most likely to cause the error.

3.3 Propagating Universe Edits with Universe-to-Multiverse-Specification Diffs

After making changes to a universe during debugging, the analyst may experience difficulty remembering all their universe edits and locating where to place edits in the multiverse specification. We design UNIVERSE-TO-MULTIVERSE DIFF to support abstracting and propagating edits in the universe to the multiverse specification (DG 3).

UNIVERSE-TO-MULTIVERSE DIFF propagates these edits automatically and presents the changes to the multiverse specification as suggestions. After an analyst finishes making edits to a universe, they can run `boba diff` to load an interface that communicates the suggested changes (Figure 5). The analyst can then refine these changes further if necessary.

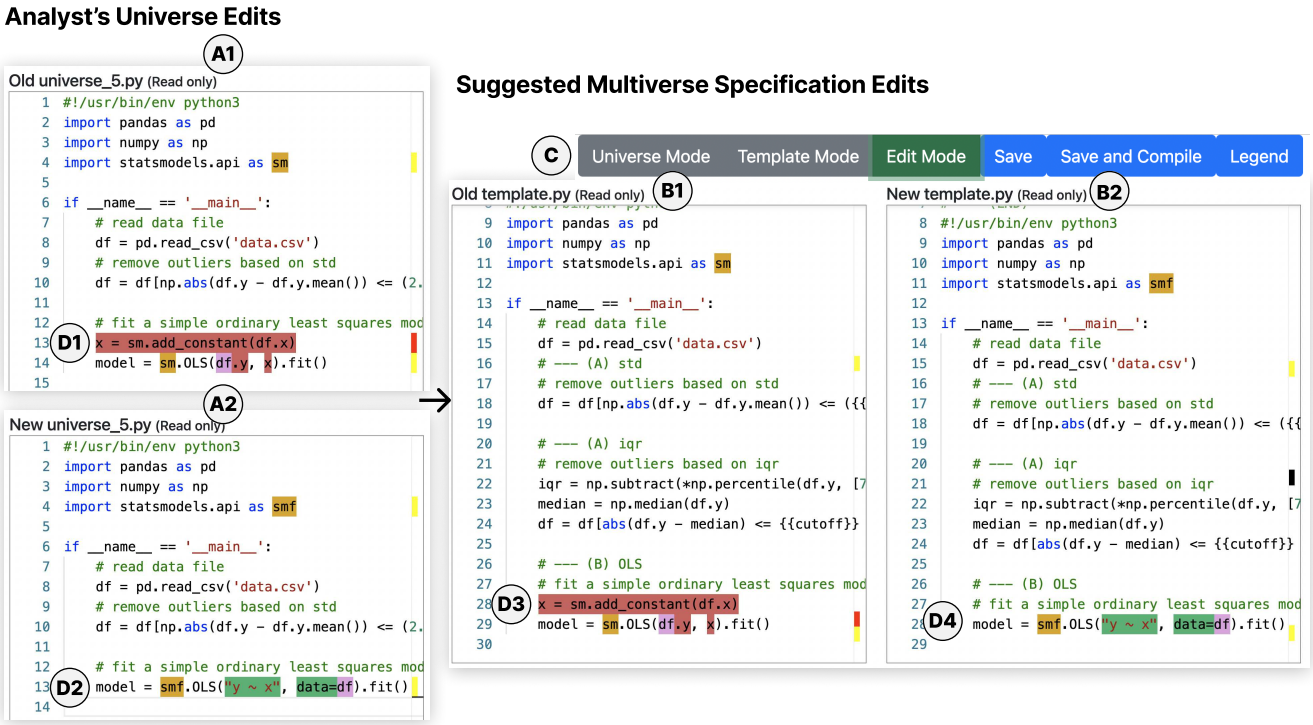

Figure 5: UNIVERSE-TO-MULTIVERSE DIFF Interface. An analyst makes changes to their universe to use the statsmodels formula API and runs `boba diff` to abstract and propagate their fixes back to the multiverse specification. This loads the various visual components in this figure. The analyst can view their edit changes via marked up code panels that show the code differences between the old unedited universe (A1) and the new edited universe (A2). Red highlighted code indicates delete edits while green highlighted code indicates insert edits. Yellow highlights show update edits and pink highlights show move edits. The analyst can then navigate via the navigation buttons (C) to view the suggested edits to the new multiverse specification (B2), the contents of which is generated from their universe edits. The interface shows these suggestions by highlighting the edits between the unedited multiverse specification (B1) and the new suggested multiverse specification (B2). Highlights in the old universe matches with those in the old multiverse specification (e.g., D1 and D3). Likewise, highlights in the new edited universe matches with those in the suggested multiverse specification (e.g., D2 and D4). Analysts can make any additional edits to the suggestions in another editor (not shown) before saving the new multiverse specification to disk. Without UNIVERSE-TO-MULTIVERSE DIFF, analysts would need to remember all their edits in a universe and how those edits propagate to the multiverse specification. UNIVERSE-TO-MULTIVERSE DIFF makes this process easier by automatically suggesting the necessary propagation of edits.

UNIVERSE-TO-MULTIVERSE DIFF's interface has three modes. There is a *universe* mode for viewing changes in the universe and a *template* mode for viewing suggested changes in the multiverse specification. The changes are shown as two-panel diffs. Additionally, there is an *edit* mode to make final edits (if necessary) to the suggested changes. The analyst navigates between modes with buttons in the top right (Figure 5C). The analyst may view the *universe* mode to best understand the universe-level changes they made, then proceed to the *template* mode to see how these changes are propagated to the multiverse specification, before finally entering the *edit* mode to finalize suggestions.

In the *universe* mode, the analyst starts with a better grasp of all the edits they made in the universe through viewing a code diff of their universe. The analyst can compare a panel containing highlighted code of the unedited universe (Figure 5A1) with a panel containing highlighted code of the edited universe (Figure 5A2). Highlights to the code show where insertion (green), deletion (red), move (pink), and update (yellow) edits are made (e.g., Figure 5D1-2).

In the *template* mode, the analyst can view how their changes in the universe are suggested in the multiverse specification. The analyst can compare a panel containing highlighted code of the old multiverse specification (Figure 5B1) with a panel containing highlighted code of the suggested new multiverse specification (Figure 5B2). The highlights for the old multiverse specification are propagated from the unedited universe (e.g., Figure 5D1 to D3). Analogously, the code and highlights for the new multiverse specification are propagated from the edited universe (e.g., Figure 5D2 to D4).

Finally, in the *edit* mode, the analyst can interact with a writable editor that contains the contents of the suggested multiverse specification. We implement a separate mode for editing to encourage a workflow in which the analyst is aware of their changes to the

universe and how those changes affect the multiverse specification. To support this further, we include a button for saving the new multiverse specification to disk (based on contents in the editor panel) and a button for saving and compiling only in the *edit* mode.

Beyond navigating these modes, the analyst can navigate between panels via the highlighted code edits. Highlighted code edits that correspond to the same code between panels are linked. For example, move edits in the old and new universe specifications are linked. When a highlighted edit is double-clicked, the linked edit in another panel will appear at the center of editor.

To implement Multiverse Debugger, because we need to propagate changes in the universe to specific decision options in the multiverse specification, we must identify decision options in the edited universe. To achieve this, Multiverse Debugger compares abstract syntax trees (ASTs) and lines of code of the edited and buggy universe. To compare ASTs, we adapt the gumtree algorithm, a popular source code differencing algorithm based on matching ASTs [21][2]. To compare lines, we use Python's difflib [2] library's mdiff function. Details of the universe-to-multiverse diff algorithm are in Appendix C.

4 LAB STUDY: RESEARCH QUESTIONS AND METHODS

Using our prototype Multiverse Debugger, we conduct a lab study to more specifically understand multiverse debugging workflows. Our primary goal was not to evaluate Multiverse Debugger but rather to create a potential improvement to debugging multiverse analysis in a tangible tool such that analysts could more concretely raise issues, benefits, and design guidelines that are tractable for future tool builders. Additionally, we wanted to allow analysts to explore alternative workflows and elicit responses regarding how features in Multiverse Debugger could enable or affect such a workflow.

Three research questions guide our study design and analysis.

- **RQ1 - Challenges:** What challenges do analysts need to overcome when debugging multiverse analyses? Specifically, do analysts really face the challenges we hypothesized based on prior work, our experiences, and initial correspondences with mutliverse practitioners and tool developers? What additional challenges do they face?
- **RQ2 - Workflows:** What workflows do analysts gravitate towards?
- **RQ3 - Tool:** To what extent do features like those in Multiverse Debugger address debugging challenges? How does Multiverse Debugger affect analysts' workflows?

The first two research questions are more open-ended and exploratory whereas the last research question assesses the benefits of Multiverse Debugger's design interventions and opportunities for further improvement.

Participants. Given that the population of multiverse analysis authors is relatively small, we focused on recruiting analysts who were interested in learning about multiverse analysis and represented potential adopters of multiverse analysis. We contacted data analysts through analysis-related mailing lists at our institution. In the initial interest form, we asked analysts to self-rate their statistical background on a 5-point scale (higher being more familiar). In this scale 4 described analysts who have taken graduate-level courses related to statistical analysis, and 5 described analysts having multiple years of experience with real-word projects involving statistical and data analysis. We also asked analysts to rate their familiarity with R or Python on a 5-point scale with 1 "being equivalent to have taken an introductory course" and 5 being having "5+ years of industry experience". From the interest forms, we further selected participants with strong backgrounds in statistical analysis (self-rated 4s and 5s) and comfort with Python or R. A total of 13 analysts participated and their background is summarized in Table 1.

Procedure. The study was conducted in a lab using a designated MacBook Pro computer on a 27-inch monitor. We allowed participants to use the programming language (i.e., R or Python) of their choice and installed what they needed. Analysts primarily used R Studio or Visual Studio Code for their integrated development environment. Before inviting analysts into the lab, we ensured they were familiar with our setup. We wanted to create a debugging environment that was as close to their own experiences.

For materials, we gathered two real-world multiverses from real-world analyses [28, 36] and we created buggy R and Python versions. To introduce realistic bugs, we searched Stack Overflow [6] with relevant keywords and online statistics blogs with consolidated lists of errors [14, 15] to find common bugs encountered during typical statistical analyses. We make the buggy multiverses publicly available and explain the multiverse preparation process in more detail in Appendix D.

The study was structured into an initial tutorial phase, followed by two separate debugging task phases that differ in whether the analyst was introduced to Multiverse Debugger and was able to use it. We followed this protocol to observe analyst workflows both prior to introducing Multiverse Debugger and afterwards.

At the beginning of the study (tutorial phase), we guided analysts through a tutorial that introduced the multiverse analysis paradigm and how to use Boba. The tutorial explained how to specify decisions and decision options using Boba syntax. To ensure analysts understood the concepts behind multiverse analysis and felt comfortable using Boba, we asked analysts to update a Boba multiverse specification to add another decision option. We also walked analysts through Boba's compile and execute commands.

Next, we asked analysts to debug a realistic multiverse analysis with bugs (Phase 1). In this first part of the study, analysts had 25 to 35 minutes to address as many bugs as they could with the existing Boba tools. Analysts debugged the first multiverse on their own and then completed a survey about their experience.

Afterwards, in Phase 2, the first author gave an overview of Multiverse Debugger and how to invoke each command and use the interfaces. Analysts were explicitly told they were free to debug however they wanted. Subsequently, depending on their progress in the first portion (i.e., whether they solved the bugs in the first multiverse), analysts were asked to either continue debugging the

[2]We release a Python re-implementation of the gumtree algorithm with adaptations for universe-to-multiverse diff available at https://github.com/behavioral-data/multiverse-tooling/tree/main/src/gumtree

Table 1: Participant information. Proficiency was self-rated on a 5-point scale with 5 being the highest.

ID	Gender	Occupation/Background	Programming Lang.	Lang. Proficiency	Statistics Proficiency
A01	Female	Researcher in Data Science	Python	4	4
A02	Male	Masters Student in Data Science	Python	5	5
A03	Female	Masters Student in Industrial Engineering	Python	4	4
A04	Female	PhD Student in Information Science	Python	5	5
A05	Male	PhD Student in Public Policy	R	3	5
A06	Female	PhD Student in Quantitative Ecology	R	3	4
A07	Female	PhD Student in Psychology	R	5	5
A08	Female	Data Analyst in Medicine	R	5	5
A09	Female	Data Scientist	R	3	4
A10	Female	PhD Student in Applied Mathematics	Python	4	4
A11	Male	Data Scientist	R	5	5
A12	Male	PhD Student in Biostatistics	Python	4	5
A13	Male	Professor in Biostatistics	R	5	5

first multiverse or debug a second multiverse. More time was spent in the first portion such that analysts can become familiar with Boba and the multiverse paradigm. This also ensured analysts had time to experience challenges specific to multiverse debugging. Finally, analysts completed a survey about their experience using Multiverse Debugger.

We encouraged analysts to talk about their process when they could. If not, they were regularly prompted to speak about their process and describe their thinking. After each debugging session, we also asked open-ended questions with the objective to understand the processes and challenges of multiverse debugging. We gave analysts minimal help beyond pointing out the tools and resources they have available (i.e., the IDE debugging tools, the Internet, and Boba documentation). If analysts were stuck diagnosing and fixing the bug at the single script level (Figure 3D3) for longer than 15 minutes, we guided analysts by pointing out what the bug is to allow insights along all parts of the workflow.

The study lasted approximately 2 hours. Analysts received a $50 Amazon gift card as compensation for their time. This study was determined exempt through the IRB at our institution. We include all lab study materials in our supplemental material.

Qualitative Coding Process. With the exception of one participant (A10) who did not consent to be recorded, we recorded participants' audio and screens. In addition to writing notes of analysts' behaviors while conducting the study, the first author viewed the recordings and transcribed all episodes of interest to the debugging process. To understand common themes that emerged, we used iterative open coding. The themes we observed highlighted analysts' challenges in debugging multiverse analysis, workflows that analysts gravitated towards, and finally how Multiverse Debugger addressed these challenges.

5 LAB STUDY: RESULTS

Our lab study identifies four challenges to debugging multiverse analyses and two approaches analysts take to debug. We also observe how Multiverse Debugger affects analysts' workflows and enables them to overcome the debugging challenges. These findings inform our updated model of the multiverse debugging workflow, as summarized in Figure 6. The key differences between the updated model and the initial hypothesized model (Figure 3) are the expanded steps in diagnosing a multiverse (Figure 6D2-4) (Section 5.2.1), the additional path to editing a multiverse specification directly (Figure 6D7-8) (Section 5.2.2), and the additional choice of selectively executing a semantically meaningful subset of universes (Figure 6D1) (Section 5.3.1).

5.1 What challenges do analysts need to overcome when debugging multiverse analyses?

We found that analysts experienced difficulty with two of the three hypothesized challenges: detecting bugs quickly (Challenge 1 in Section 2.4) and finding the root causes of bugs (Challenge 2 in Section 2.4). In order to find the cause of bugs, we found that analysts needed to group unique errors and identify shared decisions of an error. Maintaining a mental model of the multiverse was also challenging for analysts.

5.1.1 Minimize latency between executing a multiverse and detecting errors. Running the entire multiverse took a non-trivial duration of time, making it difficult for analysts to receive quick feedback on what errors existed. To minimize this latency, some analysts picked an arbitrary number of universes to run [A04, A07, A12]. For instance, prior to using Multiverse Debugger, A04 was reluctant to rerun the multiverse after fixing a bug. Instead, A04 spot-checked three universes. Similarly, A01 reduced the size of the multiverse by commenting out decision options that were irrelevant to the bug she was addressing.

5.1.2 Group unique errors and find the number of universes affected. In the existing workflow without Multiverse Debugger, analysts have no grasp on what the unique errors are and the number of universes that are affected. Thus, analysts do not know what a bug fix would even solve and can be left feeling overwhelmed. A05 captures this perfectly: "*Seeing that there are 1500 errors but not having any idea how many were unique makes the process feel overwhelming.*"

Multiple analysts while debugging without Multiverse Debugger, and prior to learning about the tool's existence, asked if there

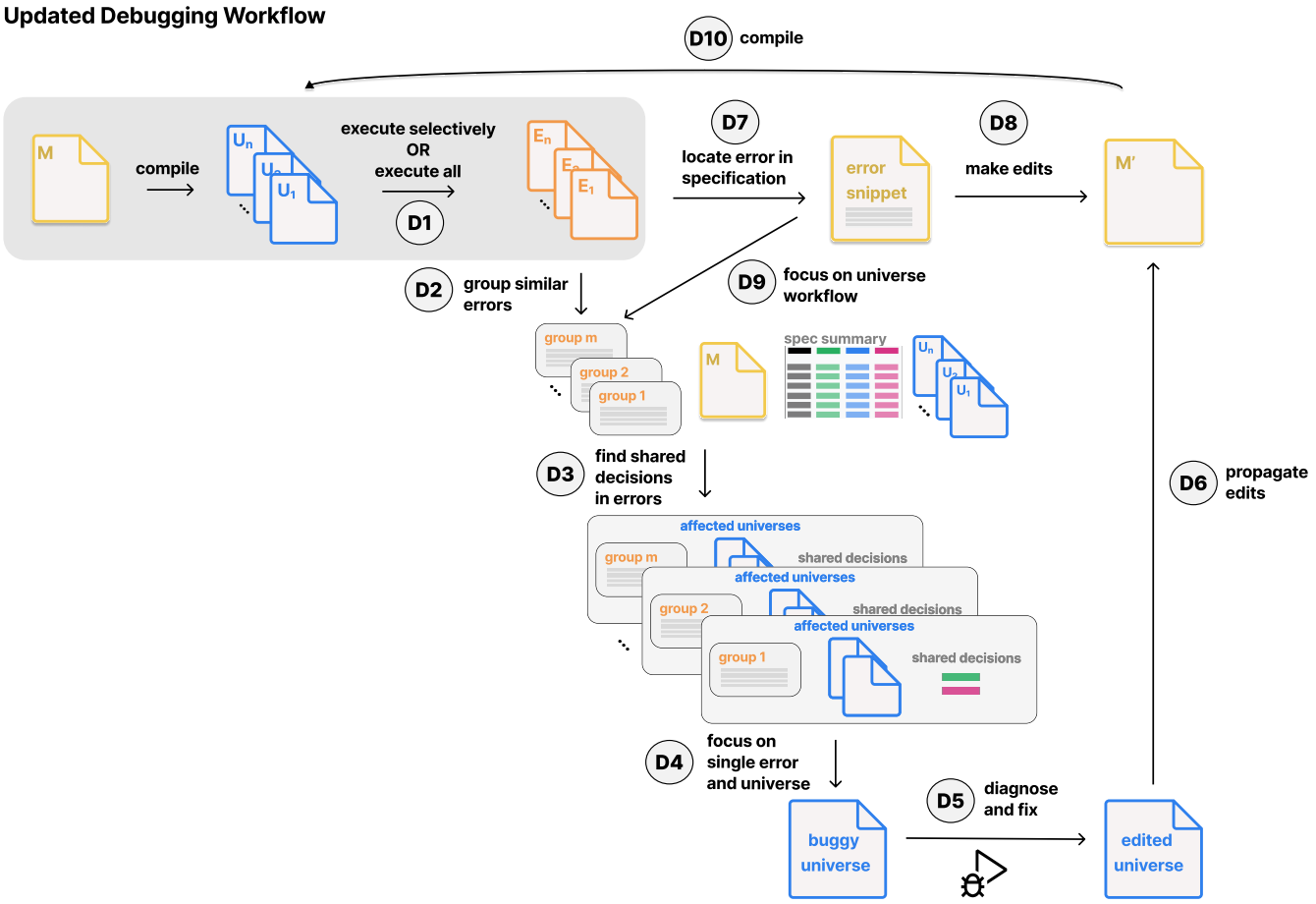

Figure 6: Updated Model of Debugging Workflow. The updated workflow model shows a revised and extended version of the multiverse debugging workflow, enabled through our lab study and Multiverse Debugger. Compared to the hypothesized workflow model (Figure 3), the model derived from the lab study has multiple refinements . First, beyond executing all universes (D1 in Figure 3), the execution step (D1) now captures analysts' propensity to run a select few universes via decision cover and interest in running their own subset based on specific decision options. Next, our initial understanding of diagnosing the multiverse (D2 in Figure 3) is expanded to include steps of grouping similar errors (D2), using this grouping along with the specification summary and associated universe information to find shared decisions in error groups (D3), before prioritizing an error and focusing on a single universe (D4). These steps also surfaced an additional challenge of analysts' trouble in understanding the composition of the multiverse. Lastly, to capture analysts' tendency to make fixes directly in the multiverse specification, there is now an additional path in which after observing error messages, an analyst locates the error in the multiverse specification (D7) and then makes the bug patches there directly (D8). Analysts can also go back to the universe workflow (D9) to leverage their comfort with single universe debugging tools.

was a way to see the errors grouped together or mentioned lack of grouping as a challenge [A02, A03, A05, A08, A11, A12].

5.1.3 Identify shared decisions of an error. Once analysts found an error common across multiple universes, they tried to isolate the decision choices responsible for producing the error (Figure 6D3). To do so without Multiverse Debugger, analysts cross-referenced the error messages with the specification summary [A02, A05, A06, A11, A12, A13]. Most participants gave up because the specification summary was "hard to read", especially when it contained hundreds of entries with no semantically meaningful structure.

5.1.4 Understand the composition of the multiverse. Understanding the composition of the multiverse means to "understand the components and processes that define and make up this multiverse" [25]. For analysts, the composition was not obvious from the information available. To aid in debugging, analysts referenced the multiverse specification file, the specification summary, and the universes to build up a mental map of the multiverse. For A01, this mental map was essential in her debugging process: "*Many of these different paths have co-dependencies so I'm not quite sure yet which one of these is truly the issue*"". To understand common errors in universes, analysts consulted error messages and the specification summary

to find a common error among several universes. To locate the potential source that caused the error and understand how a specific universe was generated, analysts looked at the universes, the multiverse specification, and the specification summary. Because the information conveying the composition of the multiverse was scattered, many analysts mentioned processing and navigating the disjointed information as a challenge [A01, A02, A05, A06, A08, A09, A11, A13]. From just these sources alone, analysts struggled to construct a mental model of how decision options were related and contributed to errors common across multiple universes [A01, A02, A05]. A05 stated how it was "*not naturally obvious that there are duplicates stemming from the exact same piece of code*".

As a result, analysts mentioned desiring features that can be broadly categorized into two groups: features that connect information sources and features that can help visualize the multiverse.

For connecting information, analysts desired a feature that enabled them to locate the code in the multiverse specification which ultimately resulted in an error [A03, A07, A08]. Similarly, others wanted an explicit mapping between code in the universe file and code in the multiverse specification [A02, A03]. For desired visualization features, A11, for example, mentioned wanting a tree structure (like in Figure 1) that summarizes the multiverse and associated artifacts: "*What if I had a tree diagram that I could select which universes does this error happen in that lights up the tree, and show me that they all have this condition.*"

5.2 What workflows do analysts gravitate towards?

5.2.1 Analysts address bugs in order of error messages but seek new ways to prioritize bugs. Without Multiverse Debugger, analysts often inspected the first error message and set out to fix it [A01, A02, A03, A04, A05, A06, A08, A09, A10, A11, A12, A13]. A01 found this approach comfortable and reasonable, saying "*I want to kind of fully tackle that one and then resolve it and then go on to the next one as opposed to having a higher-level plan.*" However, others wanted a more strategic way to prioritize bugs, which required a more holistic picture of bugs across multiple universes [A03, A09, A12, A11, A13]. A11 explained his debugging priority was to solve the error affecting many universes:

> "*I am more interested in spending my time addressing the bugs that occur in several universes versus the bugs in the first universe but I did not have a good sense for how to determine that, so I just went to the first error*"

To prioritize, analysts expressed interest in grouping errors together to see unique errors [A02, A03, A05, A08, A11, A12] (Figure 6D2). Once they have a sense of the unique errors, analysts wanted to see what was similar and different among universes that encountered the same error in order to isolate the shared buggy code [A02, A05, A06, A11, A12, A13] (Figure 6D3). Some analysts [A02, A10, A11] did so by comparing entries in the specification summary that corresponded to universes that had a common error. A02 went so far as to write a script that parsed the specification summary with error "lines" (i.e., error messages):

> "*What I was trying to do was to read which (error) lines contain the options and just parse those lines. I was going to write a small script to just parse the lines.*"

This idea matches our error message aggregation feature that they had not yet learned about.

5.2.2 Analysts adopt different strategies based on perceived bug severity. When analysts perceived an error to have an easy fix, they directly updated the multiverse specification file without consulting a specific universe script at all [A02, A03, A04, A07, A10, A13] (Figure 6D7-8). Analysts stayed in the multiverse specification file because they knew they had to update it eventually anyway. For example, A03 wanted to reduce effort: "*because the template is the place where we generate the whole universe so I think as long as the bug is fixed in the template, the universe will be free of bugs*". Meanwhile, A07 expressed she preferred staying in the multiverse specification because she observed a lot of shared code occurred early in the multiverse specification.

> "*I could see that the branching points weren't actually that many if you scroll down through the template file. I saw that there were only really the model points that were breaking routes. If I can get everything before those points to be okay, and then everything subsequently can be re-edited to the template.*"

When finding errors, analysts also simplified their diagnosing process to just locating the line referenced in the traceback in the multiverse specification (Figure 6D7). However, because the multiverse specification is not executable, not every bug could be understood and solved there.

For more involved errors requiring analysts to run large code snippets or inspect intermediate variable values, analysts defaulted to finding and debugging a specific universe. Of the 13 analysts, 12 (everyone except A07) attempted to fix a bug in a specific universe before making similar fixes to the multiverse specification file. Focusing on one universe at a time was more familiar to analysts who could rely on their idiosyncratic debugging approaches, such as using print statements [A02, A03, A04, A12], the interactive debugger [A02, A10], or the interactive console (i.e., the R console and the Python console) [A03, A05, A06, A08, A09, A11, A13]. Analysts stayed in the same universe until they fixed a specific bug [A01, A02, A03, A06, A10, A11, A12, A13] or ensured the universe was completely bug-free [A04, A05, A08, A09]. Once analysts were satisfied with their changes, they updated the multiverse specification file, re-compiled and re-started the debugging loop.

In some situations, analysts misjudged the complexity of the error and started with the multiverse specification but then went to a universe workflow (Figure 6D9) after realizing it would have been more effective [A01, A02, A05, A09, A11, A13]. In these cases, analysts wanted to fully leverage their single universe debugging workflows.

5.3 To what extent do features like those in Multiverse Debugger address debugging challenges? How does Multiverse Debugger affect analysts' workflows?

Analysts' debugging patterns, which were present without Multiverse Debugger but further supported by Multiverse Debugger, are described in our updated model of the debugging workflow (Figure 6). Analysts leveraged error message aggregation to

group similar errors (Figure 6D2), find shared decisions in an error (Figure 6D3), before then prioritizing an error and focusing on one universe (Figure 6D4). Moreover, analysts used DECISION COVER to detect errors faster (Figure 6D1) which inspired them to desire even greater control on what subsets of universes to run. However, analysts seldom used UNIVERSE-TO-MULTIVERSE DIFF and elected to propagate universe edits manually (Figure 6D6).

5.3.1 DECISION COVER reduces latency in detecting bugs and speeds up the development and debugging loop. Nearly all analysts found DECISION COVER feature helpful in expediting the incremental development and debugging loop [A01, A02, A03, A04, A06, A07, A08, A09, A10, A11, A12, A13]. Analysts found the DECISION COVER useful for finding the most common errors quickly and expressed interest in using it as the first step in debugging multiverse analyses in the future. For example, A07 expressed, "*I really like the ability use* `boba --cover` *which helped pinpoint the most common errors.*" Furthermore, for A04, the DECISION COVER enabled her to work directly in the multiverse specification: "*These tools drastically reduced the amount of feedback loop time. Instead of editing the individual universe files, I mainly worked from the template file.*"

Analysts expressed wanting greater control in specifying which subset of universes to execute [A01, A04, A07]. Furthermore, other analysts wished they could version their error messages to maintain the results and errors from a long multiverse run [A01, A12].

5.3.2 ERROR MESSAGE AGGREGATION helps analysts see unique errors and isolate potential causes to specific decision options. Analysts used ERROR MESSAGE AGGREGATION to identify (i) what the unique errors were and (ii) how many universes each error message affected. Knowing the unique errors helped analysts identify familiar error messages they could quickly address [A13] or prioritize error messages that affected the greatest number of universes [A01, A05, A07, A11]. For instance, A01's strategy was the former: "*After seeing the breakdown of the different errors, I would prioritize them and in my head, get a sense of if I fix this fundamental error, would it fix other errors.*"

We designed ERROR MESSAGE AGGREGATION anticipating the challenge of grouping similar errors and finding shared decisions in a common error. All 13 analysts liked ERROR MESSAGE AGGREGATION and said they would want to use it in their workflow. A05, who was frustrated by his initial lack of awareness of which bugs overlapped with each other, especially liked the ERROR MESSAGE AGGREGATION: "*The error aggregate is definitely the most useful because it allows for seeing not only the groups of errors but how many universes are affected.*"

A particularly illustrative example was A02. Prior to using MULTIVERSE DEBUGGER, A02 wrote a custom script to parse the error messages and the specification summary for 15 minutes before running out of time. When he started to use MULTIVERSE DEBUGGER, A02 found ERROR MESSAGE AGGREGATION especially useful: "*I really like that you could get a high-level overview of all the choices that are getting affected.*" Although analysts found ERROR MESSAGE AGGREGATION beneficial, they also recommended using visualizations or changing the button layout to make the interface more intuitive [A01, A03, A06, A12, A13].

5.3.3 UNIVERSE-TO-MULTIVERSE DIFF is not as necessary to abstract and propagate patches. Analysts found UNIVERSE-TO-MULTIVERSE DIFF the least useful. One analyst [A02] used the tool to mainly test the feature. As expected, when analysts stayed in the multiverse specification, UNIVERSE-TO-MULTIVERSE DIFF was unnecessary. When analysts dove into specific universes, analysts had mixed feelings about UNIVERSE-TO-MULTIVERSE DIFF. On one hand, A07, who in her own workflow uses `git` diffs only in the CLI, thought UNIVERSE-TO-MULTIVERSE DIFF would help people who more "visual." On the other hand, A12 thought UNIVERSE-TO-MULTIVERSE DIFF could be helpful if he spent more time in a universe and needed to remember more changes: "*Most of the cases right now you give me are simple but once the debug time is too long then you'll easily forget how you did the changes. That would be the most useful case.*"

6 DISCUSSION

In this work, we built a prototype tool and conducted a subsequent lab study to understand and address multiverse debugging challenges. From our lab study, which leveraged our tool as a design probe, we developed an updated model of multiverse debugging workflows (Figure 6). In this section, we synthesize the results from our lab study and share implications for improving multiverse analysis tools. We highlight four key design implications that would better support multiverse debugging, review the limitations of our work, and discuss future work.

6.1 Design Implications

6.1.1 Tools should reduce the latency in encountering multiverse errors. The long time to detect an error message (step D1 in Figure 6) was a challenge we hypothesized (Section 2.4) and later confirmed in our lab study (Section 5.1.1). In the lab study, we even found analysts trying their own ways to increase the speed of detecting error messages (i.e., commenting out code). We also found the DECISION COVER feature to be especially useful because it enabled this faster detection (Section 5.3.1). Future tools should consider features that reduce the latency to detect erroneous multiverse code whether that is through something like DECISION COVER or letting analysts run subsets of universes (something we discuss as another design implication in Section 6.1.4)

6.1.2 Tools should summarize unique errors and highlight shared decision options. The challenge of understanding what unique errors exist (step D2 in Figure 6) and what are common decision options (step D3 in Figure 6) was pervasive in the lab study (Section 5.1.2). As a result, MULTIVERSE DEBUGGER 's ERROR MESSAGE AGGREGATION feature which directly addresses this was appreciated by all analysts (Section 5.3.2). Multiverse debugging tools will benefit from some form of error message aggregation.

6.1.3 Tools should help analysts understand the composition of the multiverse. A key challenge that surfaced among analysts in the lab study was understanding the composition of the multiverse; that is, how the specification of decisions and options led to the generation of universes (Section 5.1.4). While we hypothesized the need to understand the multiverse would contribute to the cognitive load in propagating edits (Section 2.4), our lab study revealed this

understanding is critical much earlier in the debugging cycle (Figure 6D2-4) and less important when propagating edits (Figure 6D6) (Section 5.3.3). Specifically, in diagnosing an error message, analysts needed this comprehension to begin understanding what decision options may have caused an error or how code may be shared across certain universes as a result of the multiverse specification. Moreover, multiple analysts expressed connecting the multiverse structure (i.e., showing the structure relating universes, decisions, and decision options) to the multiverse specification code as something that would aid in their debugging process (Section 5.1.4).

Informed by our lab study, future tools that aid in understanding the composition of the multiverse should connect the multiverse structure with the multiverse specification. One opportunity to support understanding is through interactive visualizations that connect a visualization of the multiverse structure with a visual representation of the multiverse specification code. Such a visualization would also support analysts' iterative authoring process [32, 33], enabling analysts to understand how the composition of the multiverse changes over time as a result of code changes. Prior work has also highlighted the need for real-time and interactive visualization of the multiverse structure [48].

While researchers have started to develop multiverse-specific visualizations [25, 38], none have focused on interactions showing the multiverse structure and the specific code implementing them in the multiverse specification. Future work should explore how to best communicate the specified multiverse structure in relation to the specification code.

6.1.4 Support Analysts in Finding Relevant Universes and Decision Options in the Multiverse. Another common theme observed in the lab study was analysts' need to have control in finding subsets of universes or subsets of decision options. For example, to better isolate a potential cause for an error message, analysts expressed wanting to know what subset of universes to run that correspond to specific combinations of analysis decisions (Section 5.3.1). This is difficult because to find that subset, analysts currently need to either consult the specification summary and navigate through hundreds of entries or write custom functions to parse this information. On the other hand, Multiverse Debugger 's error message aggregation feature, which analysts ubiquitously found helpful (Section 5.3.2), is a realization of finding a subset of meaningful decision options from a subset of universes.

Therefore, core activities involved in multiverse debugging require finding a subset of universes based on specified decision options or finding a subset of decision options based on specifying a subset of universes. Tools that enable this process would improve analysts' capability to and speed in diagnosing error messages. As such, future tools should incorporate effective multiverse selection based on universe or decision option constraints.

6.2 Limitations

Multiverse Debugger focuses on extending Boba to understand multiverse debugging workflows. Therefore, its features are all command-line based. For analysts who are less comfortable with programming and more comfortable with workflows that involve graphical user interfaces (e.g., Stata [7], SPSS [5]), Multiverse Debugger may be difficult to use.

We note several limitations of our user study. First, the study had a small sample size and consisted of people new to multiverse analysis. As the number of people who perform multiverse analysis is small, we determined an in-person lab study was the best way to gather people, provide a tutorial on multiverse analysis and get them up to speed with existing tools. Results, therefore, might be different for multiverse experts. However, as multiverse analysis is a relatively new analysis paradigm, there are very few experts to date and an important focus lies on empowering a broad set of analysts to employ multiverse analyses. Multiverse analysis is targeted to those familiar with statistical practices who may want to adopt this paradigm (which is our lab study population) and it is through making the associated challenges easier (specification, analyzing results, and debugging) that this paradigm will receive greater adoption. Prior tools [24, 38, 48] improved workflows surrounding specification and analyzing results but that adoption is still limited in part due to debugging challenges that are not yet supported [48]. Understanding the debugging challenges of a potential adopter is one step toward this larger goal.

Additionally, in order to facilitate a lab study of reasonable duration, we chose to conduct a same-day in-person study of 2 hours and give analysts a largely pre-written multiverse. Future work should explore debugging processes based on a multiverse the participant is developing themselves as well as more complex multiverses. Finally, while the bugs introduced into the pre-written multiverses reflected common analysis errors, they may not be representative of those encountered in more complex or domain-specific analyses. We hypothesize that the overall workflow will likely be similar but analysts may want to focus even more on debugging individual universes. In addition, universe-to-multiverse diff may be more useful in these larger multiverses with more complex bug fixes.

6.3 Future Work

Towards enabling debugging for larger classes of bugs. Multiverse Debugger helps analysts author a multiverse that is free from execution errors. However, there could be bugs that do not lead to execution errors, including bugs around statistical analysis misspecification (e.g., a poorly specified model and model formula). These bugs may not raise error messages but threaten the statistical validity of the analysis. This type of bugs is not specific to multiverse analysis but relevant to all analysis paradigms. Recent tools have been developed to improve statistical validity in traditional analysis [26, 27] but more work is needed to help analysts detect such bugs. Another class of bugs is related to errors in multiverse specification. For example, an analyst may have intended to perform data filtering only for a subset of models but did not specify that constraint in the multiverse specification. While there would not be any execution errors, the universes affected may not reflect the intended analysis. Future work could explore how to detect and communicate these bugs to the analyst.

Exploring the trade-offs between universe level and multiverse level workflows. While most analysts favored debugging with a single universe, we discovered in our lab study some analysts tended to debug with the multiverse specification directly (Section 5.2.2). Analysts' tendency to focus on one level could also

be influenced by the tool they are working with. Boba [38] naturally encourages a universe level workflow as the universes are separated from the multiverse specification and are no different than traditional analysis scripts. This lets analysts use their favorite tools and familiar workflows. However, the separation has the drawback that the multiverse specification cannot be directly executed. `multiverse` [48], in contrast, encourages a multiverse level workflow and lets analysts run universes via library functions in the same file in which the multiverse is specified. However, placing everything in one file puts multiverse specification logic and analysis code all in a single file, which may even more difficult to debug. Future work should explore these trade-offs between executable higher-level multiverse specifications and the complexity of navigation and debugging.

7 CONCLUSION

This paper focuses on debugging as a key, under-scrutinized barrier to broader multiverse analysis adoption. To understand analysts' challenges and debugging workflows, we build a prototype debugging tool, Multiverse Debugger, and conduct a qualitative lab study using Multiverse Debugger as a probe. This work contributes the first user study to better understand, model, and support the unique challenges that multiverse analysis poses for debugging. In addition, we provide an open-source tool, Multiverse Debugger, that alleviates some of the observed challenges. We synthesize findings to develop a model of multiverse debugging workflows and associated challenges (Figure 6) and highlight design implications for future tools to support multiverse analysis debugging.

ACKNOWLEDGMENTS

We are grateful to our anonymous reviewers for their thoughtful comments. We would also like to thank the members of the UW Behavioral Data Science group and the UW PLSE group for their feedback on this work. This research was supported in part by NSF IIS-1901386, NSF CAREER IIS-2142794, NSF CNS-2025022, NIH R01MH125179, Bill & Melinda Gates Foundation (INV-004841), the Office of Naval Research (#N00014-21-1-2154), a Microsoft AI for Accessibility grant and a Garvey Institute Innovation grant.

REFERENCES

[1] 2006. Approximation Algorithms. In *Combinatorial Optimization: Theory and Algorithms*. Springer, Berlin, Heidelberg, 377–413. https://doi.org/10.1007/3-540-29297-7_16
[2] 2022. difflib — Helpers for computing deltas — Python 3.10.7 documentation. https://docs.python.org/3/library/difflib.html
[3] 2022. Introduction — statsmodels. https://www.statsmodels.org/
[4] 2022. pandas - Python Data Analysis Library. https://pandas.pydata.org/
[5] 2022. SPSS Software. https://www.ibm.com/analytics/spss-statistics-software
[6] 2022. Stack Overflow - Where Developers Learn, Share, and Build Careers. https://stackoverflow.com/
[7] 2022. Statistical software for data science | Stata. https://www.stata.com/
[8] Abdulaziz Alaboudi and Thomas D. LaToza. 2021. An Exploratory Study of Debugging Episodes. http://arxiv.org/abs/2105.02162 arXiv:2105.02162 [cs].
[9] Monya Baker. 2016. 1,500 scientists lift the lid on reproducibility. *Nature* 533, 7604 (May 2016), 452–454. https://doi.org/10.1038/533452a Number: 7604 Publisher: Nature Publishing Group.
[10] Moritz Beller, Niels Spruit, Diomidis Spinellis, and Andy Zaidman. 2018. On the dichotomy of debugging behavior among programmers. In *Proceedings of the 40th International Conference on Software Engineering*. ACM, Gothenburg Sweden, 572–583. https://doi.org/10.1145/3180155.3180175
[11] Chris van den Berg. 2020. String Grouper. https://github.com/Bergvca/string_grouper
[12] Robert V. Binder. 1999. *Testing Object-Oriented Systems: Models, Patterns, and Tools*. Addison-Wesley Longman Publishing Co., Inc., USA.
[13] Paul Alexander Bloom. 2022. *Into the Multiverse: Methods for Studying Developmental Neuroscience*. Ph.D. Columbia University, United States – New York. https://www.proquest.com/docview/2656195811/abstract/673E811D3C8B4B71PQ/1 ISBN: 9798426817869.
[14] Zack Bobbitt. 2022. Python Guides. https://www.statology.org/python-guides/
[15] Zack Bobbitt. 2022. R Guides. https://www.statology.org/r-guides/
[16] Richard Border, Emma C. Johnson, Luke M. Evans, Andrew Smolen, Noah Berley, Patrick F. Sullivan, and Matthew C. Keller. 2019. No Support for Historical Candidate Gene or Candidate Gene-by-Interaction Hypotheses for Major Depression Across Multiple Large Samples. *American Journal of Psychiatry* 176, 5 (May 2019), 376–387. https://doi.org/10.1176/appi.ajp.2018.18070881
[17] Nate Breznau, Eike Mark Rinke, Alexander Wuttke, Hung H. V. Nguyen, Muna Adem, Jule Adriaans, Amalia Alvarez-Benjumea, Henrik K. Andersen, Daniel Auer, Flavio Azevedo, Oke Bahnsen, Dave Balzer, Gerrit Bauer, Paul C. Bauer, Markus Baumann, Sharon Baute, Verena Benoit, Julian Bernauer, Carl Berning, Anna Berthold, Felix S. Bethke, Thomas Biegert, Katharina Blinzler, Johannes N. Blumenberg, Licia Bobzien, Andrea Bohman, Thijs Bol, Amie Bostic, Zuzanna Brzozowska, Katharina Burgdorf, Kaspar Burger, Kathrin B. Busch, Juan Carlos-Castillo, Nathan Chan, Pablo Christmann, Roxanne Connelly, Christian S. Czymara, Elena Damian, Alejandro Ecker, Achim Edelmann, Maureen A. Eger, Simon Ellerbrock, Anna Forke, Andrea Forster, Chris Gaasendam, Konstantin Gavras, Vernon Gayle, Theresa Gessler, Timo Gnambs, Amélie Godefroidt, Max Grömping, Martin Groß, Stefan Gruber, Tobias Gummer, Andreas Hadjar, Jan Paul Heisig, Sebastian Hellmeier, Stefanie Heyne, Magdalena Hirsch, Mikael Hjerm, Oshrat Hochman, Andreas Hövermann, Sophia Hunger, Christian Hunkler, Nora Huth, Zsófia S. Ignácz, Laura Jacobs, Jannes Jacobsen, Bastian Jaeger, Sebastian Jungkunz, Nils Jungmann, Mathias Kauff, Manuel Kleinert, Julia Klinger, Jan-Philipp Kolb, Marta Kołczyńska, John Kuk, Katharina Kunißen, Dafina Kurti Sinatra, Alexander Langenkamp, Philipp M. Lersch, Lea-Maria Löbel, Philipp Lutscher, Matthias Mader, Joan E. Madia, Natalia Malancu, Luis Maldonado, Helge Marahrens, Nicole Martin, Paul Martinez, Jochen Mayerl, Oscar J. Mayorga, Patricia McManus, Kyle McWagner, Cecil Meeusen, Daniel Meierrieks, Jonathan Mellon, Friedolin Merhout, Samuel Merk, Daniel Meyer, Leticia Micheli, Jonathan Mijs, Cristóbal Moya, Marcel Neunhoeffer, Daniel Nüst, Olav Nygård, Fabian Ochsenfeld, Gunnar Otte, Anna O. Pechenkina, Christopher Prosser, Louis Raes, Kevin Ralston, Miguel R. Ramos, Arne Roets, Jonathan Rogers, Guido Ropers, Robin Samuel, Gregor Sand, Ariela Schachter, Merlin Schaeffer, David Schieferdecker, Elmar Schlueter, Regine Schmidt, Katja M. Schmidt, Alexander Schmidt-Catran, Claudia Schmiedeberg, Jürgen Schneider, Martijn Schoonvelde, Julia Schulte-Cloos, Sandy Schumann, Reinhard Schunck, Jürgen Schupp, Julian Seuring, Henning Silber, Willem Sleegers, Nico Sonntag, Alexander Staudt, Nadia Steiber, Nils Steiner, Sebastian Sternberg, Dieter Stiers, Dragana Stojmenovska, Nora Storz, Erich Striessnig, Anne-Kathrin Stroppe, Janna Teltemann, Andrey Tibajev, Brian Tung, Giacomo Vagni, Jasper Van Assche, Meta van der Linden, Jolanda van der Noll, Arno Van Hootegem, Stefan Vogtenhuber, Bogdan Voicu, Fieke Wagemans, Nadja Wehl, Hannah Werner, Brenton M. Wiernik, Fabian Winter, Christof Wolf, Yuki Yamada, Nan Zhang, Conrad Ziller, Stefan Zins, and Tomasz Żółtak. 2022. Observing many researchers using the same data and hypothesis reveals a hidden universe of uncertainty. *Proceedings of the National Academy of Sciences* 119, 44 (Nov. 2022), e2203150119. https://doi.org/10.1073/pnas.2203150119 Publisher: Proceedings of the National Academy of Sciences.
[18] Joseph Cesario, David J. Johnson, and William Terrill. 2019. Is There Evidence of Racial Disparity in Police Use of Deadly Force? Analyses of Officer-Involved Fatal Shootings in 2015–2016. *Social Psychological and Personality Science* 10, 5 (July 2019), 586–595. https://doi.org/10.1177/1948550618775108 Publisher: SAGE Publications Inc.
[19] Egon Dejonckheere, Merijn Mestdagh, Marlies Houben, Yasemin Erbas, Madeline Pe, Peter Koval, Annette Brose, Brock Bastian, and Peter Kuppens. 2018. The bipolarity of affect and depressive symptoms. *Journal of Personality and Social Psychology* 114, 2 (Feb. 2018), 323–341. https://doi.org/10.1037/pspp0000186
[20] Pierre Dragicevic, Yvonne Jansen, Abhraneel Sarma, Matthew Kay, and Fanny Chevalier. 2019. Increasing the Transparency of Research Papers with Explorable Multiverse Analyses. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM, Glasgow Scotland Uk, 1–15. https://doi.org/10.1145/3290605.3300295
[21] Jean-Rémy Falleri, Floréal Morandat, Xavier Blanc, Matias Martinez, and Martin Monperrus. 2014. Fine-grained and Accurate Source Code Differencing. In *Proceedings of the International Conference on Automated Software Engineering*. Västeras, Sweden, 313–324. https://doi.org/10.1145/2642937.2642982
[22] Jean-Rémy Falleri, Floréal Morandat, Xavier Blanc, Matias Martinez, and Martin Monperrus. 2014. Fine-grained and Accurate Source Code Differencing. In *Proceedings of the International Conference on Automated Software Engineering*. Västeras, Sweden, 313–324. https://doi.org/10.1145/2642937.2642982

[23] Kiran Gadhave, Jochen Görtler, Zach Tyler Cutler, Carolina Nobre, Oliver Deussen, Miriah Meyer, Jeff Phillips, and Alexander Lex. 2020. Predicting Intent Behind Selections in Scatterplot Visualizations. https://doi.org/10.31219/osf.io/mq2rk

[24] Joachim Gassen. 2022. Researcher Degrees of Freedom Analysis. https://joachim-gassen.github.io/rdfanalysis/index.html

[25] Brian D. Hall, Yang Liu, Yvonne Jansen, Pierre Dragicevic, Fanny Chevalier, and Matthew Kay. 2022. A Survey of Tasks and Visualizations in Multiverse Analysis Reports. *Computer Graphics Forum* 41, 1 (2022), 402–426. https://doi.org/10.1111/cgf.14443 _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14443.

[26] Eunice Jun, Maureen Daum, Jared Roesch, Sarah E. Chasins, Emery D. Berger, Rene Just, and Katharina Reinecke. 2019. Tea: A High-level Language and Runtime System for Automating Statistical Analysis. *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology* (Oct. 2019), 591–603. https://doi.org/10.1145/3332165.3347940 arXiv: 1904.05387.

[27] Eunice Jun, Audrey Seo, Jeffrey Heer, and René Just. 2022. Tisane: Authoring Statistical Models via Formal Reasoning from Conceptual and Data Relationships. *arXiv:2201.02705 [cs, stat]* (Jan. 2022). https://doi.org/10.1145/3491102.3501888 arXiv: 2201.02705.

[28] Kiju Jung, Sharon Shavitt, Madhu Viswanathan, and Joseph M. Hilbe. 2014. Female hurricanes are deadlier than male hurricanes. *Proceedings of the National Academy of Sciences* 111, 24 (June 2014), 8782–8787. https://doi.org/10.1073/pnas.1402786111

[29] Alex Kale, Matthew Kay, and Jessica Hullman. 2019. Decision-Making Under Uncertainty in Research Synthesis: Designing for the Garden of Forking Paths. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–14. https://doi.org/10.1145/3290605.3300432

[30] Elise K. Kalokerinos, Yasemin Erbas, Eva Ceulemans, and Peter Kuppens. 2019. Differentiate to Regulate: Low Negative Emotion Differentiation Is Associated With Ineffective Use but Not Selection of Emotion-Regulation Strategies. *Psychological Science* 30, 6 (June 2019), 863–879. https://doi.org/10.1177/0956797619838763 Publisher: SAGE Publications Inc.

[31] Richard M. Karp. 1972. Reducibility among Combinatorial Problems. In *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and sponsored by the Office of Naval Research, Mathematics Program, IBM World Trade Corporation, and the IBM Research Mathematical Sciences Department*, Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger (Eds.). Springer US, Boston, MA, 85–103. https://doi.org/10.1007/978-1-4684-2001-2_9

[32] Mary Beth Kery, Amber Horvath, and Brad Myers. 2017. Variolite: Supporting Exploratory Programming by Data Scientists. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. ACM, Denver Colorado USA, 1265–1276. https://doi.org/10.1145/3025453.3025626

[33] Mary Beth Kery and Brad A. Myers. 2018. Interactions for Untangling Messy History in a Computational Notebook. In *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 147–155. https://doi.org/10.1109/VLHCC.2018.8506576 ISSN: 1943-6106.

[34] Ruben Kleiman, Mike Brayshaw, Marc Eisenstadt, and Marc Eisenstadt. 1993. Tales of Debugging from The Front Lines.

[35] Amy J. Ko, Brad A. Myers, Michael J. Coblenz, and Htet Htet Aung. 2006. An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks. *IEEE Transactions on Software Engineering* 32, 12 (Dec. 2006), 971–987. https://doi.org/10.1109/TSE.2006.116

[36] Qisheng Li, Meredith Ringel Morris, Adam Fourney, Kevin Larson, and Katharina Reinecke. 2019. The Impact of Web Browser Reader Views on Reading Speed and User Experience. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM, Glasgow Scotland Uk, 1–12. https://doi.org/10.1145/3290605.3300754

[37] Yang Liu, Tim Althoff, and Jeffrey Heer. 2020. Paths Explored, Paths Omitted, Paths Obscured: Decision Points & Selective Reporting in End-to-End Data Analysis. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. ACM, Honolulu HI USA, 1–14. https://doi.org/10.1145/3313831.3376533

[38] Yang Liu, Alex Kale, Tim Althoff, and Jeffrey Heer. 2021. Boba: Authoring and Visualizing Multiverse Analyses. *IEEE Transactions on Visualization and Computer Graphics* 27, 2 (Feb. 2021), 1753–1763. https://doi.org/10.1109/TVCG.2020.3028985 arXiv: 2007.05551.

[39] Microsoft. 2022. Monaco Editor. https://microsoft.github.io/monaco-editor/

[40] Roberto Minelli, Andrea Mocci, and Michele Lanza. 2015. I Know What You Did Last Summer - An Investigation of How Developers Spend Their Time. In *2015 IEEE 23rd International Conference on Program Comprehension*. 25–35. https://doi.org/10.1109/ICPC.2015.12 ISSN: 1092-8138.

[41] Guiomar Niso, Rotem Botvinik-Nezer, Stefan Appelhoff, Alejandro De La Vega, Oscar Esteban, Joset A. Etzel, Karolina Finc, Melanie Ganz, Remi Gau, Yaroslav O. Halchenko, Peer Herholz, Agah Karakuzu, David Keator, Camille Maumet, Christopher J. Markiewicz, Dr Cyril Pernet, Franco Pestilli, Nazek Queder, Tina Schmitt, Weronika Sójka, Adina Svenja Wagner, Kirstie Whitaker, and Jochem Rieger. 2022. Open and reproducible neuroimaging: from study inception to publication. https://doi.org/10.31219/osf.io/pu5vb

[42] OPEN SCIENCE COLLABORATION. 2015. Estimating the reproducibility of psychological science. *Science* 349, 6251 (Aug. 2015), aac4716. https://doi.org/10.1126/science.aac4716 Publisher: American Association for the Advancement of Science.

[43] Mark Otto and Jacob Thornton. 2022. Bootstrap. https://getbootstrap.com/

[44] Chirag J. Patel, Belinda Burford, and John P. A. Ioannidis. 2015. Assessment of vibration of effects due to model specification can demonstrate the instability of observational associations. *Journal of Clinical Epidemiology* 68, 9 (Sept. 2015), 1046–1058. https://doi.org/10.1016/j.jclinepi.2015.05.029

[45] Gregory J Poarch, Jan Vanhove, and Raphael Berthele. 2019. The effect of bidialectalism on executive function. *International Journal of Bilingualism* 23, 2 (April 2019), 612–628. https://doi.org/10.1177/1367006918763132 Publisher: SAGE Publications Ltd.

[46] James R. Rae, Selin Gülgöz, Lily Durwood, Madeleine DeMeules, Riley Lowe, Gabrielle Lindquist, and Kristina R. Olson. 2019. Predicting Early-Childhood Gender Transitions. *Psychological Science* 30, 5 (May 2019), 669–681. https://doi.org/10.1177/0956797619830649 Publisher: SAGE Publications Inc.

[47] Armin Ronacher. 2022. Welcome to Flask — Flask Documentation (2.2.x). https://flask.palletsprojects.com/en/2.2.x/

[48] Abhraneel Sarma, Alexander Kale, Michael Jongho Moon, Nathan Taback, Fanny Chevalier, Jessica Hullman, and Matthew Kay. 2021. *multiverse: Multiplexing Alternative Data Analyses in R Notebooks*. Technical Report. OSF Preprints. https://doi.org/10.31219/osf.io/yfbwm type: article.

[49] Tal Schuster, Ashwin Kalyan, Alex Polozov, and Adam Kalai. 2021. Programming Puzzles. *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks* 1 (Dec. 2021). https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/3988c7f88ebcb58c6ce932b957b6f332-Abstract-round1.html

[50] Martin Schweinsberg, Michael Feldman, Nicola Staub, Olmo R. van den Akker, Robbie C.M. van Aert, Marcel A.L.M. van Assen, Yang Liu, Tim Althoff, Jeffrey Heer, Alex Kale, Zainab Mohamed, Hashem Amireh, Vaishali Venkatesh Prasad, Abraham Bernstein, Emily Robinson, Kaisa Snellman, S. Amy Sommer, Sarah M.G. Otner, David Robinson, Nikhil Madan, Raphael Silberzahn, Pavel Goldstein, Warren Tierney, Toshio Murase, Benjamin Mandl, Domenico Viganola, Carolin Strobl, Catherine B.C. Schaumans, Stijn Kelchtermans, Chan Naseeb, S. Mason Garrison, Tal Yarkoni, C.S. Richard Chan, Prestone Adie, Paulius Alaburda, Casper Albers, Sara Alspaugh, Jeff Alstott, Andrew A. Nelson, Eduardo Ariño de la Rubia, Adbi Arzi, Štěpán Bahník, Jason Baik, Laura Winther Balling, Sachin Banker, David AA Baranger, Dale J. Barr, Brenda Barros-Rivera, Matt Bauer, Enuh Blaise, Lisa Boelen, Katerina Bohle Carbonell, Robert A. Briers, Oliver Burkhard, Miguel-Angel Canela, Laura Castrillo, Timothy Catlett, Olivia Chen, Michael Clark, Brent Cohn, Alex Coppock, Natàlia Cugueró-Escofet, Paul G. Curran, Wilson Cyrus-Lai, David Dai, Giulio Valentino Dalla Riva, Henrik Danielsson, Rosaria de F.S.M. Russo, Niko de Silva, Curdin Derungs, Frank Dondelinger, Carolina Duarte de Souza, B. Tyson Dube, Marina Dubova, Ben Mark Dunn, Peter Adriaan Edelsbrunner, Sara Finley, Nick Fox, Timo Gnambs, Yuanyuan Gong, Erin Grand, Brandon Greenawalt, Dan Han, Paul H.P. Hanel, Antony B. Hong, David Hood, Justin Hsueh, Lilian Huang, Kent N. Hui, Keith A. Hultman, Azka Javaid, Lily Ji Jiang, Jonathan Jong, Jash Kamdar, David Kane, Gregor Kappler, Erikson Kaszubowski, Christopher M. Kavanagh, Madian Khabsa, Bennett Kleinberg, Jens Kouros, Heather Krause, Angelos-Miltiadis Krypotos, Dejan Lavbič, Rui Ling Lee, Timothy Leffel, Wei Yang Lim, Silvia Liverani, Bianca Loh, Dorte Lønsmann, Jia Wei Low, Alton Lu, Kyle MacDonald, Christopher R. Madan, Lasse Hjorth Madsen, Christina Maimone, Alexandra Mangold, Adrienne Marshall, Helena Ester Matskewich, Kimia Mavon, Katherine L. McLain, Amelia A. McNamara, Mhairi McNeill, Ulf Mertens, David Miller, Ben Moore, Andrew Moore, Eric Nantz, Ziauddin Nasrullah, Valentina Nejkovic, Colleen S Nell, Andrew Arthur Nelson, Gustav Nilsonne, Rory Nolan, Christopher E. O'Brien, Patrick O'Neill, Kieran O'Shea, Toto Olita, Jahna Otterbacher, Diana Palsetia, Bianca Pereira, Ivan Pozdniakov, John Protzko, Jean-Nicolas Reyt, Travis Riddle, Amal (Akmal) Ridhwan Omar Ali, Ivan Ropovik, Joshua M. Rosenberg, Stephane Rothen, Michael Schulte-Mecklenbeck, Nirek Sharma, Gordon Shotwell, Martin Skarzynski, William Stedden, Victoria Stodden, Martin A. Stoffel, Scott Stoltzman, Subashini Subbaiah, Rachael Tatman, Paul H. Thibodeau, Sabina Tomkins, Ana Valdivia, Gerrieke B. Druijff-van de Woestijne, Laura Viana, Florence Villesèche, W. Duncan Wadsworth, Florian Wanders, Krista Watts, Jason D Wells, Christopher E. Whelpley, Andy Won, Lawrence Wu, Arthur Yip, Casey Youngflesh, Ju-Chi Yu, Arash Zandian, Leilei Zhang, Chava Zibman, and Eric Luis Uhlmann. 2021. Same data, different conclusions: Radical dispersion in empirical results when independent analysts operationalize and test the same hypothesis. *Organizational Behavior and Human Decision Processes* 165 (July 2021), 228–249. https://doi.org/10.1016/j.obhdp.2021.02.003

[51] R. Silberzahn, E. L. Uhlmann, D. P. Martin, P. Anselmi, F. Aust, E. Awtrey, Š. Bahník, F. Bai, C. Bannard, E. Bonnier, R. Carlsson, F. Cheung, G. Christensen, R. Clay, M. A. Craig, A. Dalla Rosa, L. Dam, M. H. Evans, I. Flores Cervantes, N. Fong, M. Gamez-Djokic, A. Glenz, S. Gordon-McKeon, T. J. Heaton, K. Hederos,

M. Heene, A. J. Hofelich Mohr, F. Högden, K. Hui, M. Johannesson, J. Kalodimos, E. Kaszubowski, D. M. Kennedy, R. Lei, T. A. Lindsay, S. Liverani, C. R. Madan, D. Molden, E. Molleman, R. D. Morey, L. B. Mulder, B. R. Nijstad, N. G. Pope, B. Pope, J. M. Prenoveau, F. Rink, E. Robusto, H. Roderique, A. Sandberg, E. Schlüter, F. D. Schönbrodt, M. F. Sherman, S. A. Sommer, K. Sotak, S. Spain, C. Spörlein, T. Stafford, L. Stefanutti, S. Tauber, J. Ullrich, M. Vianello, E.-J. Wagenmakers, M. Witkowiak, S. Yoon, and B. A. Nosek. 2018. Many Analysts, One Data Set: Making Transparent How Variations in Analytic Choices Affect Results. *Advances in Methods and Practices in Psychological Science* 1, 3 (Sept. 2018), 337–356. https://doi.org/10.1177/2515245917747646 Publisher: SAGE Publications Inc.

[52] Uri Simonsohn, Joseph P. Simmons, and Leif D. Nelson. 2019. Specification Curve: Descriptive and Inferential Statistics on All Reasonable Specifications. https://doi.org/10.2139/ssrn.2694998

[53] Uri Simonsohn, Joseph P. Simmons, and Leif D. Nelson. 2020. Specification curve analysis. *Nature Human Behaviour* 4, 11 (Nov. 2020), 1208–1214. https://doi.org/10.1038/s41562-020-0912-z

[54] Gary Smith. 2022. Full moons and forking paths. *Significance* 19, 4 (2022), 32–35. https://doi.org/10.1111/1740-9713.01672 _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/1740-9713.01672.

[55] Sara Steegen, Francis Tuerlinckx, Andrew Gelman, and Wolf Vanpaemel. 2016. Increasing Transparency Through a Multiverse Analysis. *Perspectives on Psychological Science* 11, 5 (Sept. 2016), 702–712. https://doi.org/10.1177/1745691616658637 Publisher: SAGE Publications Inc.

[56] Wolf Vanpaemel, Sara Steegen, Francis Tuerlinckx, and Andrew Gelman. 2016. multiverse analysis. (May 2016). https://osf.io/zj68b/ Publisher: OSF.

[57] Xin Xia, Lingfeng Bao, David Lo, Zhenchang Xing, Ahmed E. Hassan, and Shanping Li. 2018. Measuring Program Comprehension: A Large-Scale Field Study with Professionals. *IEEE Transactions on Software Engineering* 44, 10 (Oct. 2018), 951–976. https://doi.org/10.1109/TSE.2017.2734091 Conference Name: IEEE Transactions on Software Engineering.

A INITIAL CORRESPONDENCES WITH MULTIVERSE EXPERTS

A.1 Interviews with Two Multiverse Practitioners

To identify specific challenges in authoring and conducting multiverse analyses, we first conduct in-depth interview studies with two researchers who have recently authored multiverse analyses. We found these researchers through our collaboration networks. Neither relied on existing multiverse tools. Instead, they wrote custom scripts that generated each universe script. During the interviews, which lasted for approximately two hours each, the researchers walked us through their analyses, including their scripts, findings, and any historical artifacts from their git repository histories. Without being prompted, both brought up how challenging finding and propagating bug fixes is for them.

We learned that the researchers approach authoring multiverse analyses in a bottom-up, iterative fashion. They focus on a few key decisions and options, consult their peers and supervisors, and then add additional decisions and options based on their team's input. This iterative nature requires keeping track of which combinations of decision options were previously considered and how, if at all, the results have altered since changing or adding decisions and decision options. The same process applies when the researchers encounter and fix bugs. They must identify bugs, fix decision options that introduce the bugs, and then re-run their multiverse analyses to see how the bugs impact their results.

This led to an understanding that multiverse debugging is a key challenge and that resolving difficulties surrounding this process could make it easier to author multiverse analyses more generally.

A.2 Additional Correspondences with Experienced Multiverse Tool Developers

We cross-examined our observed challenges and insights in debugging with two independent, experienced researchers who have authored multiverse analyses and developed multiverse analysis tools. We corresponded with these researchers via email.

Both researchers corroborated the importance of starting with a single universe and then propagating changes to the rest of the universes: "*I may look at a single universe. Then I apply the solution to all affected paths. Currently, this can only be achieved by modifying the multiverse specification.*" The other researcher had a similar debugging process: "*I always debug by looking at individual universe scripts that instantiate a particular set of decisions that I think might be involved in the error*". They also mentioned how debugging multiverse analyses is like debugging a single universe but with "*the added difficulty of figuring out why the bugs come up in a particular analysis*". Finally, one tool developer also highlighted the additional steps needed to pinpoint an error: "*I often read the error messages and pick a specific error to focus on. Then I examine all paths that lead to a specific error to distill commonality.*"

B DECISION COVER ALGORITHM

The DECISION COVER algorithm is an iterative loop of sampling a universe from the multiverse and reducing the multiverse by removing all universes that contain decision options of universes sampled so far. Algorithm algorithm 1 summarizes the DECISION COVER algorithm. We start with the set of all universes. Until this set is empty, a universe is randomly sampled and all universes that share any decision option will be removed from this set. We take the set of sampled universes as the reduced set of universes to run.

Algorithm 1: Decision Cover

Input: $S = \{u_i \mid i = 1 \dots n\}$ (set of universes in the entire multiverse), $D = \{d_j \mid j = 1 \dots m\}$ (set of decision options in the multiverse) ; `// Each universe` u_i `is represented by a unique set of decision options` $D_{u_i} \subset D$

Initialize $M \leftarrow \emptyset$

while $S \neq \emptyset$ **do**

  $u \sim Uniform(S)$ $M \leftarrow M \cup \{u\}$ $T \leftarrow \{u\}$ **for** $v \in S$ **do**

    if $D_u \cap D_v \neq \emptyset$ **then**

      $T \leftarrow T \cup \{v\}$

    end

  end

  $S \leftarrow S \setminus T$

end

return M

C ALGORITHM FOR UNIVERSE-TO-MULTIVERSE-SPECIFICATION DIFFS

C.1 Boba Background

There are two main ways to specify decisions in the *template* file: *placeholder variables* for decision options that can be placed in-line and *code blocks* for decision options that involve multiple lines of code. Placeholder variables can be placed anywhere in the template file. Users specify the placeholder decision name and its alternative options. During compilation, Boba removes the placeholder identifier and replaces it with one of its alternative values. In Figure 2A, the `cutoff` and `brm_family` decisions are defined with placeholder variables. Meanwhile, a decision block is used to specify multiple versions of a code block that act as alternative decision options for one analytical decision. For example, in Figure 2A, the `Model` decision block consists of two alternative code blocks, representing an option for a *frequentist* model and an option for a *bayesian* model. When compiling the *template* file, Boba will instantiate only one code block corresponding to a decision in a universe.

C.2 Algorithm

MULTIVERSE DEBUGGER compares abstract syntax trees (ASTs) and lines of code of the edited and unedited universe. ASTs provide the granularity needed to identify decision options and potential changes to these options that are specified in-line (Boba placeholder variables) in the new universe. Meanwhile, comparing code at the line granularity helps locate decision options specified by multiple lines of code (Boba code blocks). Furthermore, comparing lines also helps map universe code blocks to the multiverse specification blocks.

We use information from the compilation process to know where in the unedited old universe the Boba variables are located and the split points between Boba code blocks. In short, we have a mapping between the unedited universe and the multiverse specification. We then find where in the new edited universe the locations of Boba variables are via AST matching and locations of Boba code blocks via line matching. Through the mapping between old and new universe, we can then map changes in the new universe all the way back to the multiverse specification.

To pinpoint code changes in the universe that correspond to decision options specified inline in the multiverse specification, we match the ASTs of the unedited and edited universes. Matching ASTs provides additional granularity than line difference algorithms and enables direct mappings between code that corresponds to matched subtrees in the AST. We use gumtree [22]to find code in the new universe that corresponds to Boba variables. If changes exist, these are mapped to the multiverse specification.

We use the Python difflib [2] library's mdiff function to match the start of code blocks between the old and new universe files. For each line in the old universe if it is matched with the new universe and it is the start of the block boundary, we add the new universe line as the start of the corresponding Boba block. If the line is deleted and it is at the boundary of the Boba block, we add the next line in the new universe. Finally, if a new line is inserted and it is at the start of a new block, we always default to including it at the start of a new block. With our initial multiverse specification and unedited universe mapping, we can propagate edits in the universe back to the multiverse specification.

The UNIVERSE-TO-MULTIVERSE DIFF algorithm based on gumtree's AST matching algorithm is best suited for small to medium edit changes. As these edits are common in most of bug fixes, gumtree is an adequate choice.

D PROCESS FOR FINDING BUGS FOR THE LAB STUDY

We gathered two multiverses from which we created buggy R and Python versions. The first multiverse, HURRICANE, is authored by Simonsohn *et al.* [53] and challenges the reported analysis in a previous study [28]. The study explored whether hurricanes with female names resulted in more deaths. The second multiverse, READING, is an example from Boba [38]. READING is based on how researchers of a published paper [36], on whether different web layouts result in faster reading speeds, might construct a multiverse from their analysis.

To introduce realistic bugs, we first identified common bugs encountered during typical statistical analyses. We searched Stack Overflow [6] to find errors. For R, we searched Stack Overflow with tags `R` and keyword *error* to find relevant posts. Similarly, for Python, we searched with tags `Python`, `pandas`[4], and `statsmodels`[3] and the keyword *error* to find relevant posts. In addition to Stack Overflow, we consulted an online statistics blog with consolidated lists of Python [14] and R errors [15].

This resulted in errors that encompass data parsing, data splitting, and model specification. The R version of HURRICANE included 5 errors. One was a syntax error, one was a logical one-off error, two more errors were errors that resulted from poor data processing, and the last error was a model fit error due to a poorly specified model formula. The Python version contained 3 errors: the same one-off error, a data processing error, and the same model fit error.

For the READING multiverse, the R version involved 3 errors: two errors related to poor data/model specification, and a third error with misspecified data transformation. The Python version had 3 errors as well: an error as a result of using the wrong model, an error with the wrong syntax for data filtering, and a third error from parsing the data improperly. We include all lab study materials in our supplemental material.