Explaining Software Fault Predictions to Spreadsheet Users[1]

Adil Mukhtar[a,1], Birgit Hofer[a], Dietmar Jannach[b], Franz Wotawa[a]

[a] *Graz University of Technology, Austria*
[b] *University of Klagenfurt, Austria*

Abstract

A variety of automated software fault prediction techniques was proposed in recent years, in particular for the important class of spreadsheet programs. Software fault prediction techniques commonly create ranked lists of "suspicious" program statements for developers to inspect. Existing research, however, suggests that solely providing such ranked lists may not always be effective. In particular, it was found that developers often seek for *explanations* for the outcomes provided by a debugging tool and that such explanations may be key for developers to trust and rely on the tool. Research on *how to explain* the outcomes of fault prediction techniques, which are often based on complex machine learning models, is scarce, and little is known regarding how such explanations are perceived by developers. With this work, we aim to narrow this research gap and study the perception of different forms of explanations by spreadsheet users in the context of a machine learning based fault prediction tool. A between-subjects user study (N=120) revealed significant differences between the explored explanation styles. In particular, we found that well-designed natural language explanations can indeed help users better understand why certain spreadsheet cells were marked by the debugging tool and that such explanations can be effective to increase the users' trust compared to a black box system.

Keywords:
Fault Prediction, Explanations, Spreadsheets, User Study, Explainable AI

1. Introduction

Over the last decades, a multitude of automated techniques were proposed that aim for supporting software developers prevent, localizing, or even correcting faults in their programs. Software fault prediction is a particular class of such approaches, which aim to pinpoint fault-prone parts of a program as early as possible, ideally even before system testing. Technically, the prediction task is often framed as a supervised machine learning problem, where software-related metrics in combination with past fault data are used to estimate the likelihood that a given part of a software artifact is faulty, see [1] for a survey. In recent years, such learning-based fault prediction methods—as well as various other advanced debugging techniques— were also successfully applied to the important class of spreadsheet programs [2]. Nowadays, spreadsheet programs are used widely in organizations for various forms of data analysis and decision-making, and it is not uncommon that business-critical decisions are based on calculations in spreadsheets. Thus, appropriate quality assurance techniques for spreadsheet programs—which are the focus of our work—are of high practical importance.[2]

[1]Paper accepted for publication in *Journal of Systems and Software, 2023*

Email addresses: `amukhtar@ist.tugraz.at` (Adil Mukhtar), `bhofer@ist.tugraz.at` (Birgit Hofer), `dietmar.jannach@aau.at` (Dietmar Jannach), `wotawa@ist.tugraz.at` (Franz Wotawa)

[2]See `http://www.eusprig.org/horror-stories.htm` for a list of "horror stories", where faults in spreadsheets led to significant financial losses for organizations.

Today's research on automated fault prediction and fault localization—both for spreadsheets and traditional programs—is largely based on data-based ("offline") experiments, where machine learning models are trained and assessed without involving humans in the evaluation process. The central task in such machine learning approaches is to predict the likelihood of a certain part of a given software, e.g., a statement in a program or a formula in a spreadsheet, being faulty. Ultimately, the main outcome of such techniques is typically an ordered list of program elements that are considered suspicious. One fault prediction technique is then considered better than another one if it *(a)* more often ranks the true faults higher in this list and *(b)* includes fewer "false alarms" at the top places. Both aspects can be evaluated with common accuracy and ranking metrics from the field of information retrieval, e.g., with precision, recall and Mean Reciprocal Rank [3].

While such an evaluation approach is widely used in the academic literature, Parnin and Orso [4] questioned if the underlying assumptions of this approach hold, and if we can be sure that automated debugging tools are helping programmers in practice. One main outcome of their studies was that some central assumptions may not always hold in practice. Programmers for example may not necessarily inspect a given list of potentially faulty statements strictly in order; also, they might not identify a statement as being faulty, even though it is presented as potentially faulty by the debugging tool. Different remedies to these issues were proposed, including in particular the use of alternative evaluation metrics, support for program comprehension, the provision of ecosystems that support the entire toolchain, and more user studies [5].

In the context of program comprehension, one important direction to improve the practical usefulness of automated debugging techniques lies in the provision of *explanations*. In a recent study on defect prediction methods, Wan and colleagues [6] found that several respondents of their survey explicitly mentioned it to be a desirable feature if the tool could explain "*[...] why an error is considered an error*" or "*[t]ell my why certain lines are potentially wrong*". In this context, a largely open question is *how* a system should explain such aspects to its users: What kind of information should be displayed? How much detail should be presented? Should the information be conveyed through natural language or some form of visualization?

In this work, we investigate these questions for the particular case of spreadsheet debugging. Specifically, we investigate different forms of explanations for the machine-learning-based spreadsheet debugging approach proposed in [7]. Given a spreadsheet as input, this debugging technique returns a "suspiciousness" score for each formula in the spreadsheet.[3] The data-based experiments in [7] indicate that the method is highly effective to identify faults in spreadsheets. Continuing this research, our present goal is to explore in which ways explanations can help to further increase the usefulness of the automated fault prediction method. In particular, we aim to assess to what extent different forms of explanations help developers to understand why a given formula is faulty. Moreover, we are interested in the effects of providing explanations to spreadsheet users[4] in terms of the quality perception of the debugging tool, e.g., perceived usefulness, usability, transparency, and trust. Therefore, the main research questions in this work are as follows.

- **RQ-1**: *Can explanations for individual fault predictions help spreadsheet users obtain a better understanding of the spreadsheet program and its faults?*
- **RQ-2**: *Can explanations help to improve the quality perception of the debugging tool by spreadsheet users and the trust in the system?*

[3]We note here that we focus on *faulty formulas* in this work. Extending the scope to erroneous input data or constants is possible in principle. For more general discussions of faults (errors) in spreadsheets, we refer to [8, 9].

[4]In the context of our work, we use the term "spreadsheet users" to refer to people who use spreadsheet on a regular basis, who understand the basic concept of entering formulas in cells, and who are able to create or modify spreadsheets by themselves.

To answer these research questions, we conducted a between-subjects user study (N=120), where participants were presented with faulty spreadsheets in which several formulas were highlighted as being suspicious. We then provided different forms of explanations to the participants and asked them to *(a)* describe why they think that a given formula might be faulty and *(b)* report on their perception of the debugging tool in different dimensions. The results of our study indicate that providing explanations can indeed be beneficial compared to a black box system both in terms of program comprehension and other quality dimensions. We also observed that the particular form of explaining matters.

Overall, with our work, we *contribute to a better understanding of the value of explanations in the context of automated fault prediction techniques for the class of spreadsheet programs*. To our knowledge, no prior work exists that examines such questions in the domain of spreadsheets. To what extent our findings generalize to other types of programs, however, still has to be explored. While we could merely see spreadsheets as a particular type of software, one crucial difference to general software development is that the users and creators of spreadsheets often have no formal education in software engineering. Moreover, there are typically no *systematic* testing and debugging procedures implemented during spreadsheet development, and automated fault localization and debugging support in today's spreadsheet environments is still limited. These aspects make the design of the technical solutions to present explanations to spreadsheet users particularly challenging, because appropriate and concise ways of conveying the relevant information, e.g., through text summarizations, visualizations or explanation-by-example [10], must be designed to ensure the understandability of the explanations for this particular type of users.

On a more general level, our work may also be seen as a contribution to the area of Explainable AI (XAI), where one main goal is to make the underlying reasoning of an AI-based system transparent to developers and end users. In the context of automated fault prediction, explainability turns out to be particularly challenging when the underlying machine learning models become increasingly complex with the use of modern deep learning architectures, see [11] for an overview.

The paper is organized as follows. We discuss related work next in Section 2. Our research methodology and the details of our experimental design are described in Section 3. In Section 4 we discuss the obtained results and practical implications in detail. The paper ends with a summary and an outlook on future works in Section 6.

2. Related Work

The related works discussed in this section cover software fault prediction techniques and the use of explanations in the context of software engineering and in particular debugging. In the following, we outline recent work on fault prediction, discuss existing studies with humans in this domain, and elaborate on research dealing with explanations given to users with the aim of improving usability and trust.

2.1. Software Fault Prediction Methods

In software development and maintenance, we spend a lot of effort that is related to the detection of failures and the identification of their corresponding root causes. Hence, there has always been an interest in automating debugging, which aims at identifying those parts of a program that potentially induce failures as early as possible. For a recent survey on software fault localization, we refer to [12]. In contrast to regular debugging activities in practice, software fault prediction, also called software defect prediction, does not require passing and failing test executions. Instead, software fault prediction utilizes information regarding programming patterns and metrics for estimating whether or not certain parts of a program are faulty. In the following, we summarize related research focusing on software fault prediction only.

We can view software fault prediction as an extension to the *checking approach* to debugging. According to Ducasse [13], a checking strategy "*parses programs and searches* *for language-dependent errors,*" where we consider program parts suspicious when they are not conforming to well-formedness rules or conform to error rules. In contrast to checking, software fault prediction goes beyond simple classification and assigns suspiciousness values to potentially faulty program parts. These suspiciousness values should correspond to the likelihood of faultiness and allow ranking of program parts.

For finding a function that maps programming patterns or metrics to a suspiciousness value, most recent work on software fault prediction relies on machine learning, e.g., see [14], where we use information regarding patterns or metrics and their effect on the behavior to learn the prediction function. A variety of different machine learning approaches has been used in the past, e.g., neural networks [15], Bayesian networks [16], and association rule mining [17] among others. There are also several surveys which deal with general fault prediction [1, 18, 19, 20, 21, 22], and the application of machine learning to fault prediction [23, 11, 24, 25].

In the context of spreadsheets, Koch and colleagues [2] successfully applied machine learning for software fault prediction based on metrics. In their paper, the authors compared different machine learning approaches for predicting the faultiness of spreadsheet cells based on several metrics. For a survey on known spreadsheet metrics, we refer the reader to [26].

Beyond the spreadsheet domain, there are several studies which evaluate and compare different fault prediction methods. The most recent studies include the following. Razu et al. [27] compared several machine learning techniques for fault prediction using well-known evaluation benchmarks. He and colleagues [28] describe and evaluate an approach for computing a reduced set of metrics for estimating defects in software. Choudhary et al. [29] introduced change metrics for software fault prediction. The commonality of these studies is the use of program collections for determining the prediction accuracy of the underlying software fault prediction approaches. They do not consider user-related characteristics, which include the perceived usefulness of predictions.

To the best of our knowledge, there is only little research that considers human-centric evaluation approaches for automated debugging tools, including [30, 31, 32, 4, 33, 34]. Most of these papers report results obtained from user studies which either involve students or a limited number of developers. For spreadsheet debugging, the situation is similar. Papers dealing with user studies like [35, 36, 37, 38] specifically concentrate on user-related characteristics of one particular tool. The content of this paper is distinct from previous work, elaborating on the effects of different ways a spreadsheet debugging tool passes explanations to the user.

2.2. Explanations

In an informal way, an explanation can be seen as a narrative answering the question of why particular facts (e.g., an event or a decision) are happening. Such a narrative may explain things in terms of causality, i.e., tracing back the fact to root causes from which we can deduce the particular fact. Although it is usually the case that one person (i.e., the explainer) is explaining a decision or event to another person (i.e., the explainee), adding similar capabilities to engineered systems has been of growing interest recently. This increase in interest might be due to the fact of increasing automation, which relies more and more on AI technology. To gain trust in decisions taken by AI, users are often interested in their origins, which leads to the field of explainable AI (XAI). Miller [39] provides an excellent discussion on explainability and the need for considering research in philosophy, psychology, and cognitive science when constructing XAI. In his paper, Miller distinguishes two complementary approaches for AI-based systems: "*(1) generating decisions in which one of the criteria taken into account during the computation is how well a human could understand the decisions in the given context, which is often called interpretability or explainability; and (2) explicitly explaining decisions to people, which we will call explanation.*". In this paper, the context is more related to the first approach, for which we discuss related research in more detail.

In the context of software debugging, a widely cited study from Parnin and Orso [4] questioned the applicability of debugging tool in real life. One of the main implications from their work is to improve the explanatory capabilities of debugging tools. They found that expert programmers are faster with tool support, and that such debugging tools do not provide enough explanations when performing harder tasks. Many, if not most, of the fault localization and software fault prediction tools described in research articles rank the faulty components of a program in a list based on the suspiciousness score. These faulty components can be explained depending on the underlying methodology, however, in machine learning approaches the explanations are often generated through heuristic methods.

In a shallow way, we may also consider a given ranked list as an explanation for a bug or faulty proneness of statements. Such a list provides information on which part of a program is more likely to lead to misbehavior or a potential threat. Of course, it does not explain the ranking itself. We can extract such an explanation from the underlying ranking method. We only need to state how to compute all the ranking scores. However, a good explanation for any element in the ranked list would indicate why it is considered faulty and how we can distinguish it from the others. Parnin and Orso mentioned that the ranked list is not leading to a sufficient bug understanding and require improved explanatory capabilities. In this paper, we contribute to this challenge comparing different kinds of explanations given to users for bug candidates.

It is worth mentioning that Parnin and Orso [4] also provided a survey on user studies to evaluate automated debugging tools. In this study, the authors discuss the work [30, 31] by Ko and Myers on introducing the *Whyline* tool that allows to answer *why* and *why not* questions. This kind of explanation seems to improve the understanding of programs, and to increase debugging efficiency substantially. The *Whyline* tool goes beyond ranked lists as explanations.

2.3. Beyond Ranked Lists

Besides the mentioned *Whyline* tool, there is other related work that deals with explanations beyond ranked lists. This include early work on tutoring systems [40] for novice programmers. Recent works include [41] and [42]. Fey and colleagues [41] combine fault localization with explanations that are based on counterexamples and their traces. Counterexamples help programmers to understand how and why a program breaks. Fey and colleagues also presented the results of an experimental study showing that the combination of localization and explanation improves debugging. However, the authors provide no user study. Groce et al. [42] present a debugging approach that utilizes counterfactuals and causality for explanations. The approach is also based on counterexamples and distance metrics between passing and failing runs. In our work, we contribute to improving explanations beyond providing ranked lists. In particular, our aim is to find out which information is helpful for users to improve their understanding of the origin of faults.

3. Methodology – Experiment Design

In this section, we summarize our research methodology and provide details of our experimental design. Let us recall our research goals here, which are to investigate *(i)* to what extent different types of explanations help users to understand faults in a given spreadsheet program (*RQ-1*), and *(ii)* how different types of explanations affect the quality perception of the debugging tool (*RQ-2*).

3.1. Experiment Overview

To address the research questions, we designed a user study in which participants interacted with an online version of Microsoft's Excel spreadsheet tool, which we extended with a plug-in component for debugging called SmellChecker [7]. The main functionality of the debugging plug-in is to automatically highlight formula cells in a spreadsheet that are considered "suspicious" based on a machine learning approach. For the purpose of the study, we enriched the SmellChecker tool with the capability to provide different forms of explanations to users. These explanations indicate, in different ways, why the tool considered a given formula to be potentially faulty.

All participants of our study were directed to an online spreadsheet containing *three* faults, and these faults were correctly highlighted[5] by the SmellChecker tool. The participants in each treatment group in our between-subjects user study were additionally provided with one form of explanation. The control group was only shown the "suspiciousness score" computed by the tool without further explanation. The task of the participants was then to inspect the cells that were highlighted (and explained) by the tool and to describe in their own words why the formula may be faulty. An analysis of these responses through human judges will help us answer *RQ-1* on the fault understanding of participants when given different types of explanations.

After the participants had completed this main task, they were forwarded to a comprehensive questionnaire. In this questionnaire, the participants informed about various facets of their subjective perceptions regarding the tool, e.g., in terms of usefulness and usability, and the debugging process. This questionnaire allows us address *RQ-2* on subjective user perceptions when provided different forms of explanations.

3.2. Detailed Experiment Flow

The more detailed flow of the experiment is illustrated in Figure 1. In a preparatory step, participants read the instructions that were provided online to them and which explained their tasks and how to install the plug-in in the online version of Microsoft Excel. The descriptions of the task and the user manual for the tool were tailored for the different treatment groups, as they saw different types of explanations. We provide all materials used in the study online for reproducibility.[6]

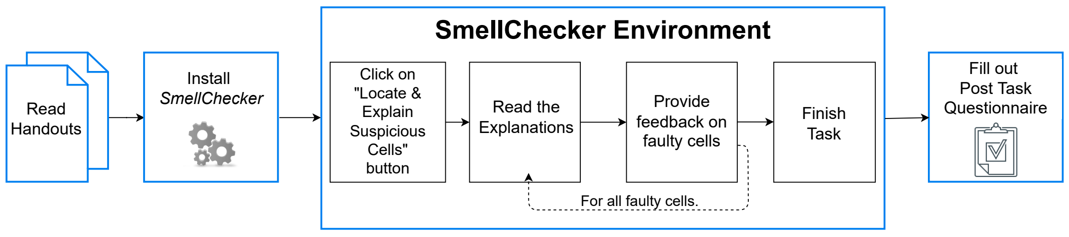

SmellChecker Environment
Read Handouts
Install *SmellChecker*
Click on "Locate & Explain Suspicious Cells" button
Read the Explanations
Provide feedback on faulty cells
Finish Task
For all faulty cells.
Fill out Post Task Questionnaire

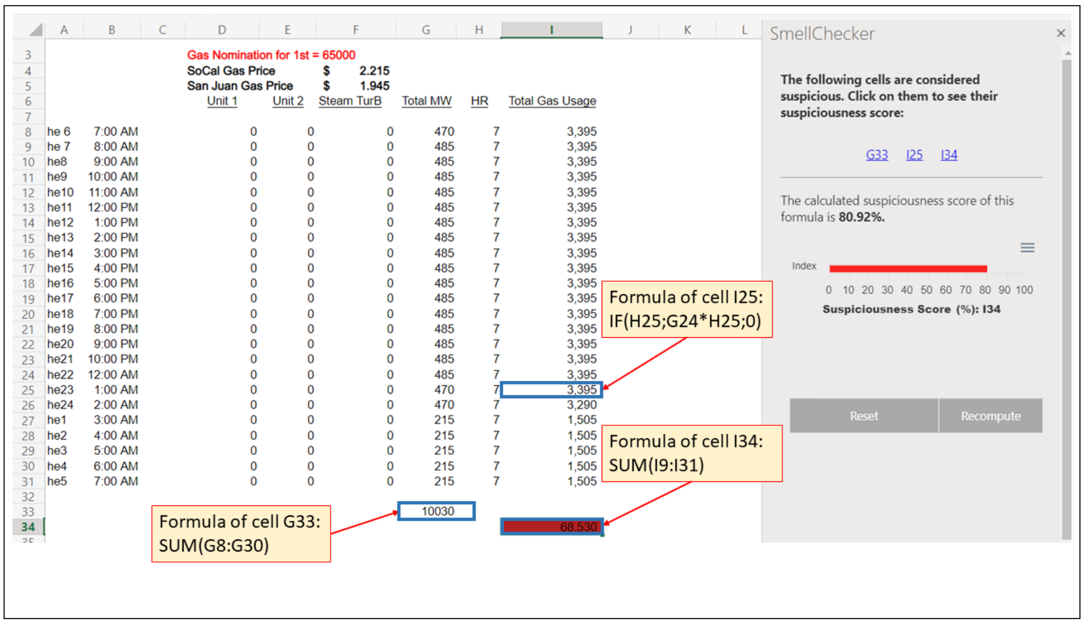
Figure 1: Overall flow of the experiment

After having provided informed consent and after the participants had installed the SmellChecker tool, they were performing the main task in the spreadsheet environment. A screen capture of how the environment looked like in the control condition—the one in which only the suspiciousness score is displayed without further explanations—is shown in Figure 2. Here, we can see that cell I34 was highlighted by the tool, and a (high) suspiciousness score of 80.92% is reported for this cell. Note that the annotations for the faulty cells shown in Figure 2 were not part of the user interface, but introduced here to better illustrate the problems in the spreadsheet.

The main task for the participants, as mentioned, was to study the information provided by the tool for each cell and then fill a free-text input box for each highlighted cell, describing

[5]In our experiment, the fault prediction tool therefore did not return "false positives", i.e., it did not mark cells as suspicious, which did not actually contain a fault. We note that dealing with false positives is a challenge that concerns the fault prediction task that precedes the explanation phase. Improving the fault prediction accuracy in our case could for example be achieved by incorporating additional types of smells in the prediction model, including those discussed in [43, 44, 45].

[6]`https://bit.ly/3QbVA6O`.



	A	B	C	D	E	F	G	H	I
3				Gas Nomination for 1st = 65000					
4				SoCal Gas Price	$	2.215			
5				San Juan Gas Price	$	1.945			
6				Unit 1	Unit 2	Steam TurB	Total MW	HR	Total Gas Usage
7									
8	he 6	7:00 AM		0	0	0	470	7	3,395
9	he 7	8:00 AM		0	0	0	485	7	3,395
10	he8	9:00 AM		0	0	0	485	7	3,395
11	he9	10:00 AM		0	0	0	485	7	3,395
12	he10	11:00 AM		0	0	0	485	7	3,395
13	he11	12:00 PM		0	0	0	485	7	3,395
14	he12	1:00 PM		0	0	0	485	7	3,395
15	he13	2:00 PM		0	0	0	485	7	3,395
16	he14	3:00 PM		0	0	0	485	7	3,395
17	he15	4:00 PM		0	0	0	485	7	3,395
18	he16	5:00 PM		0	0	0	485	7	3,395
19	he17	6:00 PM		0	0	0	485	7	3,395
20	he18	7:00 PM		0	0	0	485	7	3,395
21	he19	8:00 PM		0	0	0	485	7	3,395
22	he20	9:00 PM		0	0	0	485	7	3,395
23	he21	10:00 PM		0	0	0	485	7	3,395
24	he22	12:00 AM		0	0	0	485	7	3,395
25	he23	1:00 AM		0	0	0	470	7	3,395
26	he24	2:00 AM		0	0	0	470	7	3,290
27	he1	3:00 AM		0	0	0	215	7	1,505
28	he2	4:00 AM		0	0	0	215	7	1,505
29	he3	5:00 AM		0	0	0	215	7	1,505
30	he4	6:00 AM		0	0	0	215	7	1,505
31	he5	7:00 AM		0	0	0	215	7	1,505
32									
33							10030		
34									68.530

Formula of cell I25: IF(H25;G24*H25;0)

Formula of cell I34: SUM(I9:I31)

Formula of cell G33: SUM(G8:G30)

SmellChecker

The following cells are considered suspicious. Click on them to see their suspiciousness score:

G33 I25 I34

The calculated suspiciousness score of this formula is **80.92%.**

Figure 2: SmellChecker marks cells as faulty based on the suspiciousness score [38]

what they think might be the issue with the formula in the cell. Once this task was done for all highlighted cells the participants were forwarded to the post-task questionnaire. Note that all participants saw the same spreadsheet with the same highlighted cells.

3.3. Studied Explanation Types

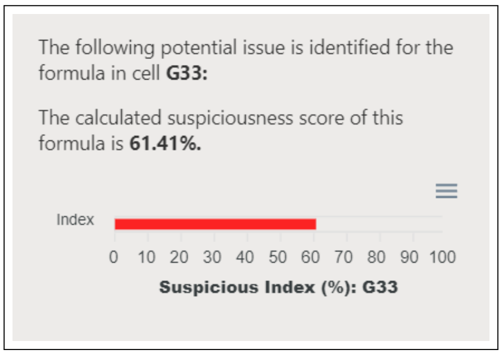
The design space for providing explanations for the outputs of a machine learning based system is generally rich. Typically, there is a multitude of possible ways to explain things to users not only in terms of what *kind of content* is shown, but also in terms of *how* it is presented, e.g., in the form of text or as a visualization.[7]

Overview of Explanation Types. In our study, we investigated the effects of *three* different ways of explaining to the study participants why a cell is highlighted, plus a blackbox approach. Since we adopt a between-subjects design, these forms of explanations correspond to the treatment and the control group.

1. Blackbox: In this condition, the debugging tool acts as a black box and only displays a suspiciousness score. This is the implementation of SmellChecker as reported in [7], and no information is provided by the tool in terms of how this score is computed or how it should be interpreted, cf. Figure 2 and Figure 3a.

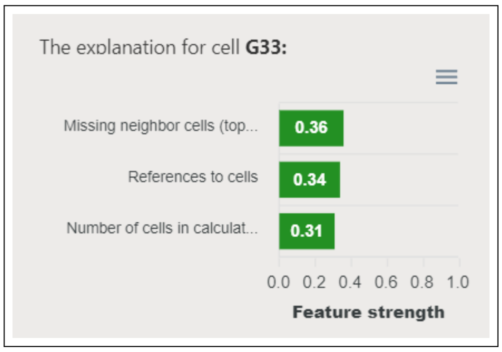
2. Visualization: This form of explanation is based on the visualization of *feature importance* values. The machine learning approach underlying SmellChecker is based on a set of 64 metrics that are derived from the structure of a given spreadsheet and which serve as predictor variables for the fault probability of a given target cell, see [7]. To determine the feature importance values, we rely on the widely used LIME (Local Interpretable Model-agnostic Explanations) [47] approach; see below for additional details. Figure 3b shows how the feature importance values of the most relevant features (as per LIME) are visualized in a diagram.

[7]See for example [46] for a taxonomy of possible ways of explaining the outputs of a machine learning system in the context of decision support systems and recommender systems.

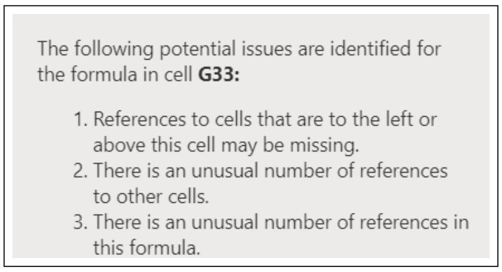

The following potential issue is identified for the formula in cell **G33:**

The calculated suspiciousness score of this formula is **61.41%.**

(a) Black Box

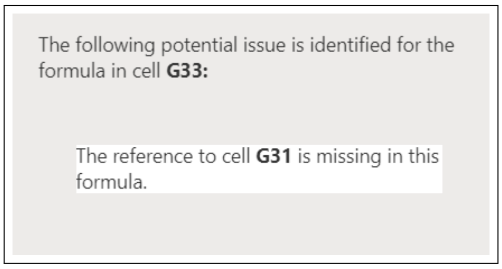

The explanation for cell **G33:**

(b) Visualization


The following potential issues are identified for the formula in cell **G33:**

1. References to cells that are to the left or above this cell may be missing.
2. There is an unusual number of references to other cells.
3. There is an unusual number of references in this formula.

(c) Natural Language Text


The following potential issue is identified for the formula in cell **G33:**

The reference to cell **G31** is missing in this formula.

(d) Human Explanation

Figure 3: Provided explanations

3. NaturalLanguage: These explanations are also based on feature importance values determined with LIME. This time, however, we do not display them visually, but show natural language explanations of each feature (predictor variable). To that purpose, we developed corresponding spreadsheet-independent natural language explanations for each predictor variable. An example of such an explanation is shown in Figure 3c.

4. HumanExplanation: Finally, we included an explanation that corresponds to how a knowledgeable human might describe the problem(s) for a given formula. The explanations shown in this condition thus serve as a form of upper limit of the quality of the explanations that could be provided by a tool. We iterate that these explanations were manually developed by us for the particular spreadsheet used in the experiment and that these explanations are not based on or restricted to the top features identified through LIME. An example of such an explanation is provided in Figure 3d.

Design Rationale and Background. Overall, our goal was to cover a diverse set of approaches to explanation, including both textual and visual ones. We consider the BlackBox method as a lower bound ("no explanations"), and the HumanExplanation version represents a knowledgable and precise explanation as provided by a human expert. The lower and upper bounds, therefore, let us assess where the automated explanations are located on this range. Note that the provision of "suspiciousness score", even when this score is not explained to users in detail, might be considered as a form of explanation as well. However, in the context of our research goals, the score does not carry any information that would help participants understand *why* a certain formula might be faulty.

LIME is a recent and very popular technique to explain the prediction of any classifier in terms of its features. The outcome of LIME is a linear model with coefficient weights assigned to each feature, i.e., product metric in our case. Technically, these weights are obtained by slightly tweaking the feature values for a given learning instance in a way that they are still close to the original values, but the prediction of the base model changes.

Ultimately, those features that have little impact on the outcome of the base model after tweaking their values are considered less important. An alternative to LIME would be to use SHAP [48] values. For our work, we, however, found that LIME explanations can be more efficiently computed. Moreover, LIME explanations are particularly helpful to explain one single prediction of a model. Finally, both for the Visualization and NaturalLanguage explanations, we relied on this approach for explaining the predictions of SmellChecker.

To keep the complexity of the Visualization and NaturalLanguage explanations simple, we only considered the three most important features for a given formula according to the LIME model. In the Visualization approach, we presented users with a descriptive name of the feature and a bar chart that illustrated the value of the corresponding coefficient weight. For the NaturalLanguage explanation, we displayed natural language descriptions of the top three features to the users. These descriptions were manually curated by us for the purpose of the study. Note that these descriptions characterize the respective product metric *in general*, and do not rely on specifics of the given spreadsheet, i.e., they can be used for any spreadsheet. The final descriptions used in our study are based on the descriptions of the 64 product metrics proposed earlier in [7]. The HumanExplanation descriptions, in contrast, are specific to the given spreadsheet and *exactly* explain the fault in a given formula, e.g., "*The reference to cell* ***G31*** *is missing in this formula.*". The exact wording of all relevant explanations can be found in Section 3.5, where we provide all relevant study material.

An important aspect to note here is that while the HumanExplanation explanations were *exact and accurate* (by design), this is not necessarily the case for the automated explanations based on LIME. In particular, none of our study's top three features always point to the actual problem. In other words, the study participants were sometimes confronted with explanations, which could also contain elements that were actually not required or pointing in the right direction. This is important to mention because, in real-world applications, there is no guarantee that LIME explanations are always exact.

3.4. Measurement Method

In this subsection, we discuss the measurement method used in our study in detail.

3.4.1. Measuring Fault Comprehension

To answer *RQ-1* on the effects of explanations on the participants' comprehension of the faults, we derived an average *comprehension score* based on a qualitative analysis of the plain text explanations provided by the participants for each of the three faults that were indicated by the tool. To avoid any bias, a scoring scheme was defined in advance, where individual participant responses could receive scores between 0 and 2. The scoring scheme is shown in Table 1.

Two of the authors scored all responses of the participants independently, and they did not know to which treatment or control group a particular response belonged to. On average, the two scorers largely agreed in terms of their assessments, leading to a high intraclass correlation coefficient. The researchers then inspected the explanations where they initially disagreed and then decided together on the final score.

We note that some participants in the HumanExplanation condition may have been tempted to copy and paste the provided (correct) explanation into their response, which would make it unclear if they truly understood the underlying problem. A manual inspection of the provided responses, however, revealed no case where the system's response was simply copied. Instead, participants circumscribed the problem in their own words. We observed the same for the participants in the NaturalLanguage group.

3.4.2. Measuring Subjective Perceptions

To assess the subjective quality perceptions of the different explanations (*RQ-2)*, we designed a questionnaire that relied on instruments that were used earlier in the literature. First, we used the System Usability Scale (SUS) to assess the usability of the overall

Faulty Cell	Score	Rule
G33	0	Participant states that there is no fault or provides factually wrong explanation
	1	Participant states that a reference is missing
	2	Participant states that cell G31 is missing in the sum *OR* Participant states that the formula should be =SUM(G8:G31)
I25	0	Participant states that there is no fault or provides factually wrong explanation
	1	Participant states that the IF is not necessary *OR* Participant states that cell G25 should be referenced instead of cell G24
	2	Participants states that the formula should be consistent with the formulas in the same column *OR* Participants states that the formula should be =G25*H25
I34	0	Participant states that there is no fault or provides factually wrong explanation
	1	Participant states that a reference is missing
	2	Participant states that cell I8 is missing in the sum *OR* Participant states that the formula should be =SUM(I8:G31)

Table 1: Scoring method for the explanations provided by the participants

SmellChecker tool. In addition, we took inspiration from the Technology Acceptance Model (TAM) [49] to assess the participants' beliefs, attitudes, and behavioral intentions, both with respect to the explanations and to the tool as a whole. Specifically, we were interested in the system's perceived *usefulness*, *ease-of-use*, *transparency and control*, *trust*, *satisfaction* and their *intentions to reuse or recommend* the system in the future. The detailed questionnaire items are shown in Table 3 below. In addition to these questions, we asked participants about demographics and their expertise with spreadsheets. Finally, participants could leave free-text comments before finalizing the questionnaire.

3.5. Study Materials

In this section, we provide additional information with respect to materials used in our experiment.

Faulty Spreadsheet. Selecting a suitable spreadsheet for the purpose of the study is a challenging task. On the one hand, the spreadsheet must not be too complex, because participants must be able to understand or at least properly guess the intended semantics of the calculations within a short period of time. On the other hand, a too simple spreadsheet might not be representative for real-world situations, see also for related discussions in [38]. After a thorough analysis of existing spreadsheet corpora, we selected a real-world spreadsheet from the Enron Error corpus [50] (see Figure 2) because of its size, formula complexity, and the faults it contains.

The spreadsheet originally contained nine faulty formula cells. The cells in the range I25:I31 contained seven identical faults. The other cells with faulty formulas were G33 and I34. The SmellChecker tool successfully located all faults of the spreadsheet. However, to keep the user study manageable, we corrected all but one of the identical faults, which left us with three faulty cells:

- A range error in the formula in cell G33, where we have `=SUM(G8:G30)` instead of `=SUM(G8:G31)`,
- a double (range/logic) fault in cell I25, where we have `=IF(H25;G24*H25;0)` instead of `=G25*H25`, and

- another range error in the formula of cell I34, with `=SUM(I9:I31)` instead of `=SUM(I8:I31)`.

Natural Language Explanations and Post-Task Questionnaire. The exact phrasing of the natural language explanations (NaturalLanguage) and the human explanations (HumanExplanation) for the three faulty cells of the spreadsheet used in the study are given in Table 2. Table 3 lists the exact questions that were asked to participants in the post-task questionnaire.

Faulty Cell	**Predictive Product Metric**	**Natural Language Explanation**	**Human Explanation**
G33	Missing neighbor cells (top/left)	References to cells that are to the left or above this cell may be missing.	The reference to cell G31 is missing in this formula.
	References to cells	There is an unusual number of references to other cells.	
	Number of cells in calculation	There is an unusual number of references in this formula.	
I25	IF formula complexity	The IF statement is either not required or it is too complex.	The IF statement is not required in this formula and the reference to cell G24 is incorrect.
	Same type cell distance (row)	This row contains both formula cells and data cells.	
	References from other cells	There is an unusual number of references to this cell.	
I34	References to cells	There is an unusual number of references to other cells.	The reference to cell I8 is missing in this formula.
	Missing neighbor cells (top/left)	References to cells that are to the left or above this cell may be missing.	
	Number of cells in calculation	There is an unusual number of references in this formula.	

Table 2: Top three product metrics for faulty cells.

3.6. Participants

We recruited the participants for our study through the crowdsourcing platform Prolific[8]. Through an initial screening process, we only considered participants with a certain level of experience with spreadsheets, e.g., who could write formulas and manage data with spreadsheets. Furthermore, during the study, we asked the participants about their experience with spreadsheets. The statistics showed that more than half of the participants declared to have more than 7 years of experience with spreadsheets, more than half of the participants use spreadsheets daily, and less than 7% considered them as "beginners" concerning their experience. The detailed distribution of the participants' responses in terms of "years of experience" are shown in Figure 4. We believe the participants have sufficient expertise in using spreadsheets and represent at least some common class of spreadsheet users in practice.

4. Results

In this section, we first analyze to what extent explanations can assist developers in understanding the faults of a spreadsheet (Section 4.1 on *RQ-1*) and then study the subjective quality perceptions of the participants in more depth (Section 4.2 on *RQ-2*).

4.1. Fault Comprehension (RQ-1)

Fault Comprehension Scores. We recall that we assess the fault comprehension level of participants through a manual inspection and scoring procedure by two independent assessors.

ID	Question and answer options
Demographics	
D1	How often do you use spreadsheets with formulas in your daily work? *Daily / Once in a week / Once in a month / Rarely / Don't use at all*
D2	How many years have you been using spreadsheets in your daily work? *Less than a year / 1-3 years / 4-7 years / More than 7 years*
D3	I rate my expertise with spreadsheets as follows: *Beginner / Intermediate / Advanced / Expert*
D4	How old are you? *18-24 years / 25-29 years / 30-34 years / 35-39 years / 40 and over*
D5	What is your gender? *Male / Female / Prefer not to say*
All answers are on a 1-5 Likert scale (1=strongly disagree; 5=strongly agree)	
User Beliefs: Perceived Usefulness	
U1	The explanations provided by SmellChecker helped me understand why the marked cells were suspicious.
U2	I could not understand the explanations provided by SmellChecker.
U3	The explanations provided by SmellChecker were distracting.
U4	The explanations provided by SmellChecker allowed me to accomplish the task more quickly.
User Beliefs: Perceived Ease of Use	
E1	I would need an expert opinion to understand the explanations provided by SmellChecker.
E2	The interaction with SmellChecker was clear and understandable.
E3	The explanations provided by SmellChecker were easy to understand.
E4	It took too much effort to understand the explanations provided by SmellChecker.
User Beliefs: Control	
C1	I felt in control of the various functionalities of SmellChecker.
User Beliefs: Transparency	
T1	I understand why the cells were marked suspicious through the explanations.
All answers are on a 1-5 Likert scale (1=strongly disagree; 5=strongly agree)	
User Attitudes: Trust and Confidence	
TC1	The explanations provided by SmellChecker are trustworthy.
TC2	Overall, I trust SmellChecker.
User Attitudes: Satisfaction	
S1	I needed to learn a lot of things before I could get going with SmellChecker.
S2	My overall satisfaction with SmellChecker is high.
All answers are on a 1-5 Likert scale (1=strongly disagree; 5=strongly agree)	
Behavioral Intentions: Recommendation Intentions	
R1	I would recommend SmellChecker to my friends or colleagues who use spreadsheet tools.
Behavioral Intentions: Intention to use	
I1	I would like to use SmellChecker again because I found the explanations helpful.
I2	I could not understand the explanations, so I would not use SmellChecker again.
All answers are on a 1-5 Likert scale (1=strongly disagree; 5=strongly agree)	
System Usability	
SU1	I would like to use SmellChecker frequently.
SU2	I found SmellChecker unnecessarily complex.
SU3	I thought SmellChecker was easy to use.
SU4	I would need the support of a technical person to be able to use SmellChecker.
SU5	I found the various functions in SmellChecker were well integrated.
SU6	There was too much inconsistency in SmellChecker.
SU7	I would imagine that most people would learn to use SmellChecker very quickly.
SU8	I found SmellChecker very cumbersome to use.
SU9	I felt very confident using SmellChecker.
SU10	I needed to learn a lot of things before I could get going with SmellChecker.
All answers are free text	
Comments and Suggestions	
CS1	What do you think about the explanation(s) given for the suspicious cells?
CS2	Write your comments and suggestions about SmellChecker in general here.

Table 3: Post-Task Questionnaire Items

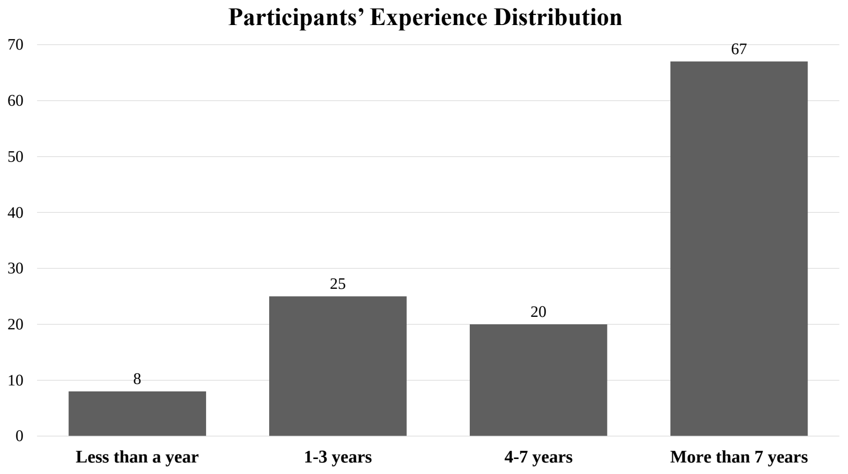


Figure 4: Distribution of Years of Experience of the Participants

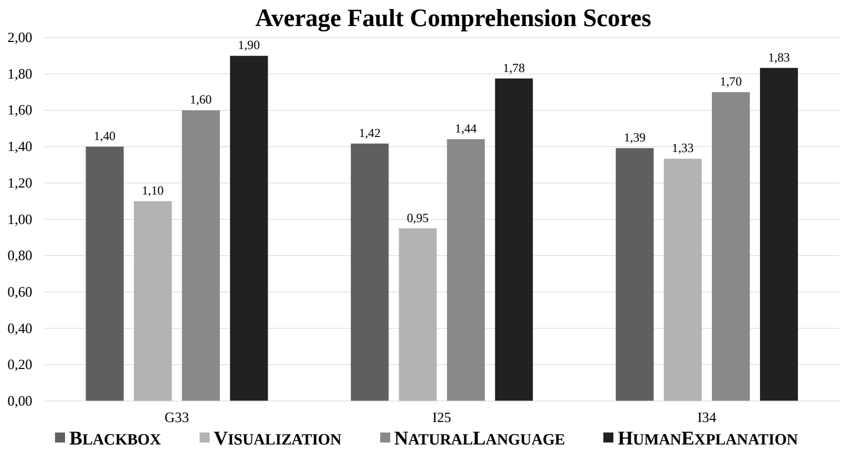


Figure 5: Average Fault Comprehension Score

The outcomes of this scoring process for each faulty cell and participant group are shown in Figure 5.

We observe that the general pattern is consistent across the three faulty cells. The highest level of fault comprehension was achieved by the participants of the HUMANEXPLANATION group. This is expected, as the explanations for this group were created manually and should serve as an upper bound in terms of fault comprehension. The average comprehension scores on our 0-2 scale for the three cells ranged from 1.78 to 1.90, with an average of 1.83. The lowest score of 1.78 was achieved for cell I25, which contained the most complex fault.

The second highest comprehension level was achieved by participants in the NATURALLANGUAGE group, with an average of 1.58 across all faults. Looking at the detailed scoring results, we found that participants in this group fully comprehended (and properly described) more than half of the faulty cells, and they showed a reasonable understanding of the problems in the other cells.

Interestingly, the visualization-based explanations, which are based on LIME outputs like NATURALLANGUAGE, did not work too well in our study. In fact, on average the fault comprehension level for this treatment group was even slightly lower than for the

[8]`https://www.prolific.co/`, participants were paid 7£ for participating in the study.

Blackbox group, where participants were only shown the overall suspiciousness score. This may indicate that the chosen visualization was too difficult to interpret for the participants, or they did not contain sufficient information for the participants. On an absolute scale, the average comprehension level was at 1.4 for the Blackbox explanations, and 1.12 for the Visualization approach. Again, the lowest scores were obtained for the most complex fault in each group. Overall, the higher average score for the Blackbox explanations might indicate that the visualization may have in fact distracted the participants, and that the participants in the Blackbox group could better focus on understanding the problems in the faulty cells. Looking at individual responses by participants in the Visualization group, we could observe that they in a number of cases interpreted the feature importance values as shown in the visualizations in a wrong way. The NaturalLanguage explanation approach, in contrast, seems to be indeed helpful to participants in terms of fault comprehension.

The distribution of the comprehension scores for the individual cells and participant groups are shown in Figure 6 in the form of box plots. An ANOVA analysis revealed significance differences between the participant groups for each cell.[9] We, therefore, applied Tukey's HSD post-hoc tests with Bonferroni correction.[10] The tests revealed a number of significant differences between the participant groups, which we summarize in Table 4.

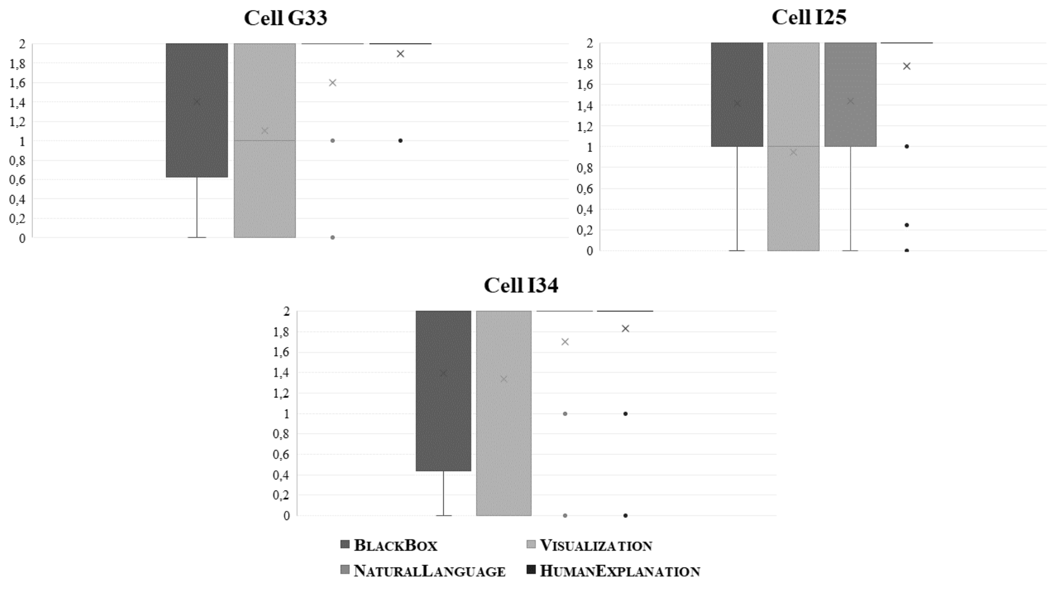


Figure 6: Box plot (Fault Comprehension Score)

The extent of the differences in the mean comprehension scores vary across the faulty formulas. The difference between the "optimal" human explanations and the visualization-based approach are at least moderately significant in all cases. In the case of the most complex fault in cell I25, it turned out that also the automatically generated NaturalLanguage approach was significantly better than the Visualization technique.

Overall, answering ***RQ-1***, we see our results as *clear indications that well-designed explanations can be a suitable tool for improved fault comprehension in the spreadsheet debugging process.* It also turns out that the usefulness of explanations depends on their particular implementation and there are indications that it may also depend on the complexity and nature of the faults. In our specific comparison, we found that showing natural language descriptions for important predictor variables in the NaturalLanguage approach is promising.

[9]The p-values were below 0.001 for cell G33 and cell I25, and at 0.03 for cell I34.

[10]Bonferroni correction is done to account for the multiple comparisons. After applying the correction, the α level to reject the null hypothesis is lowered from 0.05 to 0.0083. An α level of 0.1 is lowered to 0.016.

A widely used visualization approach (Visualization), on the other hand, actually can even have slightly detrimental effects compared to a situation where the system does not provide specific explanations at all.

	G33	I25	I34
	HumanExplanation > Blackbox **	–	–
	HumanExplanation > Visualization **	HumanExplanation > Visualization **	HumanExplanation > Visualization *
	–	NaturalLanguage > Visualization **	–

Table 4: Overview of significant differences in terms of fault comprehension. ** indicates significance at α=0.05, * significance at α=0.1.

Feedback Analysis. Given the observations regarding the participants' level of fault comprehension as just discussed, we examined if the free-form feedback on the explanations—from questionnaire item CS1[11] in Table 3—aligns with these general tendencies. To analyze this aspect, we used Inductive Coding [51] as a technique to extract concepts or themes from a body of text in a structured way. During this process, the coders were not aware to which participant group a given free text belonged to. The identified concepts are shown in Table 5. To obtain a broader picture of the general assessment of the participants of the explanations we then further categorized the concept as having either a positive, negative, or neutral sentiment.

Sentiment	**Concepts**
Positive	Useful
	Helpful
	Understandable
	Good/Great
	Recommend
Negative	Complicated/Cumbersome
	Dislike
	Confusing
	Unhelpful
	Needs improvement
Neutral	General remarks

Table 5: Concepts Categorization

We then counted the occurrence of each concept in the free-form feedback text. The frequency distribution of the concept sentiments for the groups is presented in Figure 7.

We clearly observe that the HumanExplanation and NaturalLanguage groups provided largely positive feedback on the explanations. The participants who saw the visualization-based explanations where less positive, but still provided positive feedback more often than the participants who only saw the suspiciousness scores in the Blackbox group. However, participants in the Visualization group also frequently reported negative feedback. This supports our observations from the fault comprehension analysis that participants often may have had difficulties in interpreting the visualization.[12] Overall, this qualitative analysis of the free-form feedback seems largely consistent with the fault comprehension analyses reported above, where natural language explanations generally had a positive effect.

[11]The exact question read: "*What do you think about the explanation(s) given for the suspicious cells?*"

[12]Figure 7 shows a small set of negative feedback also for the HumanExplanation condition. An inspection of these cases revealed that all but one of the few negative comments did not refer to the explanations, but to other aspects like the complexity of the installation of the plug-in. The one remaining comment with a negative sentiment mentioned that *"a step-by-step guide, particularly on I25, would have been useful."*

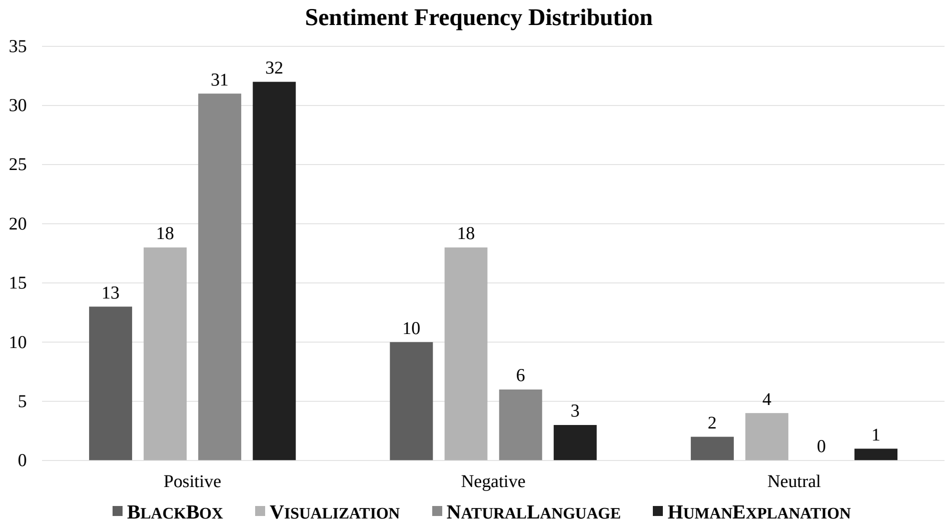


Figure 7: Sentiment Frequency Distribution

4.2. Subjective Perceptions (RQ-2)

We recall that the study participants were asked to answer in-depth questionnaires after completing the main task of analyzing potential faults. We evaluated four dimensions of perceptions, namely *Tool Usability*, *User Attitudes*, *User Beliefs*, and *Behavioral Intentions.* Our analyses involve a larger number of comparisons and significance tests, and we share all detailed outcomes in an online repository[13]. For the sake of brevity, we limit our discussions here to the general trends and insights gathered through our user study.

Tool Usability. Participants answered a 10-item questionnaire from the System Usability Scale (SUS), using a 1-to-5 scale. The range of the overall SUS score for a system lies between 0 and 100, which is obtained by normalizing the response values of the questionnaire items. For this purpose, value 1 is subtracted from each positively framed question's score, and each negatively framed question's score is subtracted from value 5. Finally, the sum of these scores is multiplied by 2.5 to normalize the scores between 0 and 100. The overall SUS scores for the different participant groups are shown in Table 6.

Treatment Group	SUS score
HUMANEXPLANATION	82.25
NATURALLANGUAGE	79.50
VISUALIZATION	68.03
BLACKBOX	67.26

Table 6: Overall System Usability Scale (SUS) scores. Values larger than 68 are considered above average ("good").

Overall, we observe that all versions of the tool obtained overall scores that are close or markedly higher than the general threshold of 68. Systems that score higher than this value are usually considered above average or "good". The scores that were obtained for the tool versions that provided HUMANEXPLANATION and NATURALLANGUAGE explanations are much higher than for the other types of explanations. This is in line with our findings reported above with respect to fault comprehension, and these results provide strong indications that these two types of explanations can strongly contribute to the usability of the

[13]`https://bit.ly/3QbVA6O`

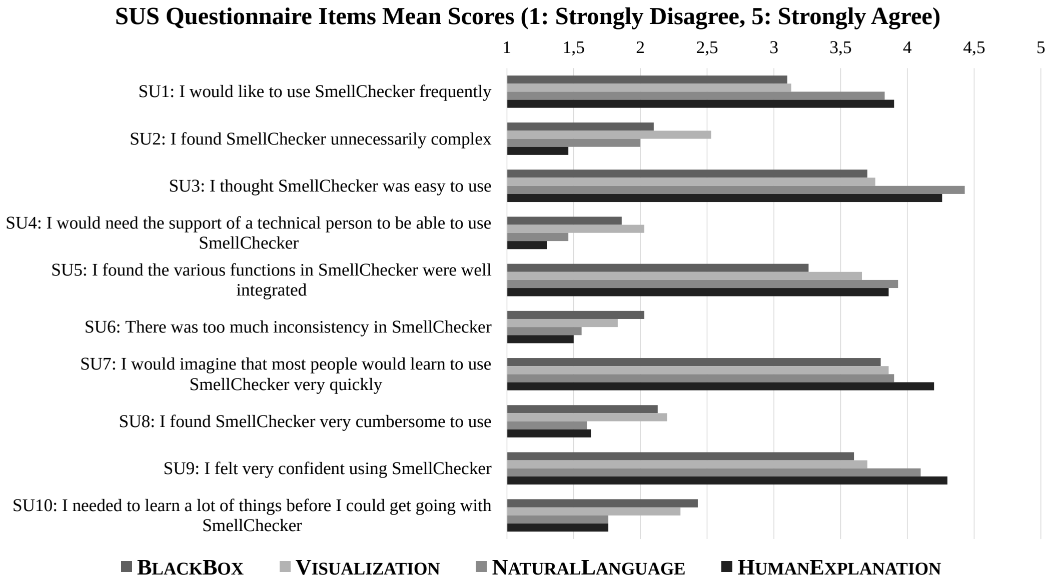


Figure 8: Mean scores of the SUS questionnaire items

SmellChecker tool as a whole. We acknowledge that the observed differences may be particularly pronounced in our study, where the participants had to verbalize the potential reasons for a suspected fault as a main task when interacting with the tool.

The mean responses for each of the items of the SUS scale are shown in Figure 8. Aligned with the overall scores, we find that the assessments by participants in the HumanExplanation and NaturalLanguage groups were consistently better than those for the other two participant groups. While in some cases the differences are not too large, there are stronger differences for some of the questions. Looking at item **SU2**, for example, we observe that the Visualization group found the SmellChecker tool markedly more complex than the other groups. This effect can apparently be attributed to the design of the explanations, because nothing else was changed across the groups.

User Beliefs. The average responses to the questionnaire items relating to *user beliefs* are presented in Figure 9.[14] In general, the participants in the HumanExplanation group reported the best values for various aspects covered in this questionnaire, followed by the participants in the NaturalLanguage group. Looking at some of the details, we for example find that participants in the HumanExplanation and NaturalLanguage groups found the explanations to be more *useful*, more *understandable* and less *distracting* than participants in the other groups. Conversely, participants in the Visualization group often seemed to have difficulties understanding the explanations. Again, this is aligned with the other observations that we reported so far.

An interesting observation can be made of item *T1*, where participants had to inform to what extent they understood why the tool marked a cell as being potentially faulty. While the participants in the Blackbox group gave lower scores than participants in other groups, their average response was not at the very low end. This is to some extent unexpected because the tool in that condition did not provide any information about the possible reasons for a cell being suspiciousness other than an overall suspiciousness score. Thus, we suspect that participants in the Blackbox group may have interpreted the question at a different level. Their reasoning might have been that they understood that the cell was marked

[14]The mean participant responses and standard deviations for user beliefs, attitudes, and behavioral intentions can also be found in Table 7 in the Appendix.

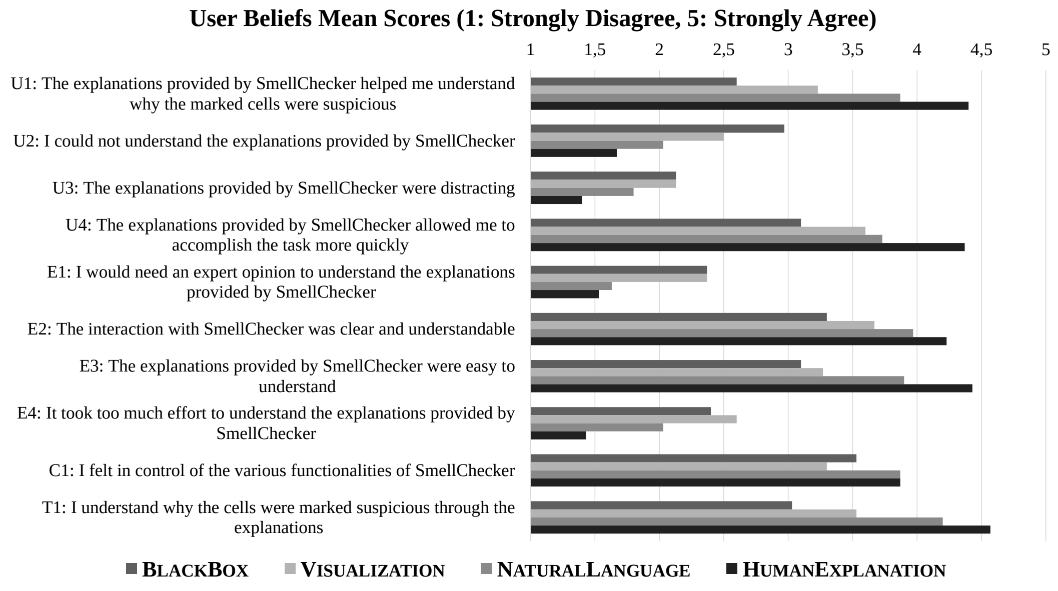


Figure 9: User Beliefs: Perceived Usefulness (U1-U4), Perceived Ease of Use (E1-E4), Control (C1), Transparency (T1)

because it had a high suspiciousness score, i.e., they might not have considered reasons that relate to the formula in the cell itself.

User Attitudes. In terms of user attitudes, we asked participants about their trust and confidence[15] with respect to the explanations and about their overall satisfaction. The results for this part of the questionnaire are shown in Figure 10.

We find that participants in the HumanExplanation group consistently reported the best values in terms of these factors, i.e., the participants in the HumanExplanation group found SmellChecker and the explanations more trustworthy and their overall satisfaction is slightly higher compared to the other groups. The average responses of the NaturalLanguage group for trustworthiness and satisfaction were also encouraging, e.g., they are around 4 for the positively framed question items (TC1, TC2 and S2). The lowest values were consistently observed for the Blackbox group. On an absolute scale, these values were also not too bad, and participants in this group were still satisfied to a certain extent with the overall tool.

An interesting observation can be made with respect to the Visualization group. Even though the level of fault comprehension was rather low for this group, and even though participants found these explanations to be rather complex and more difficult to understand (see *User Beliefs*), they reported the same level of trust as the participants in the NaturalLanguage group. This may have been caused by the fact that participants in these groups were provided information about the same set of features, albeit in a different form. We may also speculate that the more formal, numeric explanation in the Visualization group may have led participants to attribute similar levels of trust, even though the visualization was more complex and less understandable than the NaturalLanguage explanations.

[15]We acknowledge that in general trust towards a software application is a complex and multi-faceted concept, and may be based on various factors such as multiple positive past experiences with a given tool or similar tools over time, or from the credibility of the provider of the application. In our study, we assess the participants' trust (or: perceived trustworthiness of the system) after one single experience. This may represent a certain limitation of our present study and earlier works in this area; see [52] for a meta-analysis of the impact of trust in the Technology Acceptance Model.

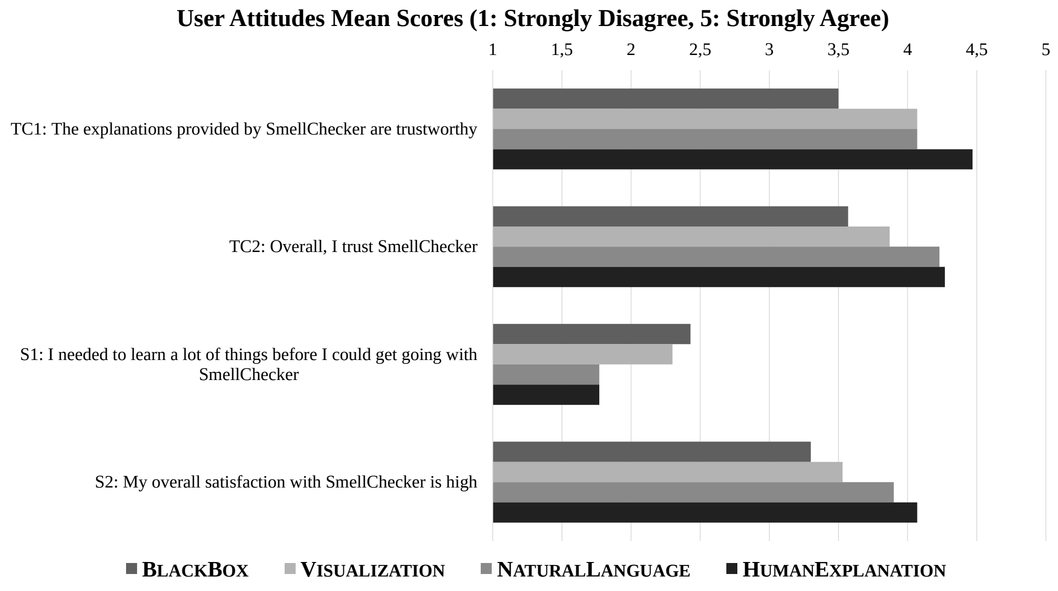


Figure 10: User Attitudes: Trust and Confidence (TC1-TC2), Satisfaction (S1-S2)

Behavioral Intentions. We report the responses of the questionnaire items relating to *behavioral intentions* (to reuse the tool and to recommend it to others) in Figure 11.

The observations so far made in terms of user beliefs and user attitudes are clearly reflected in the responses to these final questions. Participants in the HumanExplanation condition expressed the highest level of intentions to use the tool in the future and to recommend it. The NaturalLanguage explanations led to slightly lower positive intentions, followed by the Visualization and the Blackbox explanations. Overall, we observe a positive effect of adding understandable explanations both in terms to the tool as a whole (R1) and in terms of the contribution of the explanations to the future intentions (I1, I2).

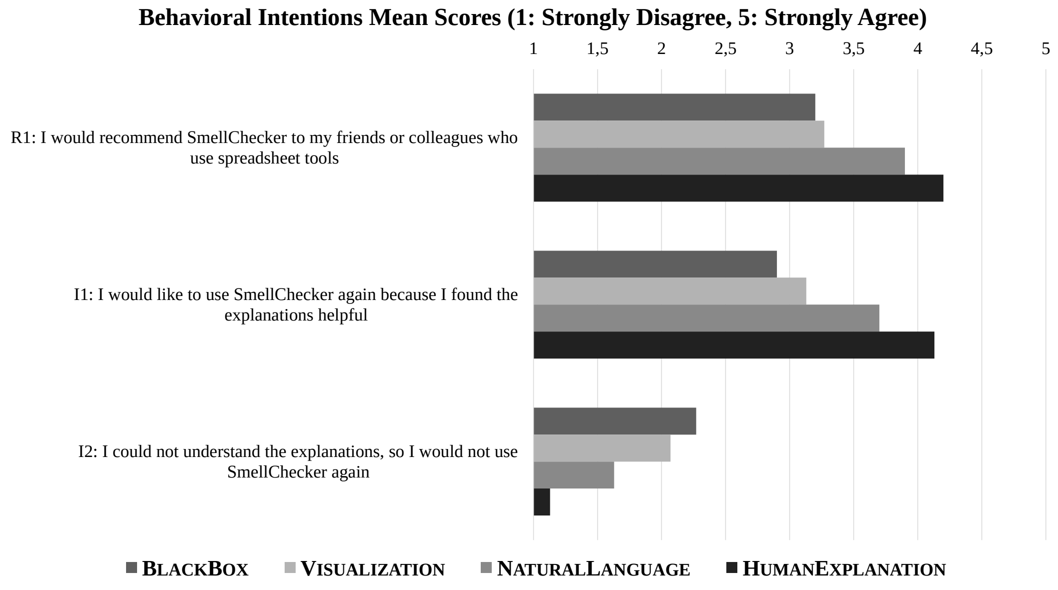


Figure 11: Behavioral Intentions: Recommendation Intentions (R1), Intentions to Use (I1, I2)

Overall, answering ***RQ-2***, our results indicate that *well-designed explanations can help* *to improve various quality perceptions and the trust towards a spreadsheet debugging tool.*

5. Implications & Threats to Validity

5.1. Implications

Our research has a number of important practical implications. With respect to *RQ-1*, the main insight of our study is that providing explanations in the context of automated fault prediction support can indeed help spreadsheet users improve their understanding of the potential faults that are highlighted by such a tool. Thus, designers of fault prediction tools should consider enhancing their tools by a corresponding functionality, since an increased level of fault comprehension will naturally be beneficial for the subsequent fault removal step.

Second, and also importantly, we found that the design of the explanation matters a lot. The most effective explanations in our study were manually engineered ones written in natural language, which we considered as an upper bound in our experiments. The natural language explanations, which were automatically selected based on the widely-used LIME approach, however, turned out to be promising as well. The approach requires some initial manual effort, but this effort is very limited as it amounts to merely creating one natural language sentence for each predictor variable (feature) of the underlying supervised learning problem. We recall that these descriptions are not tied to a specific spreadsheet or application domain. Thus, these explanations can be crafted once and then shipped with the debugging tool. In the context of design choices, we furthermore found that some forms of explanations might not be effective at all or even have a slightly detrimental effect on fault comprehension. This may happen when the explanation is not easy to comprehend. In our study, we found indications that a popular form of visualizations in the literate based on bar charts may even misguide spreadsheet users.

Given these observations in the context of *RQ-1*, we conclude that any form of explanations that are presented to users must be carefully evaluated in terms of their true usefulness with studies involving humans. In our case, this evaluation was done in two ways: *(a)* by manually scoring the participants' level of fault comprehension in a quantitative way and *(b)* by qualitatively analyzing the participants' sentiment towards the provided explanations as expressed in the free-form feedback statements. The observations for both types of analysis were well aligned in our study. Generally, relying on more than one form of evaluation procedure is advisable in such forms of human-centered research to obtain robust results.

Our analyses in the context of *RQ-2* highlight that explanations in the spreadsheet domain may not only serve the primary target of improving the participants' understanding of potential faults. We found that well-designed explanations lead to better quality perceptions in terms of usability and usefulness, and that such explanations may help to increase the users' trust in the debugging system as a whole and, consequently, their intentions to use the tool or to recommend it to others. These findings underline that there are several potential advantages of providing explanations when highlighting parts of a program to be as potentially faulty.

5.2. Threats to Validity

Our work comes not without limitations. First, our study was based on one particular spreadsheet and on one given set of faults.[16] Thus, it remains to study if we would observe similar effects for spreadsheets that are largely different from the one used in our experiment. Nonetheless, we believe that the spreadsheet used in our study and the faults it contains are representative of a larger set of spreadsheets that can be found in practice. The used spreadsheet in fact is a real-world one and all the faults in it were also "natural". Also, in

[16]Also, certain *types* of faults, e.g., when users entered a constant value instead of a formula in a cell, were not investigated yet in our present study.

terms of the number and complexity of the formulas, there are many spreadsheets in the Enron error corpus [53] of real-world spreadsheets that are very similar to our spreadsheet.

Generally, our work shares the potential limitations of any user study in which study participants are completing tasks in an artificial environment and setting. In terms of the *realism* of the study, we are confident that our setting was perceived to be quite natural by participants, given their feedback on the general usefulness of the tool and their intentions to use such a tool in the future. In terms of study participants, we relied on crowdworkers. Since most of these crowdworkers indicated that they had at least seven years of experience with spreadsheets, we argue that the study participants are representative for at least some part of the typical population of spreadsheet users in practice. Furthermore, to ensure that our results are not based on the work of potentially inattentive and unmotivated crowdworkers, we *(a)* only admitted crowdworkers with a positive track record to the study and *(b)* implemented attention checks to only include the responses of careful crowdworkers in our analyses.

Furthermore, our study was so far limited to four conditions, where in two of them the explanations were automatically selected and/or visualized based on a widely-used technique from the field of explainable AI. Other forms of explaining the fault predictions are certainly possible and are part of our future work. However, we recall here that the goal of our study was not to find the "best" form of explanations but to take the first step towards an understanding of the general usefulness of providing explanations for automated fault predictions. Moreover, in our experiment, we assumed that there were no "false positives", i.e., that the tool only marked formulas as suspicious, which were faulty. While avoiding such false alarms is generally a problem of the underlying fault prediction method, it may be interesting to study how different forms of explanations affect spreadsheet users when there is no fault. For example, one hypothesis could be that more understandable explanations are also helpful for users to detect that the highlighted cell is correct. But, it may also be that users overly rely on the predictions and explanations by the tool and may even start modifying a correct formula, see [38] for a study on tool over-reliance in the context of spreadsheets.

6. Summary and Outlook

In a widely cited study, Parnin and Orso [4] questioned if certain automated debugging tools from the literature are *really* helping developers. They concluded that this might not always be the case and that this may be caused by certain assumptions that are made by researchers which may not hold in practice, e.g., how developers inspect lists of potentially faulty program elements. As a consequence, the authors call for more research that study questions of *fault comprehension* and, therefore, for more research that is based on studies with users.

In our research, we have addressed the question of the usefulness of a recent spreadsheet fault prediction technique in this spirit for the class of spreadsheet programs. In particular, we have investigated to what extent it may be beneficial to explain the tool's fault prediction to users. Overall, we find strong indications that providing explanations for the outcomes of an underlying machine learning model can indeed improve the usefulness and quality perception of the fault localization tool. To our knowledge, this is the first work in the area of spreadsheet debugging, and we found also very limited works that address the role of explanations for fault prediction techniques for general software. More research is therefore required to understand if the insights from our experiment in the spreadsheet domain generalize for other types of software programs. Nonetheless, in a broader perspective, we see our work also a contribution to the area of explainable AI, which follows the assumption that a wider and adoption of AI-fueled technology might often depend on a system's capability of explaining its decisions or recommendations.

In terms of future works, we plan to investigate alternative ways of explaining the system's fault predictions to users, including in particular other forms of visualizations, given that visualizations have proven a promising means to support spreadsheet debugging processes in the past, see [54]. Moreover, besides validating our findings with other spreadsheets and to analyze to what extent the effectiveness of explanations is correlated with the complexity of the faults, it remains important to study the usefulness of explanations when alternative tools and underlying techniques are used for fault prediction, e.g., based on spectrum-based fault localization or on model-based reasoning.

Acknowledgement

The work described in this paper is funded by the Austrian Science Fund (FWF) project Interactive Spreadsheet Debugging under contract number P 32445.

References

[1] C. Catal, Software fault prediction: A literature review and current trends, Expert Systems with Applications 38 (4) (2011) 4626–4636.

[2] P. Koch, K. Schekotihin, D. Jannach, B. Hofer, F. Wotawa, Metric-based fault prediction for spreadsheets, IEEE Transactions on Software Engineering (2019) 1–1.

[3] N. Miryeganeh, S. Hashtroudi, H. Hemmati, GloBug: Using global data in Fault Localization, Journal of Systems and Software 177 (2021) 110961.

[4] C. Parnin, A. Orso, Are automated debugging techniques actually helping programmers?, in: 2011 International Symposium on Software Testing and Analysis, 2011, pp. 199–209.

[5] A. Ang, A. Perez, A. Van Deursen, R. Abreu, Revisiting the practical use of automated software fault localization techniques, in: 2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), 2017, pp. 175–182.

[6] Z. Wan, X. Xia, A. E. Hassan, D. Lo, J. Yin, X. Yang, Perceptions, expectations, and challenges in defect prediction, IEEE Transactions on Software Engineering 46 (11) (2020) 1241–1266.

[7] P. Koch, K. Schekotihin, D. Jannach, B. Hofer, F. Wotawa, Metric-based fault prediction for spreadsheets, IEEE Transactions on Software Engineering (2019) 1–1.

[8] R. R. Panko, What we know about spreadsheet errors, Journal of Organizational and End User Computing 10 (2) (1998) 15–21.

[9] R. R. Panko, Revisiting the Panko-Halverson Taxonomy of Spreadsheet Errors, in: Proc. European Spreadsheet Risks Int. Grp. (EuSpRIG) 2008, 2008, pp. 199–220.

[10] A. Barredo Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcia, S. Gil-Lopez, D. Molina, R. Benjamins, R. Chatila, F. Herrera, Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai, Information Fusion 58 (2020) 82–115.

[11] S. Omri, C. Sinz, Deep learning for software defect prediction: A survey, in: Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops, 2020, p. 209–214.

[12] W. E. Wong, R. Gao, Y. Li, R. Abreu, F. Wotawa, A survey on software fault localization, IEEE Transactions on Software Engineering 42 (8) (2016) 707–740.

[13] M. Ducasse, A pragmatic survey of automated debugging, in: International Workshop on Automated and Algorithmic Debugging (AADEBUG), Springer, 1993, pp. 1–15, Lecture Notes in Computer Science (LNCS), volume 749.

[14] M. Shepperd, D. Bowes, T. Hall, Researcher bias: The use of machine learning in software defect prediction, IEEE Transactions on Software Engineering 40 (6) (2014) 603–616.

[15] C. Manjula, L. Florence, Deep neural network based hybrid approach for software defect prediction using software metrics, Cluster Computing 22 (2019) 9847–9863.

[16] A. Okutan, O. Yildiz, Software defect prediction using bayesian networks, Empirical Software Engineering 19 (2014) 154–181.

[17] G. Czibula, Z. Marian, I. G. Czibula, Software defect prediction using relational association rule mining, Information Sciences 264 (2014) 260–278, serious Games.

[18] D. Radjenović, M. Heričko, R. Torkar, A. Živkovič, Software fault prediction metrics: A systematic literature review, Information and Software Technology 55 (8) (2013) 1397–1418.

[19] R. S. Wahono, A systematic literature review of software defect prediction: Research trends, datasets, methods and frameworks, Journal of Software Engineering 1 (1) (2015) 1–16.

[20] Z. Li, X.-Y. Jing, X. Zhu, Progress on approaches to software defect prediction, IET Software 12 (3) (2018) 161–175.

[21] S. Rathore, S. Kumar, A study on software fault prediction techniques, Artificial Intelligence Review 51 (2019) 255–327.

[22] M. K. Thota, F. H. Shajin, P. Rajesh, Survey on software defect prediction techniques, International Journal of Applied Science and Engineering 17 (2020) 331–344.

[23] R. Malhotra, A systematic review of machine learning techniques for software fault prediction, Applied Soft Computing 27 (2015) 504–518.

[24] N. Li, M. Shepperd, Y. Guo, A systematic review of unsupervised learning techniques for software defect prediction, Information and Software Technology 122 (2020) 106287.

[25] F. Matloob, T. M. Ghazal, N. Taleb, S. Aftab, M. Ahmad, M. A. Khan, S. Abbas, T. R. Soomro, Software defect prediction using ensemble learning: A systematic literature review, IEEE Access 9 (2021) 98754–98771.

[26] B. Hofer, D. Jannach, P. Koch, K. Schekotihin, F. Wotawa, Product metrics for spreadsheets - a systematic review, Journal of Systems and Software 175 (2021) 110910.

[27] M. R. Ahmed, M. A. Ali, N. Ahmed, M. F. B. Zamal, F. J. M. Shamrat, The impact of software fault prediction in real-world application: An automated approach for software engineering, in: Proceedings of 2020 the 6th International Conference on Computing and Data Engineering, Association for Computing Machinery, 2020, p. 247–251.

[28] P. He, B. Li, X. Liu, J. Chen, Y. Ma, An empirical study on software defect prediction with a simplified metric set, Information and Software Technology 59 (2015) 170–190.

[29] G. R. Choudhary, S. Kumar, K. Kumar, A. Mishra, C. Catal, Empirical analysis of change metrics for software fault prediction, Computers & Electrical Engineering 67 (2018) 15–24.

[30] A. J. Ko, B. A. Myers, Designing the whyline: a debugging interface for asking questions about program behavior, in: CHI, ACM, 2004, pp. 151–158.

[31] A. J. Ko, B. A. Myers, Debugging reinvented: Asking and answering why and why not questions about program behavior, in: Proceedings of the 30th International Conference on Software Engineering, ICSE '08, 2008, p. 301–310.

[32] B. Burg, R. Bailey, A. J. Ko, M. D. Ernst, Interactive record/replay for web application debugging, in: Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology, UIST '13, 2013, p. 473–484.

[33] X. Xie, Z. Liu, S. Song, Z. Chen, J. Xuan, B. Xu, Revisit of automatic debugging via human focus-tracking analysis, in: Proceedings of the 38th International Conference on Software Engineering, ICSE '16, Association for Computing Machinery, 2016, p. 808–819.

[34] X. Xia, L. Bao, D. Lo, S. Li, "Automated Debugging Considered Harmful" Considered Harmful: A User Study Revisiting the Usefulness of Spectra-Based Fault Localization Techniques with Professionals Using Real Bugs from Large Systems, in: 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2016, pp. 267–278.

[35] S. Aurigemma, R. R. Panko, The detection of human spreadsheet errors by humans versus inspection (auditing) software, in: EuSpRIG 2010 Conference, 2010, pp. 1–14.

[36] J. R. Ruthruff, M. Burnett, G. Rothermel, Interactive fault localization techniques in a spreadsheet environment, IEEE Transactions on Software Engineering 32 (4) (2006) 213–239.

[37] D. Jannach, T. Schmitz, B. Hofer, K. Schekotihin, P. W. Koch, F. Wotawa, Fragment-based spreadsheet debugging, Automated Software Engineering 26 (1) (2019) 203–239.

[38] A. Mukhtar, B. Hofer, D. Jannach, F. Wotawa, Spreadsheet debugging: The perils of tool over-reliance, Journal of Systems and Software 184 (2022) 111119.

[39] T. Miller, Explanation in artificial intelligence: Insights from the social sciences, Artificial Intelligence 267 (2019) 1–38.

[40] W. R. Murray, Automatic Program Debugging for Intelligent Tutoring Systems, 1988.

[41] G. Fey, A. Sulflow, R. Drechsler, Towards unifying localization and explanation for automated debugging, in: 2010 11th International Workshop on Microprocessor Test and Verification, 2010, pp. 3–8.

[42] A. Groce, S. Chaki, D. Kroening, O. Strichman, Error explanation with distance metrics, International Journal of Software Tools for Technology Transfer 8 (2006) 229–247.

[43] W. Dou, S. Cheung, J. Wei, Is spreadsheet ambiguity harmful? detecting and repairing spreadsheet smells due to ambiguous computation, in: 36th International Conference on Software Engineering, 2014, pp. 848–858.

[44] S.-C. Cheung, W. Chen, Y. Liu, C. Xu, CUSTODES: automatic spreadsheet cell clustering and smell detection using strong and weak features, in: 38th International Conference on Software Engineering, 2016, pp. 464–475.

[45] F. Hermans, M. Pinzger, A. van Deursen, Detecting and visualizing inter-worksheet smells in spreadsheets, in: 34th International Conference on Software Engineering, 2012, pp. 441–451.

[46] I. Nunes, D. Jannach, A systematic review and taxonomy of explanations in decision support and recommender systems, User-Modeling and User-Adapted Interaction 27 (3–5) (2017) 393–444.

[47] M. Ribeiro, S. Singh, C. Guestrin, "Why Should I Trust You?": Explaining the Predictions of Any Classifier, in: Procedings NAACL 2016, 2016, pp. 97–101.

[48] S. Lundberg, S.-I. Lee, A unified approach to interpreting model predictions, in: Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS '17), 2017, pp. 4768–4777.

[49] F. D. Davis, Perceived usefulness, perceived ease of use, and user acceptance of information technology, MIS Quarterly 13 (3) (1989) 319–340.

[50] T. Schmitz, D. Jannach, Finding errors in the Enron spreadsheet corpus, in: IEEE Symposium on Visual Languages and Human-Centric Computing, 2016, pp. 157–161.

[51] Y. Chandra, L. Shang, Inductive coding, in: Y. Chandra, L. Shang (Eds.), Qualitative Research Using R: A Systematic Approach, Springer, 2019, pp. 91–106.

[52] K. Wu, Y. Zhao, Q. Zhu, X. Tan, H. Zheng, A meta-analysis of the impact of trust on technology acceptance model: Investigation of moderating influence of subject and context type, International

Journal of Information Management 31 (6) (2011) 572–581.

[53] M. Fisher II, G. Rothermel, The EUSES spreadsheet corpus: a shared resource for supporting experimentation with spreadsheet dependability mechanisms, ACM SIGSOFT Software Engineering Notes 30 (4) (2005) 1–5.

[54] D. Jannach, T. Schmitz, B. Hofer, F. Wotawa, Avoiding, finding and fixing spreadsheet errors - A survey of automated approaches for spreadsheet QA, Journal of Systems and Software 94 (2014) 129–150.

Appendix

		Question ID	Blackbox	Visualization	Natural Language	Human Explanation
User Beliefs	**Perceived Usefulness**	U1	2.60 (1.33)	3.23 (1.05)	3.87 (1.02)	4.40 (0.88)
		U2	2.97 (1.40)	2.50 (1.38)	2.03 (1.14)	1.67 (1.04)
		U3	2.13 (1.18)	2.13 (1.18)	1.80 (0.98)	1.40 (0.92)
		U4	3.10 (1.42)	3.60 (1.31)	3.73 (1.29)	4.37 (0.80)
	Perceived Ease of Use	E1	2.37 (1.30)	2.37 (1.38)	1.63 (0.95)	1.53 (0.92)
		E2	3.30 (1.19)	3.67 (1.14)	3.97 (0.87)	4.23 (0.80)
		E3	3.10 (1.27)	3.27 (1.15)	3.90 (0.91)	4.43 (0.72)
		E4	2.40 (1.14)	2.60 (1.20)	2.03 (1.02)	1.43 (0.76)
	Control	C1	3.53 (1.15)	3.30 (1.10)	3.87 (1.02)	3.87 (0.85)
	Transparency	T1	3.03 (1.40)	3.53 (1.33)	4.20 (0.83)	4.57 (0.56)
User Attitudes	**Trust and Confidence**	TC1	3.50 (1.18)	4.07 (0.96)	4.07 (0.93)	4.47 (0.72)
		TC2	3.57 (1.09)	3.87 (1.09)	4.23 (0.84)	4.27 (0.73)
	Satisfaction	S1	2.43 (1.41)	2.30 (1.27)	1.77 (1.02)	1.77 (0.96)
		S2	3.30 (1.22)	3.53 (0.99)	3.90 (0.83)	4.07 (0.73)
Behavioral Intentions	**Recommendation Intentions**	R1	3.20 (1.28)	3.27 (1.26)	3.90 (1.16)	4.2 (0.87)
	Intention to use	I1	2.90 (1.33)	3.13 (1.36)	3.70 (1.07)	4.13 (1.02)
		I2	2.27 (1.21)	2.07 (1.18)	1.63 (0.80)	1.13 (0.34)

Table 7: Mean (standard deviations) of participant responses in terms of user beliefs, user attitudes, and behavioral intentions. The exact questions are shown in Table 3.