Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Spreadsheet debugging: The perils of tool over-reliance[☆]

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

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

ARTICLE INFO

Article history:
Received 17 March 2021
Received in revised form 29 September 2021
Accepted 30 September 2021
Available online 16 October 2021

Keywords:
Fault identification
Spreadsheets
User study
Over-reliance

ABSTRACT

Spreadsheets are widely used in organizations for various purposes such as data aggregation, reporting and decision-making. Since spreadsheets, like other types of software, can contain faulty formulas, it is important to provide developers with appropriate methods to find and fix such faults. Recently, various heuristic and statistics-based fault identification methods were proposed, which point developers to potentially faulty parts of the spreadsheets. Due to their heuristic nature, these methods might, however, miss some faults. As a result, if spreadsheet developers rely too strongly on these methods, they might not pay sufficient attention to problems that are not pinpointed by the methods. In this research, we are the first to study this potential problem of *over-reliance* in spreadsheet debugging, which may lead to limited debugging effectiveness. We report the outcome of a controlled experiment where 59 participants were tasked to find faulty formulas in a given spreadsheet with and without support of a novel spreadsheet debugging tool. Our results indicate that tool over-reliance can indeed result as a phenomenon of using heuristic debugging techniques. However, the study also provides evidence that making users aware of potential tool limitations *within* the debugging environment may help to address this problem.

Editor's note: Open Science material was validated by the Journal of Systems and Software Open Science Board.

1. Introduction

Spreadsheets are widely used in organizations for all types of calculations and decision-making processes. However, spreadsheets are prone to error, and various examples exist where faulty spreadsheet formulas led to significant financial losses for organizations.[1] Worryingly, since spreadsheets are in many cases created by users who are not IT experts, the error rate is often assumed to be higher than in traditional software. According to McConnell (2004) (p. 521), the average industrial experience is 1–25 errors per 1000 lines of code. Depending on the underlying quality assurance techniques and processes, this rate can be further reduced and companies like Microsoft report 0.5 defects per 1000 lines of code in released products. The error rate in spreadsheets, in contrast, is estimated to be as high as about 3%–5% (Panko and Port, 2013). Therefore, a variety of approaches for assuring the quality of spreadsheets were proposed over the years (Jannach et al., 2014), often with a particular focus on the problem that spreadsheet users in many cases have no education in software engineering.

One important means to help users avoid or detect errors are *static checkers* and tools for automated *fault localization*. Such tools are designed to aid spreadsheet developers in particular during maintenance and debugging activities. Various fault localization techniques were proposed over the years for *general*, e.g., procedural or object-oriented, software (Wong et al., 2016; Stefanović et al., 2020). In many cases, such tools determine a set of "suspicious" elements in the program code and provide developers with a ranked list of code artifacts for further inspection. The set of suspicious elements (often termed *candidates*) can be determined in different ways. In some cases, analytical approaches are adopted, which, for example, investigate the data flow in the program (Soremekun et al., 2021) or use model-based techniques (Ceballos et al., 2019) to identify problematic parts of a program. Another approach is to use *heuristics* to pinpoint areas in a program that are assumed to have a certain probability to contain a fault. Typical approaches of that type are software *smells* (Ma et al., 2016), software fault prediction (Catal, 2011) and spectrum-based fault localization techniques (Xie and Xu, 2021; Abreu et al., 2009).

[☆] Editor: Neil Ernst.

[*] Correspondence to: Inst. f. Software Technology, Technical University Graz, Inffeldgasse 16b/EG, 8010, Graz, Austria.
E-mail addresses: amukhtar@ist.tugraz.at (A. Mukhtar), bhofer@ist.tugraz.at (B. Hofer), dietmar.jannach@aau.at (D. Jannach), wotawa@ist.tugraz.at (F. Wotawa).

[1] http://www.eusprig.org/horror-stories.htm.

https://doi.org/10.1016/j.jss.2021.111119
0164-1212/

A number of different fault identification and debugging techniques were also proposed in the spreadsheet domain, including model-based approaches (Abreu et al., 2015; Jannach and Schmitz, 2016), techniques based on heuristics (Li et al., 2019b,a; Singh et al., 2017; Barowy et al., 2018) and smells (Cunha et al., 2012a; Hermans et al., 2012a), and spectrum-based techniques (Getzner et al., 2017). To evaluate the effectiveness of fault-localization techniques, researchers—also in the spreadsheet domain—commonly rely on computational experiments. In such experiments, algorithms are typically compared in terms of their capability of ranking the fault candidates in a way that the true faults appear earlier in the list than the *false positives* that may also be returned in particular by heuristic techniques. While such an approach is generally plausible, the question was raised by Parnin and Orso (2011) and subsequent works if some of the assumptions of such computational experiments always hold. In the end, this amounts to the question to what extent existing fault localization techniques—or more generally, automated debugging techniques—really help programmers, see also Xia et al. (2016). Furthermore, it is unclear if there might even be unexpected negative side-effects of providing such tools, for example, that programmers rely too much on them and focus only on potential issues that are highlighted by the tool.

Ultimately, such questions can only be answered with certain reliability through studies that involve humans. As pointed out by Parnin and Orso (2011), developers might not behave as expected when they investigate a list of fault candidates. They, for example, might not recognize the true fault even when pointed to it, or they might not inspect the list of candidates from top to bottom but in another order. In our work, we continue this line of research on the *human side* of automated fault identification. Specifically, we focus on the question if the existence of a fault identification tool might lead to a certain level of *over-reliance* on the tool on the user's side. In this context, over-reliance might lead to the effect that developers trust the tool too much and, as a result, mainly or solely inspect the highlighted candidates but not other parts of the program.

The potential threats of user over-reliance on tools have not yet been studied deeply in the software engineering literature, and—to our knowledge—not at all in the domain of spreadsheets. Over-reliance on technology in general is, however, not a new phenomenon. In the medical domain, for example, Goddard et al. (Goddard et al., 2014) analyzed the extent of *automation bias* through an empirical investigation. Specifically, they measured how often medical doctors followed an automated *incorrect* advice and changed their previously correct prescription for a given scenario. A systematic review of the literature on automation bias (Goddard et al., 2011) indicates that this phenomenon exists across different research fields such as healthcare, aviation, military or general Human Computer Interaction.

To close these existing research gaps, we study the phenomenon of tool over-reliance—which is potentially caused by an underlying automation bias—for the problem of spreadsheet debugging. In our research, we conducted a between-subjects user study where 59 participants were tasked to locate faulty formulas in a given real-world spreadsheet. The participants in two different treatment groups were supported by a debugging tool which highlighted faulty formulas based on a recent metrics-based fault identification approach (Koch et al., 2019). To measure the extent of tool over-reliance, we, however, injected additional faults to the spreadsheet of which we knew that the metrics-based approach will not highlight them. Our general research question then was if the study participants in the treatment groups would focus too much on the formulas that are highlighted by the tool and, as a result, on average miss more of the faulty formulas than the control group, which had no tool support.

The obtained results in fact indicate that participants who were provided with the fault identification tool may end the debugging task too prematurely, and, as a result, may identify fewer of the injected faults. In particular, they primarily miss faults that are not highlighted by the tool. However, when we raised the awareness of users about the potential limitations of the fault identification tool by showing a warning, it turned out that users were inspecting the spreadsheets more carefully and missed fewer faults. Overall, our work, therefore, has important practical implications regarding the design of fault identification tools.

The paper is organized as follows. In Section 2, we discuss previous works on debugging. Background information about the novel debugging tool named SmellChecker that was used in the context of the study is given in Section 3. We provide details about the experiment design and the study materials in Section 4 and present the results of the study in Section 5. Section 6 discusses potential threats to validity. The work ends with a discussion of practical implications and an outlook on future works.

2. Previous works

This section discusses related works in the context of software and spreadsheet debugging (Section 2.1), human-centric evaluation approaches (Section 2.2), and research on over-reliance and automation bias (Section 2.3).

2.1. Debugging

Software debugging is the process of eliminating faults in programs and comprises fault detection, fault localization, the understanding of the nature of the fault, and its correction. Starting about 40 years ago, the automation of debugging has received increasing interest and several techniques have been introduced since then. Ducassé (1993) categorized debugging into three classes: *(i)* verification with respect to specifications, *(ii)* checking with respect to language knowledge, and *(iii)* filtering with respect to a symptom. Although the research area of automated debugging has evolved over time, many of today's debugging approaches can still be assigned to one of these categories. The SmellChecker tool used in our study falls in the second category, as it uses spreadsheet metrics to point to potential faulty cells.

Wong et al. (2016) provided a survey on debugging approaches focusing on spectrum-based fault localization, model-based debugging, and also the application of machine learning and information theory for fault localization in programs. Checking techniques like the SmellChecker tool for spreadsheets were not considered in this survey. Quality assurance methods and debugging techniques for spreadsheets where however reviewed by Jannach et al. (2014). Their survey covers a variety of approaches for fault localization and repair, and also includes checking techniques. Other work in the context of spreadsheet debugging providing the basis of checking techniques include (Cunha et al., 2012a; Hermans et al., 2012a; Cunha et al., 2012b; Hermans et al., 2012b).

The increased interest in debugging of spreadsheets led to the development of various tools, e.g., ExceLint (Barowy et al., 2018), Melford (Singh et al., 2017), CUSTODES (Cheung et al., 2016), Warder (Li et al., 2019a), SGUARD (Li et al., 2019b), CACheck (Dou et al., 2017), and the approach in Xu et al. (2018). All these mentioned works comprise an evaluation of the proposed tools, and these evaluations usually consider already available other tools for debugging. However, many of the existing evaluations mainly aim on demonstrating the improved precision of debugging when the tool is used. Computational metrics alone (like precision) can however not fully inform us about the usefulness of a tool as perceived by users. In contrast to such works, we are not only presenting a tool for debugging spreadsheets but also conducted a user study aiming to investigate how SmellChecker is supporting users in finding faults.

Generally, note that also techniques other than checking—as used by SmellChecker—were investigated for spreadsheet debugging in the past. This includes the use of constraints for localizing faults (Abreu et al., 2015; Jannach and Engler, 2010; Schmitz and Jannach, 2017; Abreu et al., 2012), and the use of spreadsheet fragments for improved test case generation and fault localization (Jannach et al., 2019). A comparison of such alternative spreadsheet debugging approaches is however beyond the scope of our present work.

2.2. Human-centric evaluation

Research dealing with the human-centric evaluation of automated (spreadsheet) debugging techniques is generally sparse. Ko and colleagues (Ko and Myers, 2004, 2008) introduced a tool allowing users to answer *Why?* and *Why not?* questions during debugging. They performed a first small user study aiming to show that using their tool reduces the overall debugging time. Burg et al. (2013) introduced an interactive "record-and-replay" tool for debugging. The authors conducted a user study with 14 web developers and could show that using the tool led both to a reduction in the needed time and to an increase of the degree of completion of the debugging task.

Parnin and Orso (2011) carried out a study comparing the debugging performance of programmers where one group was using a spectrum-based fault localization tool and the other group performed debugging manually. They found that tools not necessarily help users carrying out debugging tasks efficiently. As a consequence of their study, Parnin and Orso made some suggestions for future research including improving the user interface of debugging as well as tool integration. Xie and colleagues (Xie et al., 2016) revisited Parnin and Orso's study confirming that debugging tools do not substantially improve the overall debugging performance and thus require additional research work.

Interestingly, Xia and colleagues (Xia et al., 2016) also carried out a similar study, which however led to a different result regarding the usefulness of debugging tools. In contrast to the previous studies, Xia et al. focused on debugging support considering larger software and experienced programmers to carry out debugging opposed to previous studies, which relied on students. Xia et al. were able to show that tools indeed improve debugging performance, which depends to some extent on the ranking of fault candidates presented to the user.

More recent work on the evaluation of debugging tools includes feedback-based debugging (Lin et al., 2017), AI-assisted game debugging (Machado et al., 2018) and enlightened debugging (Li et al., 2018). In the context of spreadsheet debugging, Aurigemma and Panko (2010) conducted a user study comparing the error detection capabilities of humans with the one of certain inspection functions provided in spreadsheet implementations showing that humans performed much better. Ruthruff et al. (2006) introduced several techniques for debugging based on a specific testing methodology. The authors conducted a user study involving students. In their study, they focused on the effectiveness of their techniques considering various parameters that may influence the outcome. The underlying parameters refer to the type of information and the way the information is presented to the user. Ruthruff et al. showed that these parameters have an impact on debugging effectiveness.

In the context of spreadsheets, Jannach et al. (2019) carried out a user study with the objective to show that spreadsheet fragments improve test case generation. This user study is different to the user study on which we report in this paper where the focus is on effects obtained when using SmellChecker for fault identification.

In summary, we conclude that in literature regarding tools for fault localization we find both computational experiments without users and user studies. However, in none of discussed papers over-reliance has been considered. The papers mainly focused on debugging performance, i.e., whether there is an impact on time required to localize and fix faults, or the completeness of the debugging results.

2.3. Over-reliance and automation bias

Humans sometimes tend to rely on computer-generated decisions even when they have information contradicting the computer-generated decisions. This phenomenon is called automation bias. Automation bias and over-reliance are well-known problems for decades. In the 1990s, researchers highlighted the risks of automation bias (Mosier and Skitka, 1996). Over-reliance and automation bias are present in many different domains, e.g., health care (Goddard et al., 2011; Lyell and Coiera, 2016), military (Bryant, 2011), and automotive and aviation industry (Blackett, 2021). Over-reliance on technology caused many unnecessary deaths in these domains.

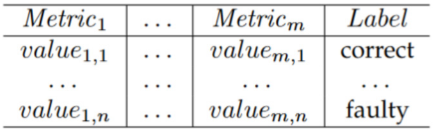
A prominent example of over-reliance are self-driving cars. Although current self-driving technology explicitly requires the constant monitoring of the traffic by the driver, a recent observational study has shown that drivers of self-driving cars are more often distracted, e.g., by their mobile phones, than drivers of conventional cars (Noble et al., 2021). The authors of the study suggest that the driver training should raise awareness about this problem.

However, even though over-reliance is a long-known problem, research on raising awareness and avoiding over-reliance is still at the beginning. In healthcare, the current recommendations to reduce over-reliance are *(i)* to reduce the amount of human interaction wherever possible (e.g., through electronic data transmission), *(ii)* to design technology in a way that reduces over-reliance (e.g., to avoid auto-completion of drug names when writing prescriptions), and *(iii)* to provide proper training (Matthew, 2019).

Mosier (2019) provide an overview of factors influencing automation bias. They distinguish between human factors (e.g., trust, experience, and accountability), system factors (e.g., reliability, informative vs. imperative/commanding, visibility), and task factors (e.g., time pressure, workload, team vs. individual).

Recently, researchers have addressed the problem that students often rely on autograders in software engineering classes. To counteract this phenomenon, they introduced a regression penalty: the student's final score decreases each time the student's new submission yields a lower score than their previous submission. The penalty achieved that the students performed more tests on their own and were more cautious when submitting their code (Baniassad et al., 2021).

The above examples used two methods to raise awareness: training and technology support. Users often learn spreadsheets by doing without any proper training. It is difficult to reach users to teach them about the risks that come from over-reliance. Therefore, we focus on raising awareness through technology support by notifying users about limitations of the tool.



$Metric_1$	...	$Metric_m$	$Label$
$value_{1,1}$	...	$value_{m,1}$	correct
...	...	...	...
$value_{1,n}$	...	$value_{m,n}$	faulty

Fig. 1. Supervised learning approach in SmellChecker.
Source: Adapted from Koch et al. (2019).

3. Study environment: The SmellChecker tool

As a basis for our user studies, we relied on the SmellChecker tool, which we developed as a plugin to the online version of Microsoft Excel. SmellChecker implements a novel machine-learning based approach for fault identification in spreadsheets described in our earlier work (Koch et al., 2019). Here, we will provide a brief overview on the tool and its functionality and the user interface.

3.1. Fault identification approach

SmellChecker uses a heuristic and *supervised* learning-based approach to fault identification. Specifically, it is based on training a machine-learning model using a collection of spreadsheets with known faults as training data to predict if a given formula in a new spreadsheet is faulty or not.

The general structure of the supervised learning problem is sketched in Fig. 1. Each row in the table corresponds to a spreadsheet formula in the training data. The binary label expresses for each formula if it was faulty or not. The columns of the table (or: *features*) correspond to spreadsheet-specific *product metrics*, see Hofer et al. (2021) for a systematic review of such metrics. Each metric can be considered as a heuristic that indicates a likelihood that the given formula is faulty. The complexity of a given formula, e.g., in terms of the used variables or the nesting level, would be an example of a metric. More complex formulas are often assumed to be more prone to error, and the corresponding metric would, therefore, return higher values for more complex formulas. In our previous work (Koch et al., 2019), we proposed a catalog of more than 60 metrics that can be used as features in the machine learning approach. We have used all of these metrics in our approach. The number of referenced cells and the number of function operators turned out to be among the most important features.

Once the input data for the learning problem is designed, as shown in Fig. 1, various machine learning techniques can be applied. In our previous research, we for example experimented with Support Vector Machines, Random Forests, and a Deep Learning model. Furthermore, we trained and tested the models using different spreadsheet corpora with known faults. Ultimately, we found that in particular Random Forests led to consistently strong performance levels for all data-sets in terms of precision, recall, and F1. In the experiments reported in our present paper, we, therefore, relied on Random Forest as classification method. To maximize the amount of training data, we furthermore combined the three individual spreadsheet corpora from our previous work (Koch et al., 2019) into one larger corpus.

While the prediction approach is learning-based, it is also a heuristic one because the features of the model, i.e., the product metrics (including the smells), are mainly heuristics that were previously proposed by spreadsheet researchers. Therefore, the prediction method's accuracy largely depends not only on the available training data, but also on the metrics that are included in the model. Moreover, the learned model returns a value between 0 and 1 for a given formula, which expresses a formula's estimated degree of being faulty. In practice, in particular this latter aspect means that a *threshold* must be defined in a fault identification tool that is based on this approach. This threshold can then be used by the debugging tool to decide if a formula "smells enough" to be brought to the attention of the spreadsheet user.

As a result, depending on the setting of the threshold, the system can either be configured to be "aggressive" or "conservative". In an aggressive setting, the threshold will be set to a low value, and already lightly smelling formulas will be reported as potentially faulty. Such a setting can easily lead to false alarms (i.e., *false positives*), where actually correct formulas are indicated as being suspicious. In a more conservative setting, only formulas with a high predicted fault probability will be highlighted, leading to the danger of missing actually faulty cells there were not brought to the attention of the spreadsheet user by the tool (i.e., *false negatives*). In our research on potential tool over-reliance in debugging settings, we are specifically interested in the second type of problem, i.e., where the system did *not* mark some existing faults as potentially faulty, and where over-relying users may not invest sufficient time to search the spreadsheet for such faults. Note that in the original proposal of SmellChecker, we evaluated the underlying learning-based approach with the help of offline experiments. Since this study focuses on the evaluation with users, we designed a user interface which we discuss next.

3.2. User interface

We assume that the design of the user interface (UI) can be highly decisive for the adoption of a fault identification tool in practice. Therefore, we designed a user interface that *(i)* is smoothly integrated into the surrounding spreadsheet environment, and that *(ii)* puts only limited cognitive burden on the users, e.g., by limiting the provided functionality and UI elements as much as possible.

SmellChecker is implemented as a plugin to the web-based version of Microsoft Excel, which also avoids the need to have local copies of Microsoft Excel installed on the computers of the study participants. After the plugin is activated, the main "ribbon" (interactive menu) of Microsoft Excel is extended with an additional symbol, as shown in Fig. 2.

Once the user (here: study participant) clicks on the symbol, a task pane[2] opens on the right hand side of the screen. When the pane is opened the first time, it displays short information about the tool and provides a single-button labeled with "Compute Suspiciousness".

After the button is hit, the tool computes the fault predictions for each cell in the spreadsheet and visually highlights those cells where the prediction value surpasses the given threshold. When copy-equivalent cells[3] are suspicious, SmellChecker highlights all of them. In addition, the tool populates the pane at the right-hand side of the screen with summary statistics and links to the formulas that are considered suspicious, see also Fig. 3. With the help of additional buttons, the user can reset the visual highlighting and recompute the suspiciousness scores in case changes to the formulas were applied. Changing individual formulas was not required in the particular study; in a real-world application, it is, however, likely that a developer would repeatedly calculate the suspiciousness of cells after a debugging activity.

For the sake of the user study, we added an additional button to the pane (labeled "Finish Task"), which study participants could use to move to the next step of the online experiment.

[2] https://docs.microsoft.com/en-us/office/dev/add-ins/design/task-pane-add-ins.

[3] Copy-equivalent cells do the same computations, but use different inputs (Schmitz et al., 2017).

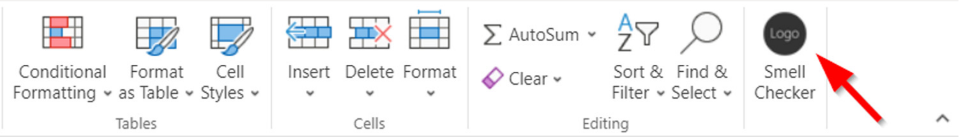
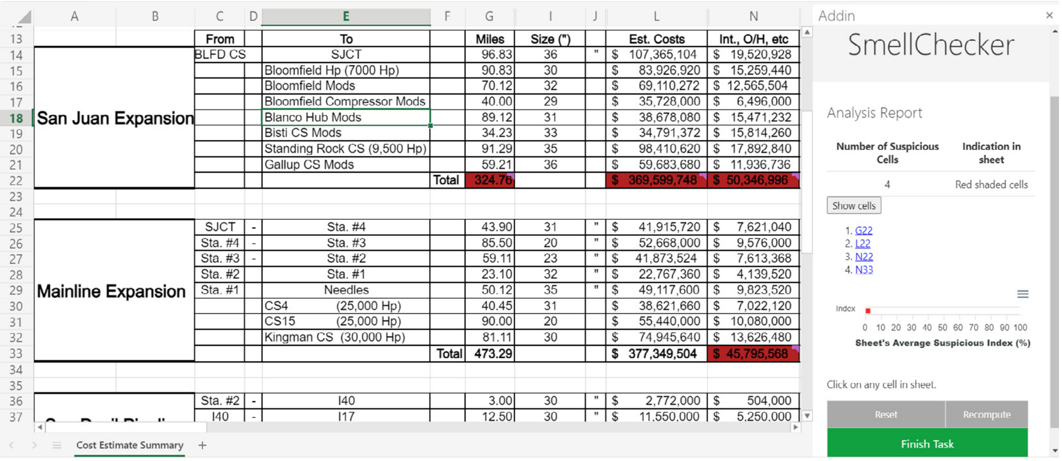


Fig. 2. Extended *ribbon* of Microsoft Excel after activating SMELLCHECKER.



	A	C	D	E	F	G	I	J	L	N
13		From		To		Miles	Size (")		Est. Costs	Int., O/H, etc
14	San Juan Expansion	BLFD CS		SJCT		96.83	36	"	$ 107,365,104	$ 19,520,928
15				Bloomfield Hp (7000 Hp)		90.83	30		$ 83,926,920	$ 15,259,440
16				Bloomfield Mods		70.12	32		$ 69,110,272	$ 12,565,504
17				Bloomfield Compressor Mods		40.00	29		$ 35,728,000	$ 6,496,000
18				Blanco Hub Mods		89.12	31		$ 38,678,080	$ 15,471,232
19				Bisti CS Mods		34.23	33		$ 34,791,372	$ 15,814,260
20				Standing Rock CS (9,500 Hp)		91.29	35		$ 98,410,620	$ 17,892,840
21				Gallup CS Mods		59.21	36		$ 59,683,680	$ 11,936,736
22					Total	324.76			$ 369,599,748	$ 50,346,996
23										
24										
25	Mainline Expansion	SJCT	-	Sta. #4		43.90	31	"	$ 41,915,720	$ 7,621,040
26		Sta. #4	-	Sta. #3		85.50	20	"	$ 52,668,000	$ 9,576,000
27		Sta. #3	-	Sta. #2		59.11	23	"	$ 41,873,524	$ 7,613,368
28		Sta. #2		Sta. #1		23.10	32	"	$ 22,767,360	$ 4,139,520
29		Sta. #1		Needles		50.12	35	"	$ 49,117,600	$ 9,823,520
30				CS4 (25,000 Hp)		40.45	31		$ 38,621,660	$ 7,022,120
31				CS15 (25,000 Hp)		90.00	20		$ 55,440,000	$ 10,080,000
32				Kingman CS (30,000 Hp)		81.11	30		$ 74,945,640	$ 13,626,480
33					Total	473.29			$ 377,349,504	$ 45,795,568
34										
35										
36		Sta. #2	-	I40		3.00	30	"	$ 2,772,000	$ 504,000
37		I40	-	I17		12.50	30	"	$ 11,550,000	$ 5,250,000

Cost Estimate Summary

Addin

SmellChecker

Analysis Report

Number of Suspicious Cells	Indication in sheet
4	Red shaded cells

Show cells

1. G22
2. L22
3. N22
4. N33

Click on any cell in sheet.

Reset Recompute

Finish Task

Fig. 3. Spreadsheet with highlighted cells and the overview pane.

3.3. System architecture

The overall software architecture of SMELLCHECKER, as used in the study, is shown in Fig. 4. The user of the tool interacts with the online version of Microsoft Excel through the browser. The online user interface of Excel is extended by us with the SMELLCHECKER tool, which is built using the "Excel Javascript API" provided by Microsoft Office. Plugins are written in JavaScript and run through *node.js*.[4]

Through the plugin mechanisms, additional UI elements are created, as shown in Fig. 3. The plugin code furthermore implements the system's reactions when the user interacts with the plugin UI elements. In our case, the plugin communicates with a web server, which is run by us and which implements the necessary back-end logic for the debugging process. Specifically, the server hosts *(i)* the code that implements the machine learning model (implemented in Python based on Koch et al. (2019)) and *(ii)* code that computes the values for the product metrics (implemented in Java based on Koch et al. (2016)). Generally, when we designed the architecture and implemented the software, we paid special attention to response times and fast client–server communication round-trips to ensure that the user interface reacts promptly to user actions. Furthermore, we logged the relevant information for the user study (task completion time, number of cells traversed, number of recomputes, etc.) in a MongoDB database. We discuss the design of our study in detail in the next section.

4. Study design and execution

In what follows we describe our research hypothesis, the specific research questions, the corresponding study design, and how the study was executed.

4.1. Research questions

Given the discussions in Section 2, over-reliance on tools is not an uncommon phenomenon, which has, however, not been explored largely within tool-support software development processes. In the context of our studies, our hypothesis is that tool over-reliance in the debugging process manifests itself in a lack of attention to parts of the program (i.e., formulas) that are not highlighted by the supporting tool as being potentially faulty. Our first research question, therefore, is:

RQ1: *Do users who receive tool support by means of* SMELLCHECKER *miss more faults, on average, than users that debug spreadsheets without additional tool support?*

However, we also hypothesize that tool users will look more carefully also at non-highlighted formulas once they are more *aware* of the potential limitations of the tool, i.e., that also non-highlighted formulas might be faulty. Note that such awareness of tool limitations in practice often comes from longer-term experiences with a tool. In our study, we will raise this awareness through explicit informational hints. Therefore, our second research question is:

RQ2: *Does raising the awareness about the potential limitations of* SMELLCHECKER *impact the fault detection and localization rate?*

A prerequisite to investigate the effects of a debugging tool on the observed outcomes is that the tool itself is perceived as being usable and useful by users in the first place. Therefore, we have to check the following research question to validate that the results are not negatively affected by the tool's usability.

RQ3: *How do users perceive and assess the general utility (usability and usefulness) of the* SMELLCHECKER *debugging tool?*

4.2. Materials and procedure

General overview. In the conducted between-subjects user study, participants were tasked to locate faults in a given spreadsheet. Participants in two *treatment* groups (referred to as treatment groups A and B) were supported through the debugging functionality of SMELLCHECKER; participants in the *control* group did

[4] https://nodejs.org/.

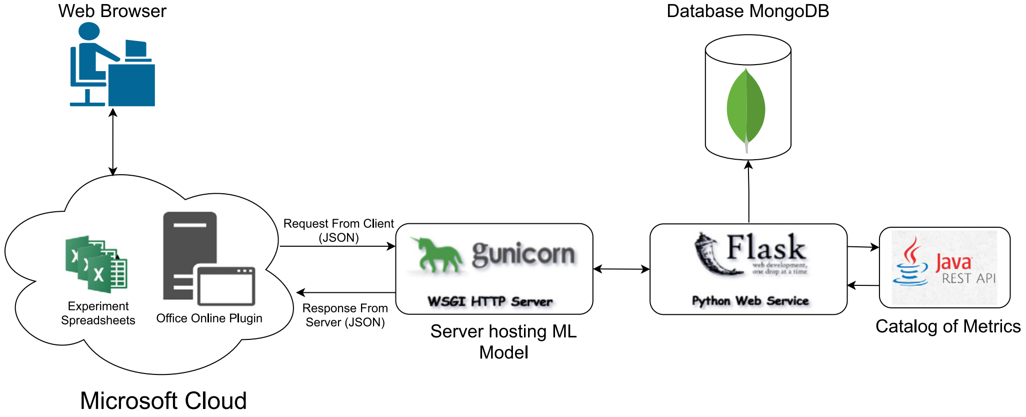


Fig. 4. Architecture of SmellChecker (as used in study).

not receive such support. As the main dependent variable, we measured the fault identification performance of the participants. Here, we determined in particular how many of the existing faults were properly identified as such.

Both treatment groups were informed in the user manual of the tool that it might not be able to find all faults. Participants in group B additionally *(i)* had to tick a checkbox before they could start the task, confirming they understood the limitations of the tool,[5] and *(ii)* were shown a message when they clicked on the button to finish the fault identification task. The message informed participants that the SmellChecker tool will not necessarily highlight all existing faults, and allowed participants to continue looking for faults if they wanted. An overview of what was presented to different groups after the plugin was installed is shown in Fig. 5.

We added these warnings for treatment group B to investigate RQ2 from above regarding the effects of raising the awareness among study participants regarding the potential limitations of the tool.

Faulty spreadsheet. Based on our experiences from previous user studies on spreadsheet debugging (Jannach et al., 2019; Jannach and Schmitz, 2016), we selected one real-world spreadsheet from the Enron error corpus (Schmitz and Jannach, 2016) for our experiment. A main challenge in that context is to find a spreadsheet that:

- is complex enough to ensure that the debugging task is not trivial;
- is simple enough so that participants are able to comprehend how it is intended to work and locate the faults within reasonable time;
- represents typical spreadsheets found in practice; and
- contains common faults that one would find in real-world spreadsheets.

The selection of one particular spreadsheet may arguably be a potential limitation of studies like ours. Through the careful selection of the test spreadsheet we are, however, confident that the findings are representative for many spreadsheets found in the real world. After inspecting various spreadsheets for the experiment and after making pretests with a smaller short list of candidates, we selected the "Sun Devil Project Request.xls" spreadsheet from the Enron error corpus, shown in Fig. 6.

The spreadsheet implements typical project cost estimations in the energy domain. It consists of 50 formulas, 12 of them are unique in terms of R1C1 notation (i.e., there are several copy-equivalent cells). The spreadsheet formulas consist of various multiplications and summations. Compared to other spreadsheets from the Enron or EUSES corpus (Marc and Rothermel, 2005), we consider this spreadsheet to be of still manageable complexity, e.g., in terms of the number of unique formulas. The spreadsheet is also comparable in terms of its complexity to spreadsheets used in related studies in the domain (Jannach and Schmitz, 2016; Jannach et al., 2019).

The original spreadsheet, as found in the Enron error corpus, contains one fault in a formula in cell L22. It is a typical range error, where not all relevant cells were included in the calculation.[6] For the purpose of study, we, however, injected eight additional faults in the form of *mutations* (Abraham and Erwig, 2009) into the spreadsheet: Three faults (cells G22, N22 and N33), which are detected by the tool, and five faults (cells L18, N19, N37, L29 and L21), which were not detected with the given configuration of the metric thresholds. To be able to assess potential effects of tool over-reliance, our goal was to deliberately create a situation where the tool highlights some of the true faults and does *not* highlight some other faults. To keep the complexity of the study manageable, we decided not to include situations where the tool incorrectly marks a correct formula as being faulty.

After injecting an equal number of range and semantic faults, our spreadsheet contained nine faults (including the original one).[7] We then configured the SmellChecker tool in a way that it will highlight four of these nine faults and not highlight the remaining five.

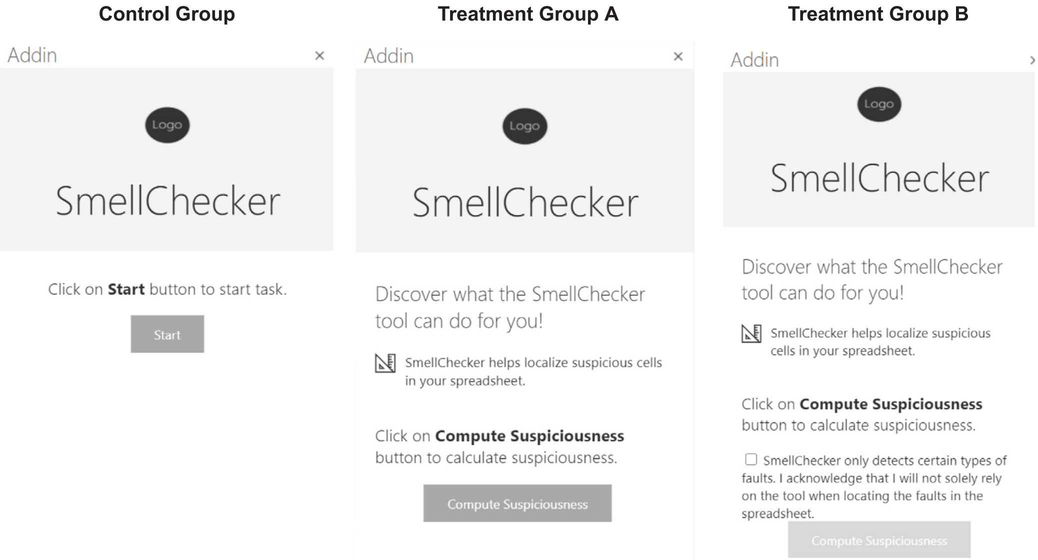
Supporting materials. All study participants were provided with the following additional materials, which we also share online:

- A short overview of their tasks during the study of about 130 words.
- An installation guide describing how to install the plugin. The installation simply consists of uploading a provided XML file to MS Excel online. All participants (i.e., in both treatment groups and in the control group) had to install a plugin; in the case of the control group, the plugin, however, did not provide the debugging functionality, but only displayed buttons for the users to indicate when they start and stop debugging, and a link to the post-task questionnaire.

[5] The text read: "SmellChecker *only detects certain types of faults. I acknowledge that I will not solely rely on the tool when locating the faults in the spreadsheet*".

[6] The cell containing this original fault will be detected and highlighted by our SmellChecker tool as being very suspicious, even when the threshold for reporting suspicious cells is set to a high value. Therefore, our method developed in previous work (Koch et al., 2019) proved to be effective for the original spreadsheet.

[7] To ensure reproducibility of our work, we share the spreadsheet together with other materials of our study online at https://doi.org/10.5281/zenodo.5533460.



Control Group

Addin ×

SmellChecker

Click on **Start** button to start task.

Start

Treatment Group A

Addin ×

SmellChecker

Discover what the SmellChecker tool can do for you!

SmellChecker helps localize suspicious cells in your spreadsheet.

Click on **Compute Suspiciousness** button to calculate suspiciousness.

Compute Suspiciousness

Treatment Group B

Addin ›

SmellChecker

Discover what the SmellChecker tool can do for you!

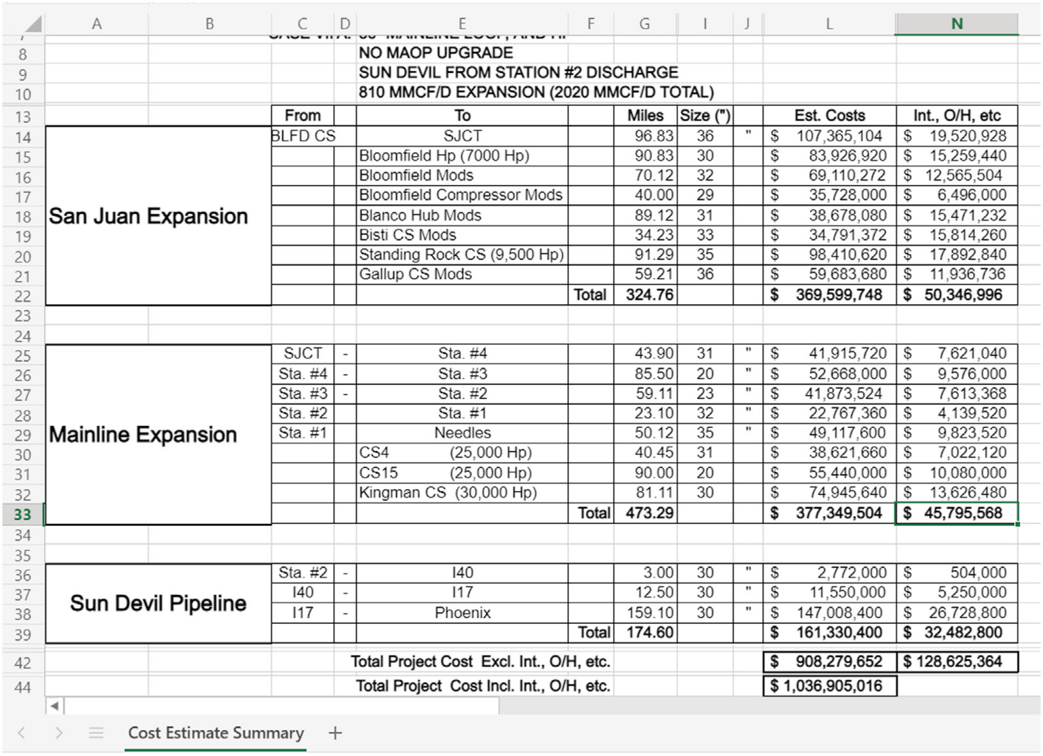
SmellChecker helps localize suspicious cells in your spreadsheet.

Click on **Compute Suspiciousness** button to calculate suspiciousness.

☐ SmellChecker only detects certain types of faults. I acknowledge that I will not solely rely on the tool when locating the faults in the spreadsheet.

Compute Suspiciousness

Fig. 5. User interfaces (UI) shown to the participants based on the group type i.e., control, treatment A and treatment B.



	A	B	C	D	E	F	G	I	J	L	N
8					NO MAOP UPGRADE						
9					SUN DEVIL FROM STATION #2 DISCHARGE						
10					810 MMCF/D EXPANSION (2020 MMCF/D TOTAL)						
13			From		To		Miles	Size (")		Est. Costs	Int., O/H, etc
14			BLFD CS		SJCT		96.83	36	"	$ 107,365,104	$ 19,520,928
15					Bloomfield Hp (7000 Hp)		90.83	30		$ 83,926,920	$ 15,259,440
16					Bloomfield Mods		70.12	32		$ 69,110,272	$ 12,565,504
17					Bloomfield Compressor Mods		40.00	29		$ 35,728,000	$ 6,496,000
18	San Juan Expansion				Blanco Hub Mods		89.12	31		$ 38,678,080	$ 15,471,232
19					Bisti CS Mods		34.23	33		$ 34,791,372	$ 15,814,260
20					Standing Rock CS (9,500 Hp)		91.29	35		$ 98,410,620	$ 17,892,840
21					Gallup CS Mods		59.21	36		$ 59,683,680	$ 11,936,736
22						Total	324.76			$ 369,599,748	$ 50,346,996
23											
24											
25			SJCT	-	Sta. #4		43.90	31	"	$ 41,915,720	$ 7,621,040
26			Sta. #4	-	Sta. #3		85.50	20	"	$ 52,668,000	$ 9,576,000
27			Sta. #3	-	Sta. #2		59.11	23	"	$ 41,873,524	$ 7,613,368
28			Sta. #2		Sta. #1		23.10	32	"	$ 22,767,360	$ 4,139,520
29	Mainline Expansion		Sta. #1		Needles		50.12	35	"	$ 49,117,600	$ 9,823,520
30					CS4 (25,000 Hp)		40.45	31		$ 38,621,660	$ 7,022,120
31					CS15 (25,000 Hp)		90.00	20		$ 55,440,000	$ 10,080,000
32					Kingman CS (30,000 Hp)		81.11	30		$ 74,945,640	$ 13,626,480
33						Total	473.29			$ 377,349,504	$ 45,795,568
34											
35											
36			Sta. #2	-	I40		3.00	30	"	$ 2,772,000	$ 504,000
37	Sun Devil Pipeline		I40	-	I17		12.50	30	"	$ 11,550,000	$ 5,250,000
38			I17	-	Phoenix		159.10	30	"	$ 147,008,400	$ 26,728,800
39						Total	174.60			$ 161,330,400	$ 32,482,800
42					Total Project Cost Excl. Int., O/H, etc.					$ 908,279,652	$ 128,625,364
44					Total Project Cost Incl. Int., O/H, etc.					$ 1,036,905,016	

Fig. 6. Spreadsheet used in the study. We slightly reformated the layout of the original spreadsheet to make it more compact and we added plausible numbers in the input cells.

- A user manual for the plugin. The manuals for the treatment and control groups were different. While the manual for the treatment groups showed how to operate the plugin's debugging functionality, the manual for the control group only explained the mentioned buttons for starting and stopping the debugging task. These buttons were present both for the control and treatment groups.
- A short description of about 110 words regarding the functionality of the spreadsheet to be debugged because some basic understanding of the intended functionality of the spreadsheet is needed to be able to debug it.

Post-task questionnaire. In the post-task questionnaire, we made inquiries regarding different aspects. We list the detailed questionnaire items below in Table 1. The questionnaire consisted of the following blocks.

- *Demographics & Expertise:* Five questions (D1 to D5) targeted at understanding the background and spreadsheet expertise of the users.
- *Perceived Effort of Fault Identification:* Six questions (E1 to E6), inspired by Sung et al. (2015), were used to assess how effortful the task was perceived by the participants. We used 5-point Likert scale items for these questions.

Table 1
Post-task questionnaire items.

ID	Question and answer options
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 spreadsheet in your daily work? *Less than a year/1-3 years/4-6 years/More than 6 years*
D3	I rate my expertise with spreadsheets as follows. *Beginner/Intermediate/Advanced/Expert/Free Text Answer*
D4	How old are you? *18-25 years/25-30 years/30-35 years/35-40 years/40 years - or older*
D5	What is your gender? *Male/Female/Other/Prefer not to say*
Perceived Effort	
All answers are on a 1-5 Likert scale (1 = strongly disagree; 5 = strongly agree)	
E1	It was difficult for me to locate faults.
E2	It took a lot of time to locate faults.
E3	The provided description of the spreadsheet was sufficient to understand the formulas.
E4	The provided spreadsheet was large and complex.
E5	Installing the tool in the spreadsheet was easy.
E6	I found the installation guide and user manual really helpful.
Perceived Usefulness (Treatment groups only)	
All answers are on a 1-5 Likert scale (1 = strongly disagree; 5 = strongly agree)	
U1	I am confident that the tool helped me in locating all faulty cells.
U2	The tool helped me to improve the quality of the spreadsheet.
U3	Overall, I found the tool useful.
U4	Using this tool enables me to accomplish tasks more quickly.
U5	I would have found all the faults even without the tool.
U6	The tool was not responsive at all.
U7	I did not understand why certain cells were colored.
Usability (Treatment groups only)	
All answers are on a 1-5 Likert scale (1 = strongly disagree; 5 = strongly agree)	
S1	I think that I would like to use this tool frequently.
S2	I found the tool unnecessarily complex.
S3	I thought the tool was easy to use.
S4	I think that I would need the support of a technical person to be able to use this tool.
S5	I found the various functions in this tool were well integrated.
S6	I thought there was too much inconsistency in this tool.
S7	I would imagine that most people would learn to use this tool very quickly.
S8	I found the tool very cumbersome to use.
S9	I felt very confident using the tool.
S10	I needed to learn a lot of things before I could get going with this tool.
Comments and Suggestions	
C1	Do you have any comment or suggestions how to improve the tool? (Treatment groups) *Free text answer*
C2	What would have helped you to complete the task? *Free text answer*
C3	Any additional comments? (Control group) *Free text answer*

- *Perceived Tool Usefulness:* This block of questions (U1 to U7) was only asked to participants in the treatment groups. Seven questions inspired by Davis (1989) were asked to determine, for example, if the participants found the SmellChecker tool useful for finding faults or how confident they were that they found all faults. Again, 5-point Likert items were used.
- *Usability:* Participants in the treatment groups finally had to answer usability questions (S1 to S10). We relied on the ten-item System Usability Scale (SUS) for this measurement. As discussed above, our goal is to minimize risk that any observed effects of using SmellChecker are due to usability issues regarding the tool.
- Finally, all participants could leave free-text comments at the end of the questionnaire.

Detailed procedure. Fig. 7 shows on overview of the participants' tasks in our experiment. At the beginning of the study, after providing informed consent, the participants were asked to read the overview document and follow the steps described in it. The overview document contained an online link which referred them to an online directory where the additional materials were provided. The first task was to read the installation guide and install the plugin. After that, participants had to read the user manual and the spreadsheet description. Once these preparatory steps were done, they started the main task by opening the provided spreadsheet in the browser (with the plugin already activated).

Once the spreadsheet was opened, they had to click on the "Start" button in the plugin and begin with the debugging process. Both the treatment and control groups were instructed to look for faults in the given spreadsheet. Once they identified a potential fault, they had to use the comment functionality of MS Excel and add a comment to every cell of which they thought that it contained a faulty formula.[8] Once they felt that they had found all faults in the spreadsheet, they were instructed to click on the "Finish" button in the plugin to proceed to the post-task questionnaire. In treatment group B, as mentioned above, a warning about the potential limitations of the tool was shown.

The control group, as mentioned, only had the "Start" and "Finish" buttons available through the plugin. In the treatment

[8] Remember that we are interested only in faults in formulas.

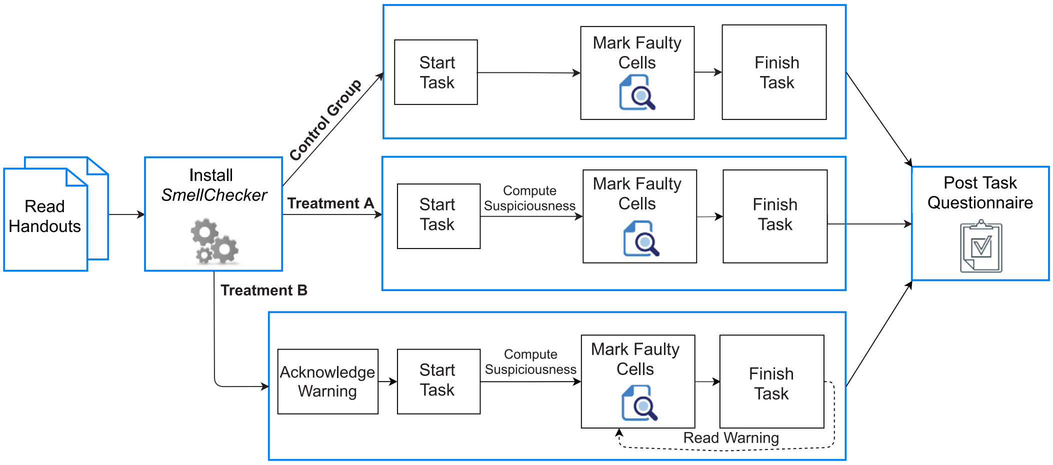


Fig. 7. Overall flow of the experiment.

groups, the SmellChecker debugging functionality was available to the participants, as shown in Fig. 3. Participants could use this functionality to compute the suspiciousness values of all formulas. Once these values were computed, cells that contained suspicious formulas were visually highlighted. Other than that, everything was identical as for the control group, i.e., participants also had to use the comment functionality of MS Excel to report faults. After clicking the “Finish” button, participants were forwarded to the post-task questionnaire.

4.3. Participants

We recruited study participants through the crowd-sourcing platform *Prolific*.[9] We ran a number of trials before we launched the experiment to gauge the reliability of the crowd workers on this platform. The results of these initial trials were generally very positive.

Overall, 59 participants successfully completed the task online through their personal computers. The participants were randomly assigned to the treatment and control groups. At the end, we had 20 participants in control and treatment group A and 19 participants in treatment group B (with awareness) group. On average, participants needed 11.57 min to complete the task. We paid £7 for each participant who completed the task through the crowd-sourcing platform.

To be allowed to participate in the study, participants were required to be proficient in English. Through our post-task questionnaire, we found the most frequent age group was between 18 and 25 years. The participant population was diverse in terms of the experience with spreadsheets, ranging from “less than one year” to “more than six years”.

We manually checked the fault identification behavior of the participants to ensure they did the task with the required care. We applied a conservative strategy and only disregarded one participant (from treatment group B) who apparently marked cells as being faulty in a random way, and in particular did not consider any cell that was highlighted by our tool.

5. Results

Our study led to the following observations, which *(i)* point to the existence of potential problems of tool over-reliance (RQ1), and *(ii)* indicate that raising the awareness of tool limitations can be beneficial (RQ2). Furthermore, we observed that participants generally found the SmellChecker tool both usable and useful (RQ3).

Table 2
Mean number and standard deviation of correctly identified faults.

Group	Mean	Std. dev.
Control	5.60	2.78
Treatment A	4.05	0.58
Treatment B	6.26	2.12

5.1. Over-reliance and awareness effects (RQ1 and RQ2)

In this section, we will first analyze how many of the faults were on average correctly identified and how much time participants spent on completing the task. We will then review in more detail which faults were found and to what extent participants marked correct formulas as faulty (*false positives*). Finally, we will examine the behavior of treatment group B, who received warnings about tool limitations, in some more depth.

Number of correctly identified faults. The main metric in our study is the number of faults that were correctly identified by participants as such. Table 2 shows the main outcomes in terms of the mean and the standard deviation for the treatment and control groups. Fig. 8 shows the distributions with the help of boxplots.

The differences between the groups, as shown in Table 2, are statistically significant according to an ANOVA analysis, with $p < 0.01$. Comparing the control group (without tool support) and treatment group A, we find that participants who were supported by the tool found *fewer* faults than those who did not have such support. This difference is also statistically significant according to a t-test, $p = 0.023$. The effect size is medium to large, with Cohen’s $d = 0.77$. This indicates that participants in group A were overly relying on the tool and did not look much for faults beyond what was highlighted by the tool.

Participants in treatment group B (with warning) found significantly more faults than those in group A ($p < 0.01$). The difference between the two group means is also statistically significant, with a large effect size ($d = 1.43$). They also only found slightly more faults than the control group participants. The difference between treatment group B and the control group did, however, *not* differ to a statistically significant extent ($p = 0.422$). The findings, therefore, suggest that the implemented mechanisms were effective to raise the awareness of study participants regarding the potential property of SmellChecker of not highlighting all existing faults. When the warning was present, participants in treatment group B apparently put more effort into finding faults in cells that were not highlighted by the tool.

[9] https://www.prolific.co.

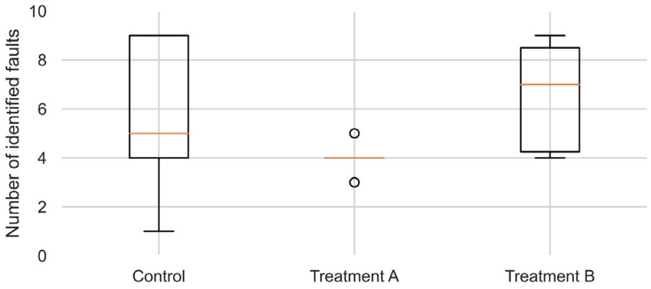


Fig. 8. Number of correctly identified faults in treatment and control groups.

Table 3
Mean Task completion time in minutes and standard deviations.

Group	Mean	Std. dev.
Control	11.28	4.16
Treatment A	6.39	2.73
Treatment B	10.85	5.56

Combined, these two observations indicate that a certain level of tool over-reliance existed in the study (RQ1) and that making users better aware of tool limitations can be a potential way to prevent such effects in practice (RQ2). As the boxplot for treatment group A reveals, in fact almost all participants found *exactly* the four faults that were highlighted by the tool. We will provide more details about the fault identification performance of individual participants later on, see also Fig. 10.

At first glance, one might conclude that the SmellChecker tool is not useful in practice, because it only helped to improve the fault detection performance to a small and non-significant extent. However, please note that the study was deliberately designed in a way that the spreadsheet contains several injected faults of which we know that the tool will not find them. The accuracy of the tool has already been evaluated previously (Koch et al., 2019) and is not the focus of this present paper. Here, the tool was deployed in an artificial situation where it probably performs worse than in a practical environment.

Moreover, previous research Panko (2008) has indicated that spreadsheet developers are often overconfident about their spreadsheets, and would probably not look much for non-obvious faults when they are not pointed to potential issues through a tool like SmellChecker (or are not asked to do so as in our study). A tool like SmellChecker may, therefore, be more beneficial in practical settings as it showed to be in our study, where participants were explicitly tasked to look for faults.

Task completion times. Table 3 shows the mean and the standard deviation of task completion time (minutes) for the treatment and control groups.

We observe that participants in treatment group A spent statistically significantly less time than the control group ($p < 0.01$). The effect size is large, with Cohen's $d = 1.38$. This supports our observation from above that participants in treatment group A overly relied on the tool. In contrast, participants in treatment group B, who received a warning, spent significantly more time on the task than those in group A ($p < 0.01$, $d = 1.02$), again indicating that raising awareness stimulated participants to look for faults beyond the highlighted cells. Ultimately, group B participants spent about the same time on the task as the control group, with differences being not statistically significant ($p = 0.8$).

Detailed fault identification performance. Fig. 9 shows for each of the nine faulty cells the percentage of study participants who had correctly identified this cell as faulty. The four left-most cells are those that were highlighted by the tool.

For both treatment groups, we can observe that the highlighted cells were clearly more often identified as faulty as the non-highlighted ones. Participants in treatment group A almost entirely focused on these highlighted cells and only in a few (nine) cases they marked other formulas as being faulty. Participants in treatment group B in contrast more often identified faults in areas that were not highlighted. Their fault identification performance was at about the same level as the control group. Interestingly, there were a few cases where participants in the treatment groups did *not* mark a cell as faulty even though the tool indicated a potential problem.[10]

Table 4
Mean (std. deviation) of true positives, false positives and precision and recall.

Group	True positives	False positives	Precision	Recall
Control	5.60 (2.78)	2.40 (2.41)	0.72	0.62
Treatment A	4.05 (0.58)	0.05 (0.21)	0.99	0.45
Treatment B	6.26 (2.12)	3.15 (3.13)	0.70	0.69

So far, we focused on the performance regarding the correct identification of the seeded faults. Table 4 now provides information about the true positives, as reported above, the false positives, i.e., about cells that were marked as faulty although being correct, as well as precision and recall. Raising awareness increases the number of faults found and recall, but it also increases the number of false positives, which negatively influences precision.

Table 4 not surprisingly shows that participants in treatment group A, who mostly focused on the highlighted cells, in most cases did not falsely mark any additional cells as faulty. However, participants who received a warning (treatment group B), while being able to correctly identify more faults than participants in other groups, also often falsely marked actually correct cells as being faulty. Note that false positives are also present in the control group, but to a lesser extent. Overall, the provided warning apparently not only led to the effect that participants in group B marked more cells as being faulty, but also to more false positives.

Generally, false positives of this type are often considered to be less critical than false negatives, i.e., missed faults. False positives may lead to extra debugging effort—until the developer finds out that the formula is actually correct—but will not lead to wrong calculations unless the developer changes the formula to become faulty.

Nonetheless, false positives of course remain undesired. We, therefore, looked closer at the data to investigate if a certain subgroup of participants is particularly susceptible to such problems. In particular, we hypothesized that a lack of experience, and corresponding insecurity, might lead to an "over-reaction" by participants in group B after receiving a warning. Nine participants in the control group and six in each treatment group had less than one year of experience with MS Excel. In Table 5 we, therefore, analyzed the true and false positives statistics (as well as precision and recall) only for participants who had more than one year of experience with MS Excel (38 of 59 participants).

The results not only show a small increase in true positives, i.e., more experienced participants found more faults, but also that the difference between treatment group B and the control group in terms of false positives disappears. We see these outcomes as an indication that the increased number of false

[10] Looking at the distribution of correctly identified faults, we have indications that the faults that were highlighted for the treatment groups incidentally were slightly easier to find, as can be seen from the distribution for the control group in Fig. 9. This, however, does not affect the outcomes of our research.

Table 5
Means (std. deviation) of true positives, false positives, and precision and recall for participants having more than one year of experience with MS Excel.

Group	True positives	False positives	Precision	Recall
Control	6.10 (2.38)	2.60 (1.80)	0.74	0.67
Treatment A	4.14 (0.63)	0.07 (0.25)	0.98	0.46
Treatment B	7.07 (1.81)	2.53 (2.27)	0.75	0.78

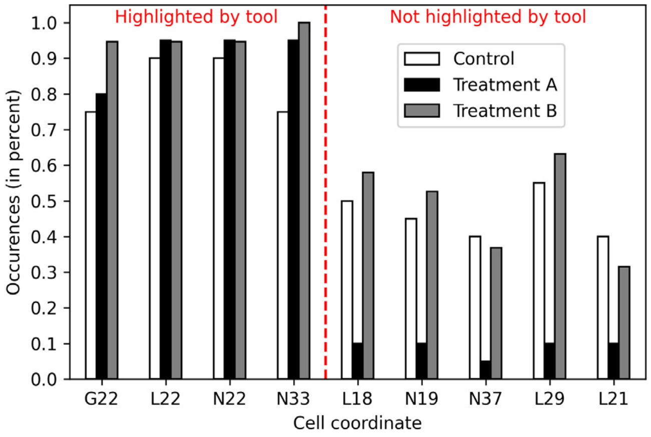


Fig. 9. Percentage of study participants in each group who correctly identified the actual faulty cells. The four left-most cells are those highlighted by the SmellChecker tool.

positives observed in Table 4 may at least partly result from to the behavior of participants with lower experience.[11] Additional research including a larger set of participants with different experience levels, however, seems required to be able to make reliable conclusions from such subgroup analyses.

Fig. 10 finally illustrates the true positives and false positives for each participant as a stacked bar chart. Confirming our observations from above, participants in treatment group A mainly relied on the tool outcome, while participants in treatment group B also investigated other cells.

Effects of raising awareness with warnings. Participants in treatment group B, as mentioned above, were pointed to potential tool limitations by the plugin two times: at the beginning of the task and when they clicked on "Finish" to complete the task. Moreover, the provided user manual, which participants *of both treatment groups* had to read at the beginning, also mentioned this potential limitation. Apparently, as we see from the experiment, the information provided in the manual went unnoticed or was forgotten by the participants during the study.

An analysis of the logs of the behavior of participants in treatment group B reveals that 57% of the participants went back to the spreadsheet after they were shown the warning before finishing the task. From the logs we can, however, not know if the other participants—those who did not go back—were aware about the tool limitations from the first prompt and confident that they found all faults, or if they simply ignored the warning. In any case, the results indicate *(i)* that explicit prompts about tool limitations *during tool usage* are effective, and *(ii)* that warnings provided in manuals, even when these manuals are read, may have a low impact on user awareness.

5.2. Tool usability and perceived usefulness (RQ3)

As mentioned previously, it is important to make sure that potentially observed undesired effects of the SmellChecker tool—such as lower fault identification rates for one treatment group—are not due to general usability problems.

Usability. To assess the *usability* of our tool on an absolute scale, we used the System Usability Scale as one element of our post-task questionnaire. Fig. 11 shows the average answers of the participants to the questionnaire items for each treatment group.

Overall, participants were largely positive about various usability aspects, leading to a SUS score (Brooke, 1995) of 78.25 for treatment group A and 73.42 for treatment group B. One benefit of the System Usability Scale is that it can be interpreted at an absolute scale. Commonly, tools that obtain a SUS score of 68 or higher are considered to be "good" (Orfanou et al., 2015; Sharfina and Santoso, 2016; Harrati et al., 2016). Since SmellChecker has a higher SUS score, we are confident that our tool did not introduce any barriers to the fault identification task.

Looking at the different treatment groups, the responses by the participants in both groups are highly consistent. A difference can be observed for question S6. Here, participants in group B considered the tool to be slightly more inconsistent, although on a generally low level of inconsistency. This inconsistency perception apparently stems from the additional warnings that were presented by the tool to participants of treatment group B.

Perceived usefulness. The responses to the questionnaire items relating to the *usefulness* of the SmellChecker tool are shown in Fig. 12. Looking at the positively framed questions about usability (U1 to U4), the average response across treatment groups was around or above 4 (on the 5-point scale), except for the responses for question U1 by treatment group B. Generally, however, differences between the treatment groups were again small. Likewise, for the negatively framed questions (U6 and U7), the average responses were below 2, again indicating that the tool is perceived as useful by the study participants.

Regarding question U1 ("I am confident that tool helped me in locating all faulty cells."), there is a pronounced difference between the treatment groups. Participants in group A were very confident in having found all faults, while in reality they on average missed half of them. In some ways, this corroborates previous findings made by Panko (2008, 2000) on overconfidence patterns among of spreadsheet developers. Participants in treatment group B, on the other hand, were less confident, with an average response of around 3 on the five-point scale. The presented warnings, therefore, seem to be effective to make users aware of the problem that they might have missed some faults.

Question U5 ("I would have found all the faults even without the tool."), finally, is a special case, as it is there to make an estimate of the participant's fault identification without the tool. The average response to the question is 2.85. Comparing the responses to those given to question U1 indicates that participants saw value in the tool.

5.3. Additional analyses

In addition to usefulness and perceived usability, we gauged additional user perceptions regarding the overall task through questions E1 to E6. Remember that these questions were also asked to participants in the control group. The average responses by participants in each group are shown in Fig. 13.

The obtained results indicate that the task was perceived to be *doable* and not too time-consuming for all participants. This supports that our selection of the spreadsheet for the study, as

[11] In some ways, this finding is in line with the observations made by Xia et al. (2016).

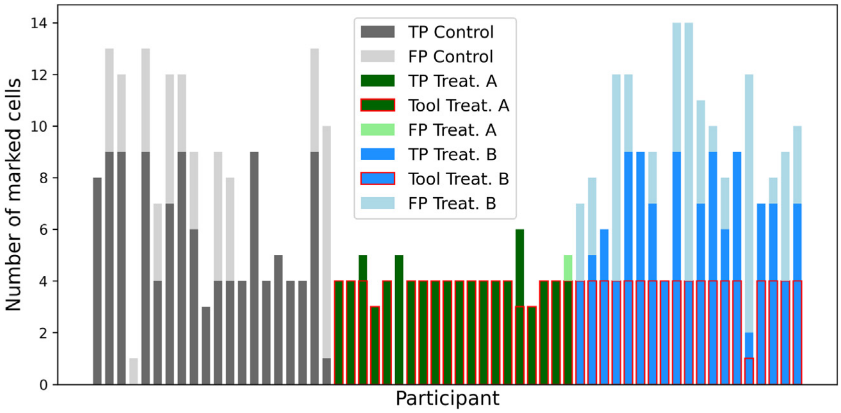


Fig. 10. Number of cells each participant marked as faulty. FP (false positives) are marked cells that are actually correct. TP (true positives) are marked cells that are actually faulty. The red frames show the marked faults that were also indicated by the tool.

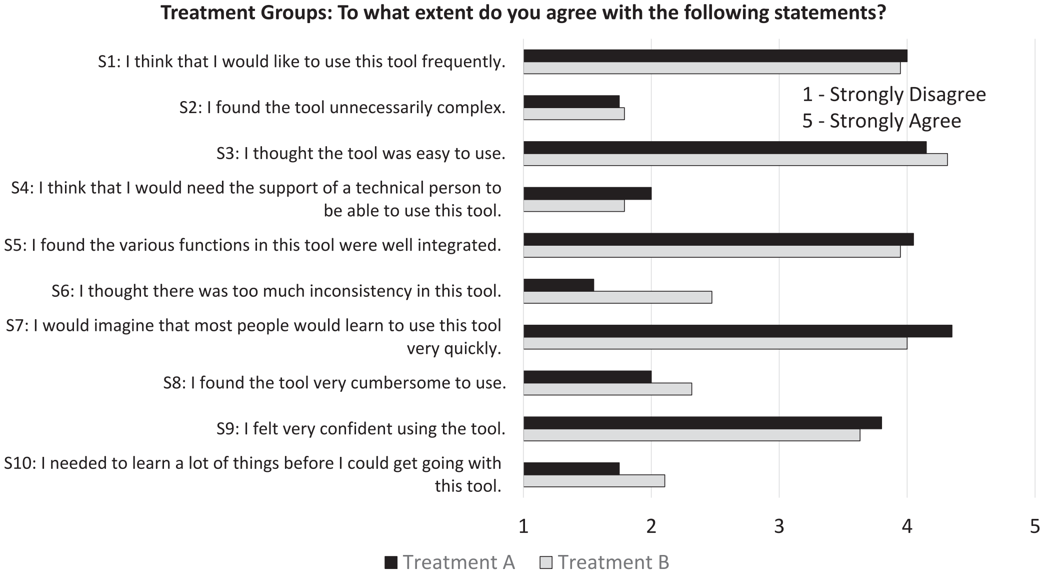


Fig. 11. System usability scale results (post-task questionnaire).

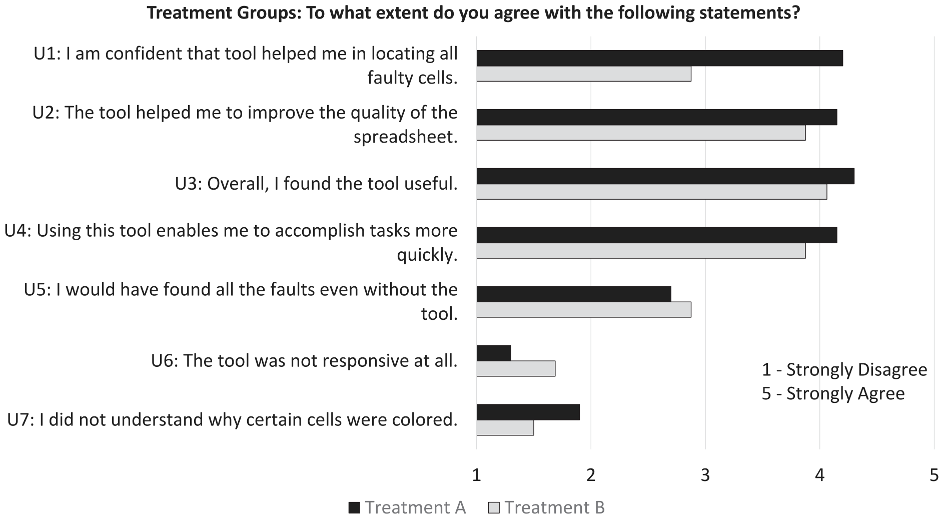


Fig. 12. Perceived usefulness (post-task questionnaire).

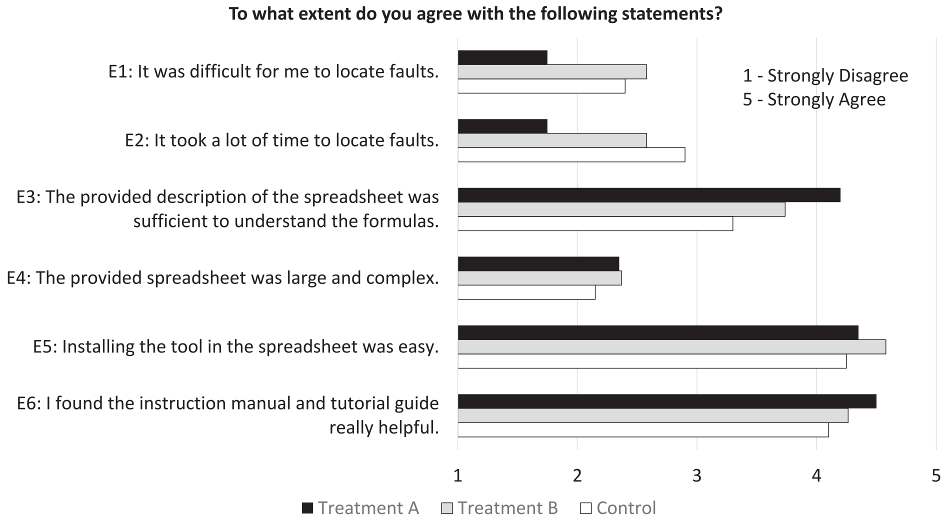


Fig. 13. Perceived effort (post-task questionnaire).

discussed in Section 4.2, was appropriate. Furthermore, participants in all groups found the provided material (installation instructions, user manual) useful.

However, a difference between treatment groups and the control can be found in terms of perceived effort and difficulty. Treatment group A, i.e., those who apparently relied too much on the tool and only identified the highlighted faults, both found the task localization less difficult (E1) and requiring less effort (E2) than participants in the other groups. At the same time, they on average found that the information provided to understand the spreadsheet was sufficient compared to other participants (E3). Remember that these questions were asked *after* the participants had completed the fault localization tasks. Thus, the process of performing the task influenced the perception of the task itself. The deviation from the other groups may in fact indicate that the tool provoked a certain *overconfidence* at the user's side, as mentioned above.

6. Threats to validity

Like for all experimental studies, there may be different threats to internal and external validity, which we discuss in this section.

Internal validity. Regarding internal validity, we followed established scientific practices to minimize the chance of confounding, i.e., that there are other explanations for the observed cause-and-effect relationship. In our experimental design, everything but the different forms of tool support and the corresponding manuals in the treatment groups was therefore kept static: participants were given the same spreadsheet, the same explanation of its content, and they performed the same main task of labeling those cells that they considered faulty. The spreadsheet that was used in the study was selected according to a number of criteria discussed above. We do not identify any source of bias that the particular spreadsheet may have caused. Finally, the study participants were also randomly assigned to treatment and control groups.

Given that we conducted a crowd-sourced study, there might be certain participant-related aspects that influence the observed outcomes. In particular, in terms of task-completion time, we cannot know if some participants were interrupted during the task. As a countermeasure, we, therefore, conducted an additional analysis, in which we considered all data points as outliers that were more than two standard deviations away from the mean, which led to the removal of 6.7% of the data. Our analysis however showed that the results did not change with respect to statistical significance when such outliers were removed.

Finally, we are also confident that the used post-task questionnaire did not introduce any bias. All items in the questionnaire were adapted from the literature, and we furthermore relied on the widely used System Usability Scale to gauge the perceived usability of the SmellChecker tool.

External validity. Regarding the generalizability of our findings, different threats exist as well. First, the participant population was limited to crowd workers on the platform Prolific. Since the majority of participants had a certain level of experience with spreadsheets, we are confident that the participants are representative for at least a subset of typical spreadsheet users. Moreover, due to the random assignment of participants to treatment and control groups, we do not expect bias regarding the obtained results. Note also that we did not allow anyone to participate in the study more than once. Regarding the number of study participants, we trust that the findings are reliable with the current sample. A repeated study with more participants may nonetheless help to further increase the reliability of the findings. Such a larger study may also help us making additional analyses regarding different subgroups, e.g., if there are strong differences between novice users and experienced users.

So far, we have only used one particular spreadsheet in our experiment, and we thus cannot conclude with certainty that we will observe very similar phenomena for largely different spreadsheets. As discussed above, we have however tried to make sure in different ways that the used spreadsheet has characteristics that are similar to those of many spreadsheets that are commonly used in organizations. As such, we believe that the obtained general observations are tied to the specific spreadsheet that was chosen in the study. Note also that the spreadsheet and the injected faults were specifically designed for investigating the phenomenon of over-reliance, where participants mostly focus on cells that are highlighted by the tool. As such, what probably matters more is if cells are highlighted or not, and the specifics of the underlying spreadsheets may thus be only secondary for questions of over-reliance.

One may also fear that the observed findings are limited to the specific debugging tool, i.e., SmellChecker, that we used in our experiment and to the specific way the suspicious cells are brought to the attention of the participants. However, SmellChecker actually follows commonly used means for providing debugging information to users, i.e., highlighting cells or code fragments. We, therefore, expect that similar results would be obtained also for other tools that rely on comparable user interface mechanisms. Regarding the phenomena related to over-reliance, we thus believe that similar debugging tools would lead to similar effects. Remember also that the main design of our study is independent from the way the underlying suspiciousness values are computed. Nonetheless, additional studies are needed to further validate that our findings generalize when different underlying spreadsheets with different faults are used.

Finally, we have good indications from the post-task questionnaire that participants found the SmellChecker tool useful, which supports that our study is realistic. Nonetheless, study settings like ours always remain artificial to some extent, as participants are not additionally rewarded, e.g., through an additional compensation, when they find more faults.[12]

7. Summary, implications and outlook

Spreadsheets are omnipresent in organizations, and faults in spreadsheets can have substantial negative consequences. More research is, therefore, required to support end user developers during the construction and maintenance of spreadsheets. Recent works in quality assurance for spreadsheets very successfully rely on heuristics or fault statistics and the corresponding tools are designed to point developers to potential problems in the spreadsheets. Due to their heuristic nature, these tools are not exact, i.e., they might not be able to highlight all existing faults or they might mark correct parts as being potentially faulty. Therefore, certain dangers arise if spreadsheet developers rely entirely on such heuristics-based tools. In particular, they might focus too much or solely on potential problems pinpointed by the tools and pay less attention to other parts of the spreadsheet, a phenomenon we refer to as over-reliance in this work.

In this paper, we have conducted a controlled experiment with a novel tool for fault identification in spreadsheets called SmellChecker. While the study participants found the tool both useful and usable, our results suggest that the described phenomenon of tool over-reliance actually exists and may lead to limited fault identification performance. At the same time, we found that increasing the users' awareness of the heuristic nature of the tool can help to address this issue.

The findings of this research—which to our knowledge is the first on this topic in the domain of fault identification in spreadsheets—have several practical implications. First of all, it is important for tool developers to understand that tool over-reliance phenomena may occur in practice. Therefore, it is crucial to communicate to developers what the tool is capable of and which types of problems it might not find. Moreover, our study shows that raising awareness regarding tool limitations may have to be done in a very explicit way. Highlighting the limitations in a manual or during the installation was not effective in our study. Respective prompts *within* the application itself, in contrast, proved to be helpful to raise user awareness. In practice, of course, the question remains when and how often such warnings should be displayed, as too many prompts may distract and annoy users sooner or later.

Finally, from the perspective of tool development, we found that SmellChecker, even though being a research prototype, is promising from the perspective of end user acceptance and usability. In particular, the learning effort for users seems relatively low compared, for example, to our own previous approaches based on test cases and model-based diagnosis techniques (Jannach and Schmitz, 2016). In terms of the underlying approach, note that the machine-learning technique used in the background has successfully found the one true fault in the original spreadsheet and three out of eight artificially injected faults. We deliberately injected faults into the spreadsheet that the tool does not detect to be able to study potential effects of over-reliance. Additional heuristics may, however, be implemented in this approach to also highlight those faults.

Following our discussions of research limitations in Section 6, in our future work, we plan to conduct additional experiments with different spreadsheets and participants with diverse backgrounds, in particular in terms of their experience with spreadsheets. Furthermore, we will empirically evaluate alternative ways of making users aware of potential tool limitations (e.g. check boxes, message dialogues) and when is the best time to make users aware. In addition to such studies, we plan to improve the fault detection mechanisms of SmellChecker, e.g., by adding new metrics that can help to more reliably detect various forms of range errors.

CRediT authorship contribution statement

Adil Mukhtar: Methodology, Software, Formal Analysis, Investigation, Resources, Data curation, Visualization, Writing – original draft, Writing – review & editing. **Birgit Hofer:** Software, Validation, Writing – original draft, Writing – review & editing, Visualization. **Dietmar Jannach:** Conceptualization, Methodology, Writing – original draft, Writing – review & editing, Supervision, Funding acquisition. **Franz Wotawa:** Conceptualization, Methodology, Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

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

References

Abraham, R., Erwig, M., 2009. Mutation operators for spreadsheets. IEEE Trans. Softw. Eng. 35 (1), 94–108.

Abreu, R., Hofer, B., Perez, A., Wotawa, F., 2015. Using constraints to diagnose faulty spreadsheets. Softw. Qual. J. 23 (2), 297–322.

Abreu, R., Riboira, A., Wotawa, F., 2012. Debugging spreadsheets: A CSP-based approach. In: International Workshop of Program Debugging (IWPD) At ISSREW 2012, pp. 159–164.

Abreu, R., Zoeteweij, P., Golsteijn, R., van Gemund, A.J., 2009. A practical evaluation of spectrum-based fault localization. J. Syst. Softw. 82 (11), 1780–1792.

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

Baniassad, E.L., Zamprogno, L., Hall, B., Holmes, R., 2021. Stop the (autograder) insanity: regression penalties to deter autograder overreliance. In: SIGCSE '21: The 52nd ACM Technical Symposium on Computer Science Education. pp. 1062–1068.

Barowy, D.W., Berger, E.D., Zorn, B.G., 2018. ExceLint: automatically finding spreadsheet formula errors. In: Proceedings of the ACM on Programming Languages, vol. 2, pp. 148:1–148:26.

Blackett, C., 2021. Human-centered design in an automated world. In: Russo, D., Ahram, T., Karwowski, W., Di Bucchianico, G., Taiar, R. (Eds.), Intelligent Human Systems Integration 2021. pp. 17–23.

Brooke, J., 1995. SUS: A quick and dirty usability scale. Usability Eval. Ind. 189.

Bryant, S.L.J., 2011. The Dangers of an Over-Reliance on Technology (Master's thesis).

Burg, B., Bailey, R., Ko, A.J., Ernst, M.D., 2013. Interactive record/replay for web application debugging. In: Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology. pp. 473–484.

[12] Providing additional compensation for more identified faults might also have an effect on the results in terms of false positives.

Catal, C., 2011. Software fault prediction: A literature review and current trends. Expert Syst. Appl. 38 (4), 4626–4636.

Ceballos, R., Abreu, R., Varela-Vaca, A.J., Gasca, R.M., 2019. Model-based software debugging. In: Fault Diagnosis of Dynamic Systems: Quantitative and Qualitative Approaches. pp. 365–387.

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

Cunha, J., Fernandes, J.P., Martins, P., Mendes, J., Saraiva, J., 2012. SmellSheet detective: A tool for detecting bad smells in spreadsheets. In: IEEE Symposium on Visual Languages and Human-Centric Computing, pp. 243–244.

Cunha, J., ao Paulo Fernandes, J., Ribeiro, H., ao Saraiva, J., 2012. Towards a catalog of spreadsheet smells. In: 12th International Conference on Computational Science and Its Applications, pp. 202–216.

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

Dou, W., Xu, C., Cheung, S.C., Wei, J., 2017. CACheck: Detecting and repairing cell arrays in spreadsheets. IEEE Trans. Softw. Eng. 43 (3), 226–251.

Ducassé, M., 1993. A pragmatic survey of automated debugging. In: AADEBUG. In: Lecture Notes in Computer Science, vol. 749, pp. 1–15.

Getzner, E., Hofer, B., Wotawa, F., 2017. Improving spectrum-based fault localization for spreadsheet debugging. In: 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS), pp. 102–113.

Goddard, K., Roudsari, A., Wyatt, J.C., 2011. Automation bias: a systematic review of frequency, effect mediators, and mitigators. J. Am. Med. Inf. Assoc. 19 (1), 121–127.

Goddard, K., Roudsari, A., Wyatt, J.C., 2014. Automation bias: Empirical results assessing influencing factors. Int. J. Med. Inf. 83 (5), 368–375.

Harrati, N., Bouchrika, I., Tari, A., Ladjailia, A., 2016. Exploring user satisfaction for e-learning systems via usage-based metrics and system usability scale analysis. Comput. Hum. Behav. 61, 463–471.

Hermans, F., Pinzger, M., van Deursen, A., 2012. Detecting code smells in spreadsheet formulas. In: International Conference on Software Maintenance, pp. 409–418.

Hermans, F., Pinzger, M., van Deursen, A., 2012. Detecting and visualizing interworksheet smells in spreadsheets. In: 34th International Conference on Software Engineering, pp. 441–451.

Hofer, B., Jannach, D., Koch, P., Schekotihin, K., Wotawa, F., 2021. Product metrics for spreadsheets - A systematic review. J. Syst. Softw. 175.

Jannach, D., Engler, U., 2010. Toward model-based debugging of spreadsheet programs. In: 9th Joint Conference on Knowledge-Based Software Engineering, pp. 252–262.

Jannach, D., Schmitz, T., 2016. Model-based diagnosis of spreadsheet programs: a constraint-based debugging approach. Autom. Softw. Eng. 23 (1), 105–144.

Jannach, D., Schmitz, T., Hofer, B., Schekotihin, K., Koch, P.W., Wotawa, F., 2019. Fragment-based spreadsheet debugging. Autom Softw Eng 26 (1), 203–239.

Jannach, D., Schmitz, T., Hofer, B., Wotawa, F., 2014. Avoiding, finding and fixing spreadsheet errors - a survey of automated approaches for spreadsheet QA. J. Syst. Softw. 94, 129–150.

Ko, A.J., Myers, B.A., 2004. Designing the whyline: a debugging interface for asking questions about program behavior. In: Proceedings of ACM CHI 2004 Conference on Human Factors in Computing Systems. pp. 151–158.

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

Koch, P.W., Hofer, B., Wotawa, F., 2016. Static spreadsheet analysis. In: 2016 IEEE International Symposium on Software Reliability Engineering Workshops, ISSRE Workshops 2016, 2016. pp. 167–174.

Koch, P., Schekotihin, K., Jannach, D., Hofer, B., Wotawa, F., 2019. Metric-based fault prediction for spreadsheets. IEEE Trans. Softw. Eng. (10), 1.

Li, D., Wang, H., Xu, C., Shi, F., Ma, X., Lu, J., 2019. WARDER: Refining cell clustering for effective spreadsheet defect detection via validity properties. In: 19th International Conference on Software Quality, Reliability and Security, pp. 139–150.

Li, D., Wang, H., Xu, C., Zhang, R., Cheung, S.-C., Ma, X., 2019. SGUARD: A feature-based clustering tool for effective spreadsheet defect detection. In: 34th IEEE/ACM International Conference on Automated Software Engineering, pp. 1142–1145.

Li, X., Zhu, S., d'Amorim, M., Orso, A., 2018. Enlightened debugging. In: 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE), pp. 82–92.

Lin, Y., Sun, J., Xue, Y., Liu, Y., Dong, J., Feedback-based debugging. In: Proceedings of the 39th International Conference on Software Engineering. ICSE '17. pp. 393–403.

Lyell, D., Coiera, E., 2016. Automation bias and verification complexity: a systematic review. J. Am. Med. Inf. Assoc. 24 (2), 423–431.

Ma, W., Chen, L., Zhou, Y., Xu, B., 2016. Do we have a chance to fix bugs when refactoring code smells? In: 2016 International Conference on Software Analysis, Testing and Evolution (SATE), pp. 24–29.

Machado, T., Gopstein, D., Nealen, A., Nov, O., Togelius, J., 2018. AI-assisted game debugging with cicero. In: 2018 IEEE Congress on Evolutionary Computation (CEC), pp. 1–8.

Marc, II, F., Rothermel, G., 2005. The EUSES spreadsheet corpus: a shared resource for supporting experimentation with spreadsheet dependability mechanisms. ACM SIGSOFT Softw. Eng. Notes 30 (4), 1–5.

Matthew, G., 2019. Understanding human over-reliance on technology. P T 44, 320–375.

McConnell, S., 2004. Code Complete - A Practical Handbook of Software Construction, second ed..

Mosier, K.L., Skitka, L.J., 1996. Human decision makers and automated decision aids: Made for each other? In: Human Factors in Transportation. Automation and Human Performance: Theory and Applications. pp. 201–220.

Mosier, D., 2019. Humans and automated decision aids: A match made in heaven? In: Mouloua, M. (Ed.), Human Performance in Automated and Autonomous Systems. pp. 1–24.

Noble, A.M., Miles, M., Perez, M.A., Guo, F., Klauer, S.G., 2021. Evaluating driver eye glance behavior and secondary task engagement while using driving automation systems. Accid. Anal. Prev. 151.

Orfanou, K., Tselios, N., Katsanos, C., 2015. Perceived usability evaluation of learning management systems: Empirical evaluation of the system usability scale. Int. Rev. Res. Open Distance Learn. 16, 227–246.

Panko, R.R., Spreadsheet errors: What we know; what we think we can do. In: Proceedings EuSpRIG, 2000.

Panko, R.R., 2008. Reducing overconfidence in spreadsheet development. CoRR arXiv:0804.0941.

Panko, R.R., Port, D.N., 2013. End user computing: The dark matter (and dark energy) of corporate IT. J. Organ. End User Comput. 25 (3).

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

Ruthruff, J., Burnett, M., Rothermel, G., 2006. Interactive fault localization techniques in a spreadsheet environment. IEEE Trans. Softw. Eng. 32 (4), 213–239.

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

Schmitz, T., Jannach, D., 2017. An AI-based interactive tool for spreadsheet debugging. In: IEEE Symposium on Visual Languages and Human-Centric Computing, pp. 333–334.

Schmitz, T., Jannach, D., Hofer, B., Koch, P., Schekotihin, K., Wotawa, F., 2017. A decomposition-based approach to spreadsheet testing and debugging. In: Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC '17), Raleigh, North Carolina, USA.

Sharfina, Z., Santoso, H., 2016. An Indonesian adaptation of the system usability scale (SUS). In: 2016 International Conference on Advanced Computer Science and Information Systems (ICACSIS), pp. 145–148.

Singh, R., Livshits, B., Zorn, B., 2017. Melford: Using Neural Networks to Find Spreadsheet Errors. Microsoft Technical Report MSR-TR-2017-5, Microsoft.

Soremekun, E.O., Kirschner, L., Böhme, M., Zeller, A., 2021. Locating faults with program slicing: An empirical analysis. CoRR arXiv:2101.03008.

Stefanović, D., Nikolić, D., Dakić, D., Spasojević, I., Ristić, S., Static code analysis tools: A systematic literature review, in: Annals of DAAAM and Proceedings of the International DAAAM Symposium, vol. 31, 2020, pp. 565–573.

Sung, H.-N., Jeong, D.-Y., Jeong, Y.-S., Shin, J.-I., 2015. The relationship among self-efficacy, social influence,performance expectancy, effort expectancy, and behavioral intention in mobile learning service. Int. J. U- E- Serv. Sci. Technol. 8, 197–206.

Wong, W.E., Gao, R., Li, Y., Abreu, R., Wotawa, F., 2016. A survey on software fault localization. IEEE Trans. Softw. Eng. 42 (8), 707–740.

Xia, X., Bao, L., Lo, D., Li, S., 2016. "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), pp. 267–278.

Xie, X., Liu, Z., Song, S., Chen, Z., Xuan, J., Xu, B., 2016. Revisit of automatic debugging via human focus-tracking analysis. In: Proceedings of the 38th International Conference on Software Engineering. pp. 808–819.

Xie, X., Xu, B., 2021. Essential Spectrum-Based Fault Localization.

Xu, L., Wang, S., Dou, W., Yang, B., Gao, C., Wei, J., Huang, T., 2018. Detecting faulty empty cells in spreadsheets. In: IEEE International Conference on Software Analysis, Evolution and Reengineering, pp. 423–433.

Adil Mukhtar is a scientific project assistant and Ph.D. candidate at Graz University of Technology, Austria. He received his Master's Degree in Computer Science from Lahore University of Management Technology. His research primarily focuses on fault localization, spreadsheet debugging, diagnostic systems and natural language processing based on AI enabled techniques.

Birgit Hofer works as researcher at Graz University of Technology. She received a Ph.D. degree in Computer Science (2013) and a Master's degree (2009) from the same University. Her main research interests are automatic fault localization and correction in imperative and object oriented software and spreadsheets, with a particular focus on spectrum-based fault localization, model-based debugging, and genetic programming approaches.

Dietmar Jannach is a full professor of Information Systems at the University of Klagenfurt, Austria. His main research area lies in the application of artificial intelligence technology to practical problems, with a particular focus on recommender systems in e-commerce, AI-based testing and debugging of spreadsheets, and on engineering problems of knowledge-intensive systems.

Franz Wotawa received an M.Sc. and a Ph.D. from Vienna University of Technology in 1994 and 1996 respectively. He is currently a professor of software engineering at Graz University of Technology. His research interests include model based and qualitative reasoning, theorem proving, mobile robots, verification and validation, and software testing and debugging.