Spreadsheet Guardian: An Approach for Protecting Semantic Correctness throughout the Evolution of Spreadsheets

Daniel Kulesz[1], Verena Käfer[1], and Stefan Wagner[1]

[1]Software Engineering Group, Institute of Software Technology, University of Stuttgart, Germany

Abstract

Spreadsheets are powerful tools which play a business-critical role in many organizations. However, many bad decisions taken due to faulty spreadsheets show that these tools need serious quality assurance. Furthermore, while collaboration on spreadsheets for maintenance tasks is common, there has been almost no support for ensuring that the spreadsheets remain correct during this process. We believe that spreadsheet users should be supported in putting test rules into their spreadsheets from which subsequent users can profit.

We have developed an approach named Spreadsheet Guardian which separates the specification of spreadsheet test rules from their execution. By automatically executing user-defined test rules, our approach is able to detect semantic faults. It also protects all collaborating spreadsheet users from introducing faults during maintenance, even if only few end-users specify test rules. We implemented Spreadsheet Guardian as an add-in for Microsoft Excel.

We evaluated Spreadsheet Guardian in two empirical evaluations with 29 typical end-users and 42 computer science students. The results indicate that our approach to specifying spreadsheet test rules is easy to learn and to apply. Furthermore, participants with spreadsheets "protected" by Spreadsheet Guardian recognize more faults during maintenance of complex spreadsheets and are more realistic about the correctness of their spreadsheets than participants who employ only "classic", non-interactive test rules based on static analysis techniques. Hence, we believe Spreadsheet Guardian can be of use for any business-critical spreadsheet.

1 Introduction

Although spreadsheets have existed for more than 35 years, their popularity remains unbroken: it is assumed that there is a population of millions of spreadsheet users and that billions of spreadsheets exist [44]. Since the 1980s, spreadsheets play a critical role in most businesses – they are used for accounting [29], data analysis [40], as decision support systems [46] and for many other purposes [12].

1.1 Motivation

Using untested spreadsheets can be risky. Over the last years, dozens of cases were disclosed where faulty spreadsheets caused severe financial and reputational damage [1], with the recent Reinhard-Rogoff being one of the most drastic examples [16, 42]. Despite these issues and some conceptional shortcomings of spreadsheets, no other technology has managed to supplant spreadsheets on a broader scale. For these reasons – and also due to the number of already existing spreadsheets – numerous scientists have investigated the creation, detection and prevention of faults in spreadsheets [19, 32]. But also governmental institutions have taken action: several recent laws like the Sarbanes Oaxley Act 404, Basel III or Solvency II demand proof from organizations that all artifacts which contribute to financial calculations have been thoroughly inspected – which also includes spreadsheets used in the process.

1.2 Problem Statement

Maintenance activities in spreadsheets typically are not clearly separated from "normal" usage activities. And because spreadsheet users are often overconfident about the correctness of their spreadsheets [34], they typically do not undertake quality assurance activities during and after maintenance activities.

The fact that spreadsheet users often collaborate and share their undocumented and untested spreadsheets with co-workers (we will discuss this in detail in section 3) further increases the risk of running into the previously mentioned problems.

1.3 Research Objective

Since spreadsheets have many similarities with traditional programs, it seems worthwhile considering to port proven insights from software engineering to the world of spreadsheets. However, spreadsheets require dealing with collaborative maintenance activities in often uncontrolled arbitrary environments. Therefore, the main objective of this work was to find a concept which can increase semantic correctness in such settings while being unintrusive and keeping the need for changing users' habits as little as possible.

1.4 Contributions

We have developed an approach named *Spreadsheet Guardian* which ports ideas from unit tests and continuous software engineering to the world of spreadsheets. Spreadsheet Guardian allows typical end-users to specify their own test rules which are then continuously and automatically executed in the background. Both the specification of test rules and the reporting of findings is tightly integrated into Microsoft Excel by an accompanying add-in.

In previous work [26], we have presented our technique for creating test rules and its implementation in our tool. This work provides three main contributions which extend our previous work on this path: (i) it proposes an approach for spreadsheet maintenance activities that separates test specification from execution, (ii) it provides a theory for the approach and (iii) it presents results from two empirical evaluations of the approach.

2 Basics and Terminology

To make it easier to understand Spreadsheet Guardian and related work, we will introduce some basics and specify our understanding of certain terms in this context.

2.1 Spreadsheet Users and Activities

In contrast to traditional programs, the spreadsheet paradigm does not distinguish between developers and end-users. Therefore, we also use the general term "spreadsheet user", but differentiate between various activities of spreadsheet users:

Development activities refers to creating, modifying and deleting cells that contain formulas.

Filling activities refers to filling in data into cells which do not contain formulas.

Viewing activities refers to reading and interpreting spreadsheets without changing cell contents (neither data nor formulas) for the purpose of getting information.

Inspection activities refers to reading and interpreting spreadsheets without changing cell contents (neither data nor formulas) for the purpose of understanding the spreadsheet or reviewing it, i.e. checking its correctness.

Depending on the activity, spreadsheet users need different levels of support as Hermans et. al have concluded after surveying 47 employees in a asset management company [14].

2.2 Spreadsheet Errors and Anomalies

In spreadsheet literature, the ambiguous term "spreadsheet error" is used often and several taxonomies for classifying spreadsheet errors have been proposed. Powell et al. [38] mention classification possibilities based on:

- Cause (typing error, copy-paste error, ...)
- Effect (wrong result, impeded maintainability, ...)
- Type (wrong formula, redundant input fields, ...)

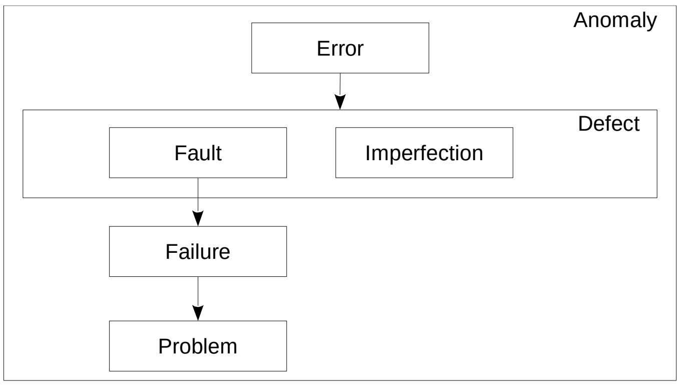


Figure 1: Taxonomy of spreadsheet anomalies

- Development stage (conception, implementation, use)

These possibilities make it clear that the understanding of the term "spreadsheet error" can be very ambiguous. To avoid this issue, we adopted the notion of an "anomaly" (that is defined in IEEE Std. 1044:2009 for traditional software) for spreadsheets (see Figure 1). In the following, we would like to explain the various types of anomalies using the following example: Assuming we have a spreadsheet with the following formula in cell C1:
$= B2 + B3 + B4 + B4 + B5 + B6 + B7 + B8$

Error refers to human errors. In the above example, a human could have clicked twice on the cell B4 unintentionally when constructing the formula.

Defect refers to any undesired data, formula or formatting in a spreadsheet – such as the formula given in the example.

Fault refers to a defect that can have a quantitative impact, i.e. on the correctness of the spreadsheet. This would be the case if, in the example above, it was not intended for B4 to be included twice in the result.

Imperfection refers to a defect that can have a purely qualitative (not quantitative) impact, i.e. on the usability or the maintainability of the spreadsheet. In the example above, this would be the case if summing up B4 twice was actually intended. Also, one could argue that using the SUM-function would be more appropriate here, as well as placing the result of the sum in B9 instead of C1.

Failure refers to a spreadsheet which is not correct, i.e. a spreadsheet with a fault that actually has a quantitative impact. In the above example, if summing up B4 twice was not intended and B4 contained a non-zero value, the resulting value in C1 would very likely be wrong (if not hidden by another fault, that is where several faults balance each other and the result is still correct).

Problem refers to a negative impact in reality due to a failure in a spreadsheet. In the example above, if the wrong business decision was taken due to B4 being summed up twice, this would be a problem.

The arrows in Figure 1 mean that there is a "can cause" relation between the anomalies, i.e. an error can cause a defect, a fault can cause a failure and a failure can cause a problem.

2.3 Spreadsheet Inspections

The presence of anomalies is an indicator for inadequate quality. However, software can only have reasonable quality because achieving "perfect" quality is not possible [28]. According to Frühauf et. al., the usual way to achieve reasonable quality is by issuing organizational, constructive and analytical steps. For traditional software, this means that a "systematic approach using proven software construction principles has to be followed". The purpose of inspections is to "detect deviances from these principles" while "organizational steps provide the basic environment to do that" [11, p. 20].

Unlike professionally crafted traditional software, spreadsheets are usually developed unsystematically, not according to proven principles and by laymen. Therefore, it is not surprising that a meta study by

Panko found that on average 94% of the spreadsheets inspected in several studies contained anomalies [35]. Therefore, the goal of diligent spreadsheet inspections is not to "inspect-in" adequate quality but to detect dangerous anomalies and to take such spreadsheets offline until they are sanitized.

On previous work [26], we reviewed both spreadsheet inspection approaches typically encountered in practice as well as inspection approaches proposed in literature for their underlying test rules. We identified three non-disjunct classes:

Fully automated approaches already contain test rules or derive them from the spreadsheets to be inspected. The whole inspection is done by a tool.

Partly automated approaches rely on the participation of users for specifying test rules. The execution of the inspection is done by a tool.

Manual approaches only provide general test instructions such as a workflow or checklists. The execution of the inspection is done by human experts (studies indicate that teams detect more defects than individuals and that experienced practitioners perform only about 16 - 23% better than untrained students [7]). The use of tools is optional – tools in this area support human experts, e.g. by providing interactive checklists or a visualization of the spreadsheet which is adjusted for inspection purposes such as [15] or [21].

Each of these classes has its pros and cons. In general, it seems obvious that the higher the level of automation, the lower the effort for executing inspections – while, on the other hand, the more humans are involved, the more the effort increases.

However, apart from highly domain-specific ontology-based approaches such as [22], fully automated approaches have limited detection capacities [48] and have not been very successful in detecting faults because they are mostly blind to semantic incorrectness [33]. Furthermore, the effects of design and coding rules for spreadsheets are mostly unknown [25] and anomaly patterns used in automated detection techniques can lead to the reporting of around 90% false positive findings [10]. Thus, we are convinced that relying on fully automated inspection approaches is not sufficient.

3 Human Factors

Spreadsheet inspection approaches are not very helpful if spreadsheet users lack the awareness of why inspections are necessary. As a consequence, inspections are not executed to the right extent, not at the right point in time or not at all. Creating such an awareness is a non-trivial task that can be difficult and tedious – especially considering the issue of overconfidence [34, 36]. We would argue that reducing overconfidence is even more important than increasing correctness as many faulty spreadsheets which caused severe damage did not undergo any inspections at all.

As organizations are forced by law to "prove" that they took actions to verify the correctness of their spreadsheets, they often deploy so-called compliance systems. Compliance systems usually work as follows: they inventory and monitor all spreadsheets in an organization and execute fully automated inspections on them. The findings are then presented to the management or the affected users for sanitization.

By using compliance systems, organizations are able to comply with regulatory requirements. However, we doubt whether the actually relevant quality attributes of the inspected spreadsheets really receive the appropriate care. While we are convinced that some imperfections can be addressed in this way, fully automated inspection approaches have severe limitations when it comes to detecting actual faults and failures (as shown in the previous section). Furthermore, studies in end-user environments show that enforcing unconvinced end-users to change their behavior only makes them find ways to circumvent regulations [30, 37].

When it comes to maintenance of spreadsheets, a highly relevant but often ignored factor is collaboration among end-users. In 2005 and 2006, researchers at Dartmouth College issued a comprehensive study about spreadsheet work habits [6]. A total of 1597 MBA graduates from 7 populations responded to an online questionnaire. For the majority of them, spreadsheets play a very important (33.6%) or critical (49%) role in their work. According to the study, most spreadsheet users (81.1%) work on their spreadsheets alone but when creating new spreadsheets, they often (62.1%) use existing spreadsheets as templates. Also, it is rather uncommon that spreadsheets are used only by their authors (11.5%) – usually, they are shared with one or two (42%) or more other persons (30.9%) and sometimes the spreadsheets become permanent assets (15.7%). When sharing spreadsheets, users typically share the whole file (67.6%). Sharing is usually done daily (19.1%), weekly (37.3%) or annually (28.9%). With these numbers in mind, it is alarming that 88.1% of the participants of this study stated that they do not invest any time in documenting their spreadsheets! An older study by Nardi and Miller shows that collaboration in spreadsheet environments was already common in the 1990s [31].

4 Spreadsheet Guardian

If one would demand collaborating spreadsheet users to adapt to the working habits of disciplined software engineers, the spreadsheet users would have to inspect their spreadsheets after each development activity before starting any filling activities. This would be inconvenient for them as the spreadsheet paradigm does not distinguish between these activities. Also, it is unlikely that typical spreadsheet users who are not aware of the necessity of taking quality assurance steps (due to the overconfidence problem) would accept such a model.

Viewing the whole situation from the perspective of a typical spreadsheet user reveals the following picture: the spreadsheet user gets an undocumented and potentially faulty spreadsheet and is asked to view it, fill in data or even develop it further. Even if the spreadsheet user was convinced of the necessity to inspect (and eventually fix) the spreadsheet first, the user would have a hard time deriving the test rules for the inspection. Most annoyingly, all the effort for deriving the test rules would be lost in the long run if just one of the subsequent users in the chain of collaborators did not care about inspections. After a short time, the spreadsheet would have degenerated back into its original (undocumented and potentially faulty) state.

This is exactly where our idea for a different approach takes up the slack: instead of trying to persuade the bulk of spreadsheet users to do inspections or even enforcing them, we separate the activities of providing a test specification, executing inspections and analyzing findings from inspections. The idea behind this is to use only the “inspection-convinced” spreadsheet users for providing test rules while making the harvest of this effort (findings about violations of the test rules) available to all spreadsheet users – including the ones not persuaded of benefits from doing inspections. Therefore, we ask the following research question:

> *How effective can spreadsheet inspections be if they are specified only by few spreadsheet users but executed by many?*

To provide a theoretical framework for our approach, we developed a method named “Spreadsheet Guardian”. Spreadsheet Guardian models inspections as a relation between producers, processors and consumers:

Test rule producers are spreadsheet users who specify test rules.

Test rule processors are machines that execute spreadsheet inspections according to given test rules and compute findings (typically, there is just one such machine).

Findings consumers are spreadsheet users who receive findings computed by test rule processors.

The interaction between producers, processors and consumers works as follows:

- The test rule producers specify test rules using a tool that supports several specification techniques (for more details see section 5).
- The specified test rules remain in the spreadsheet, even if it is shared with other users who do development activities without having the tool installed.
- The execution of spreadsheet inspections is done continuously and in the background by the test rule processor, while spreadsheet users just work with the spreadsheet as usual, i.e. carrying out filling or development activities.
- After the test rule processor finishes the execution of the test rules, the spreadsheet user is presented with the findings directly in the spreadsheet environment (e.g. in Microsoft Excel). This way, spreadsheet users become findings consumers without any effort on their part.

We claim that Spreadsheet Guardian provides a unique combination of advantages over existing methods. To make them clear, we describe our theory in terms of constructs, propositions, explanations and scope in Table 1 as proposed by Sjøberg et al. [45].

Table 1: Theory of the proposed approach (Spreadsheet Guardian)

Constructs	
C1	Spreadsheet user
C2	Test rule
C3	Test rule producer
C4	Test rule processor
C5	Findings consumer
C6	Spreadsheet Guardian
Propositions	
P1	Specifying test rules using Spreadsheet Guardian is feasible for regular spreadsheet users.
P2	Spreadsheets protected by Spreadsheet Guardian have a higher chance of remaining correct during maintenance.
Explanations	
E1	The specification of test rules is done in the familiar spreadsheet environment. It fits in the spreadsheet paradigm, is easy to learn and does not require computer science knowledge. ($\rightarrow$ P1)
E2	The test rules serve as an implicit specification for (part of) the actual requirements, providing maintainers with more knowledge about the intentions of previous users' development activities. ($\rightarrow$ P2)
E3	The clear separation between production and test code makes test code easier to find. This is especially true when compared to test code some advanced spreadsheet users embed directly in their spreadsheets ([13] argues that such practices are common). ($\rightarrow$ P2)
E4	For findings consumers, the reported findings have a high relevance because they are based on test rules specified by humans working with earlier versions of the spreadsheet. ($\rightarrow$ P2)
Scope	
	Spreadsheets in collaborative environments

5 Implementation

Since 2011, the first author has supervised nine student theses and other projects in which a tool prototype named "Spreadsheet Inspection Framework" (SIF) was developed to implement Spreadsheet Guardian. SIF and its capabilities have already been described in previous work [26]. Therefore, this section covers SIF only briefly.

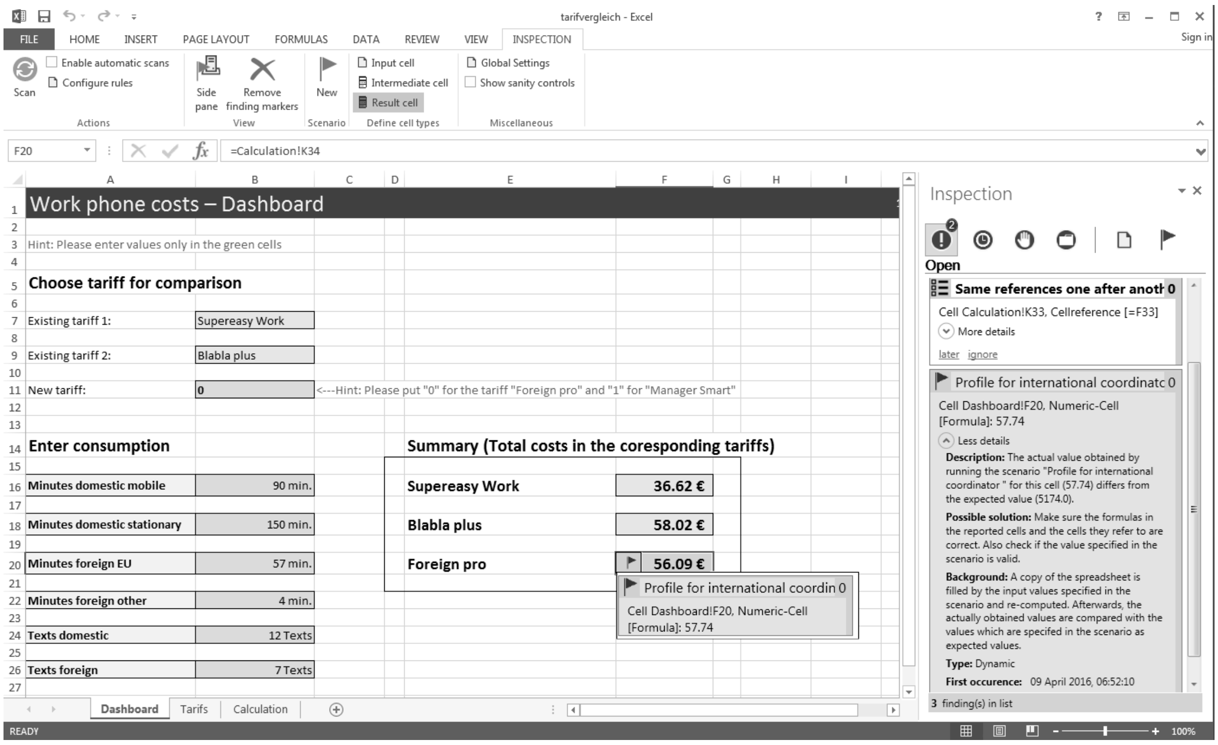
SIF consists of two components – an analytical core and a front-end. The front-end has been implemented as an add-in for Microsoft Excel, thus it provides a tight integration with the environment typical spreadsheet users work in. Both components are open source software and available on the web[1].

Basically, SIF is a framework which provides a common umbrella for experimenting with various spreadsheet inspection techniques. It also sports a total of eight fully automated inspection techniques which allow users to scan spreadsheets both for stylistic issues (e.g., the reading direction of formulas, the presence of constants in formulas) as well as for fault patterns (e.g., references to empty cells, same references one after another as discussed in the example of section 2 B). Apart from technical aspects (e.g., SIF's analytical core is written in Java while the front-end is written in C#), SIF distinguishes itself from similar tools by three key features: (i) its capabilities for reporting and managing findings, (ii) its ability to run inspections continuously in the background and (iii) its support for two partly automated inspection techniques.

As illustrated in Figure 2, SIF's front-end visualizes findings with marker icons directly in the spreadsheet as well as in a synchronized list in a side pane. This way, spreadsheet users are not distracted from their usual workflow but are still informed discretely about new findings (like in a modern IDE for traditional programming). Additionally, users can flag findings as false positives (so they don't reappear on subsequent inspections) or hold off on them.

SIF provides a so-called "live mode" which allows users to run inspections continuously in the background. A first user study from previous work indicates that this mechanism is generally acceptable and

[1] http://www.spreadsheet-inspection.org



F20 =Calculation!K34

Work phone costs – Dashboard

Hint: Please enter values only in the green cells

Choose tariff for comparison

Existing tariff 1:	Supereasy Work
Existing tariff 2:	Blabla plus
New tariff:	0

<---Hint: Please put "0" for the tariff "Foreign pro" and "1" for "Manager Smart"

Enter consumption

Minutes domestic mobile	90 min.
Minutes domestic stationary	150 min.
Minutes foreign EU	57 min.
Minutes foreign other	4 min.
Texts domestic	12 Texts
Texts foreign	7 Texts

Summary (Total costs in the coresponding tariffs)

Supereasy Work	**36.62 €**
Blabla plus	**58.02 €**
Foreign pro	**56.09 €**

Profile for international coordin 0
Cell Dashboard!F20, Numeric-Cell
[Formula]: 57.74

Same references one after anoth 0
Cell Calculation!K33, Cellreference [=F33]
More details
later ignore

Profile for international coordinatc 0
Cell Dashboard!F20, Numeric-Cell
[Formula]: 57.74
Less details

Description: The actual value obtained by running the scenario "Profile for international coordinator " for this cell (57.74) differs from the expected value (5174.0).

Possible solution: Make sure the formulas in the reported cells and the cells they refer to are correct. Also check if the value specified in the scenario is valid.

Background: A copy of the spreadsheet is filled by the input values specified in the scenario and re-computed. Afterwards, the actually obtained values are compared with the values which are specifed in the scenario as expected values.

Type: Dynamic

First occurence: 09 April 2016, 06:52:10

3 finding(s) in list

Figure 2: Spreadsheet Inspection Framework's Microsoft Excel add-in reports findings in a spreadsheet

low disruptive for spreadsheet users' workflows [27].

As of today, SIF supports two partly automated inspection techniques:

Test scenarios port the idea of unit tests to spreadsheets. In the first step, the user marks corresponding cells in the spreadsheet as input cells, intermediate cells or output cells. Then the user specifies a set of values for the input cells and expected values or ranges of values for intermediate and output cells. SIF saves this set of values as a so-called "test scenario" as well as the marked cells in a separate region of the spreadsheet, so this data neither gets lost nor gets in the way when other users change the spreadsheet (even if they don't use SIF at all).

When running an inspection, SIF opens a copy of the spreadsheet in the background, fills in the input values, recalculates the spreadsheet and checks whether the values in intermediate and output cells correspond to the specification given by the user. In case they don't, SIF generates findings and reports them using the previously described mechanism.

Advanced data validation rules allow users to specify more advanced data validation rules than typical spreadsheet environments support out of the box. For instance, it can be specified that if an entry in column A starts with the letters "foo", the corresponding cell in the same row but in a different column must contain a 10-digit number followed by the letters "bar".

6 Evaluation Planning

In previous work, we only evaluated our tool SIF in experiments on a tiny scale. To empirically evaluate the overall concept of Spreadsheet Guardian, we conducted a comprehensive evaluation based on two user studies with different populations (a justification will be provided in section 6.4):

S1 Initial user study with regular spreadsheet users and two experiments (only one experiment per participant)

S2 Follow-up user study with computer science students, replicating a variant of one experiment from S1

6.1 Goals

Our main research objective was to evaluate Spreadsheet Guardian's suitability for the maintenance of spreadsheets. To break it down, we first derived the following experiment questions (EQs) from our propositions (as described in section 4):

> EQ1: *How effective are spreadsheet users at specifying test rules using our partly automated inspection techniques?*

> EQ2: *How effective are user-specified test rules for keeping spreadsheets correct during maintenance?*

Based on that, we formulated the following goals for the evaluation:

- Goal 1: *Analyze* spreadsheet users learning Spreadsheet Guardian and creating test rules
  For the purpose of evaluating if the technique is sufficiently easy to learn and to apply
  With respect to the time required by users to specify the test rules and the effectiveness of the specified test rules
- Goal 2: *Analyze* spreadsheet users maintaining "vanilla" spreadsheets versus spreadsheets protected by Spreadsheet Guardian
  For the purpose of comparing the correctness of the spreadsheets after maintenance
  With respect to the actual correctness of the spreadsheets and the spreadsheet users' confidence

6.2 Hypotheses

We formulated the following hypotheses:

H_1 Spreadsheet users specify at least one test rule correctly using our tool.

H_2 Spreadsheet users specify at least one test rule efficiently using our tool.

H_3 Spreadsheet users perceive maintenance tasks as more complex using our tool.

H_4 After maintenance, spreadsheets not protected by Spreadsheet Guardian contain more faulty output cells than those protected by Spreadsheet Guardian.

6.3 Procedure

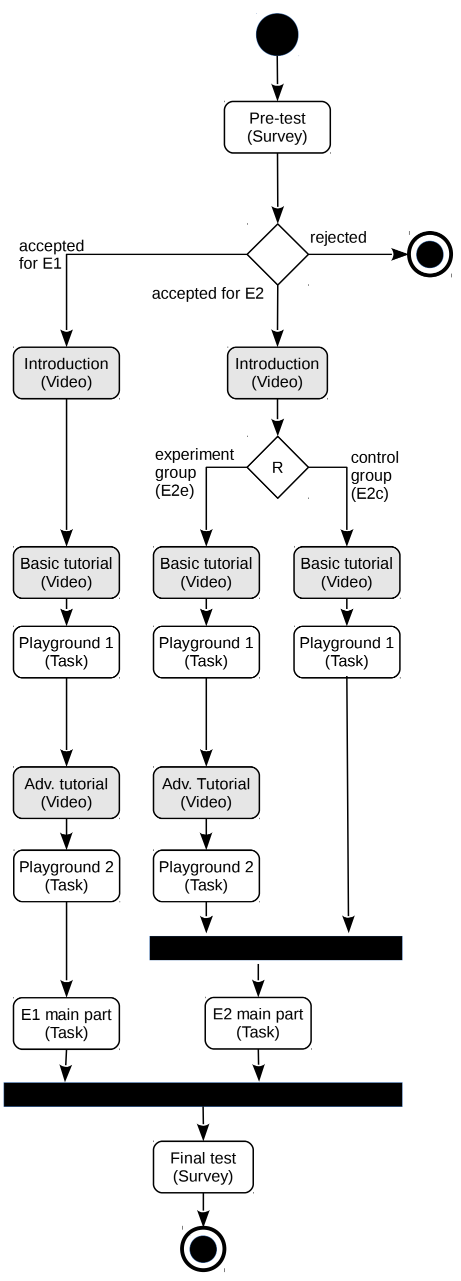
We designed three experiments: E1, E2 and E3. While experiments E1 and E2 were part of the initial user study (S1), experiment E3 was conducted in the follow-up user study (S2). The planned procedure is illustrated in Figure 3.

The procedure for S1 was planned as follows: After a pre-test (which was the same for all candidates), we wanted to compute a "suitability score" for each candidate and decide whether the candidate should be used for E1, E2 or be rejected. While candidates for E1 were planned to be assigned directly, candidates for E2 were planned to be randomly split into an experiment group (E2e) and a control group (E2c). Because our experiment was about maintenance of spreadsheets, we planned to reject novices who typically only carry out filling activities but never develop or change formulas.

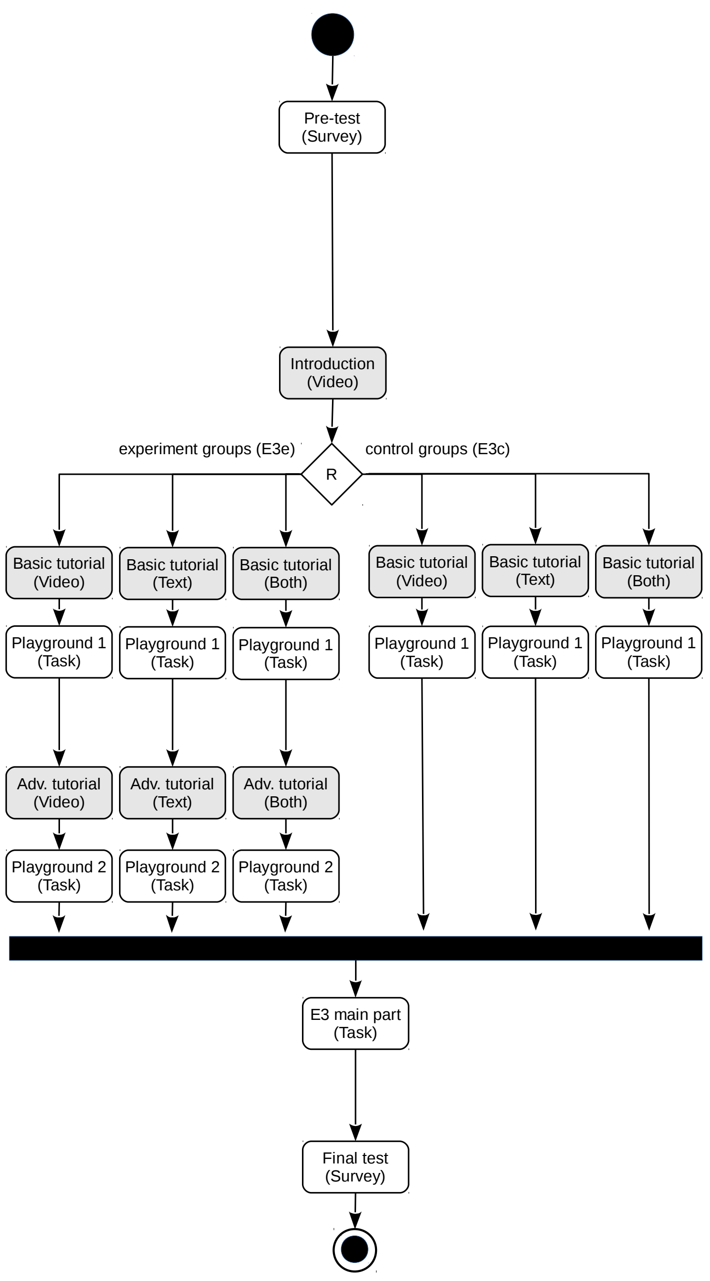
The study S2 with its only experiment E3 was planned as an enriched variant of S1's experiment E2. Therefore, the procedure for S2 was planned to differ only in two aspects:

- The pre-test was planned to be issued only to gather data about the participants but not for rejecting "unsuitable" candidates.
- The participants were supposed to learn our tool with different (but content-equivalent) tutorial types instead of only video tutorials. We implemented this variation to research the effectiveness and efficiency of different tutorial types. Explaining the details behind this is beyond the scope of this paper but we reported the results already in [20].

During the experiments, all participants received the same introduction and the same basic training. However, participants of E1, E2e and E3e received additional training which participants of E2c and E3c did not. After the training, the participants were asked to solve the main part of their experiment. The main part of E2 and E3 was the same while E1 had a completely different main part. At the end, all participants were asked to complete a small survey that differed between the experiments only in small nuances (to keep it simple, this difference is not reflected in the aforementioned Figures of the procedure).



(a) Initial user study (S1)



(b) Follow-up user study (S2), drawing derived from [20]

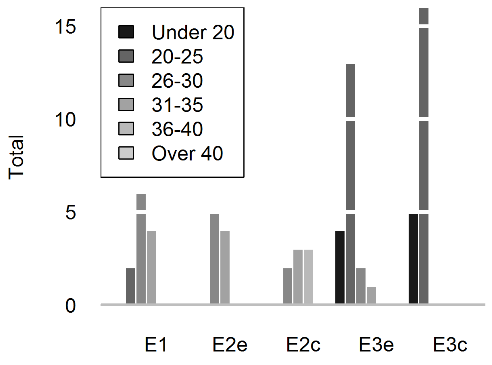
Figure 3: Procedure of the user studies

6.4 Participants

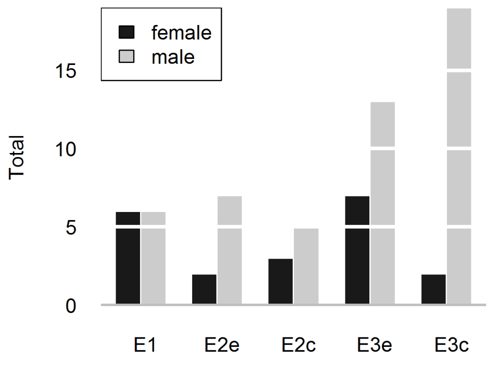
To acquire participants for S1, the first author posted messages in social media channels, contacted friends directly and distributed self-designed paper brochures on various occasions (e.g. sport courses, birthdays, house-warming parties...). We succeeded to attract a total of 48 candidates for S1 but had to reject 7 unsuitable candidates and 4 suitable candidates broke up contact. Not counting our 8 pilots, we had a total of 29 participants who finished the experiments for S1. All participants of S1 were rewarded with 10 euros and a chocolate bar for their participation.

The participants for S2 were acquired from an undergraduate software engineering lecture, where participation in a study was mandatory. Thus, participants of S2 did not receive any monetary compensation for their effort. Apart from our study, the students could also choose from two alternative studies. In our advertisement we stated that taking our experiment required basic spreadsheet skills. We used two pilots for S2 and succeeded to fill all our available slots to a total of 42 candidates. All 42 candidates showed up and took part in the experiments.

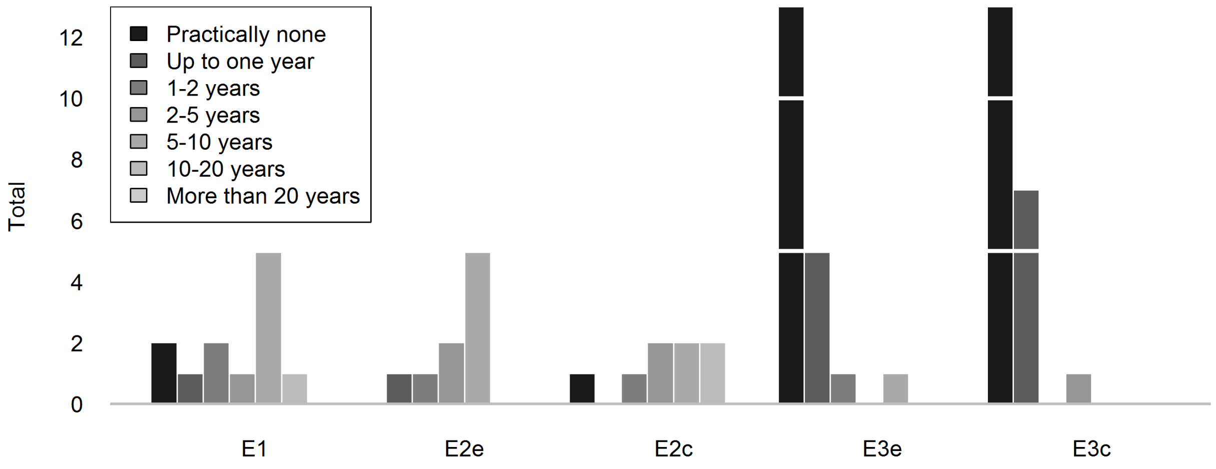
We report demographic data of our samples in Figure 4. When comparing participants from E2 and E3, it can be seen that the participants in E3 were younger and had less professional working experience. Since in E2 and E3 we randomly assigned participants to the control and experiment groups, the percental distribution of genders turned out to be pretty unbalanced – especially between E3e and E3c.



(a) Q: Which age group do you belong in?



(b) Q: What is your gender?



(c) Q: How many years of professional work experience do you have?

Figure 4: Demographic data about the participants

As mentioned, we used different populations in S1 and S2. At first sight, one might argue that using *solely* typical spreadsheet end-users would have been a better choice for targeting our goals – and this was also our initial plan. However, after conducting S1 we learned that attracting a sufficient number of participants – especially advanced spreadsheet users with some years of professional experience – was very hard. Furthermore, finding suitable time slots for the participants who have a full-time job and/or who have family turned out to be very challenging. This forced us to find a balance between tightly controlling environment variables (like time, lighting conditions or external distraction factors) and offering flexibility to reach more participants (or to find participants at all).

We addressed this issue by allowing participants of S1 to take the experiments off-site (in their homes) and in the evening or during the weekends (when they were tired after work or not in the mood for concentrating on hard work). However, to address the shortcomings of this compromise, we decided to slightly modify one of the experiments of S1 and replicate it in a more controlled environment – at the cost of running it with potentially less representative computer science students.

The goal of our pretest in S1 was to triage our population of candidates for S1 into spreadsheet beginners, intermediates and advanced users. This was necessary to honor the premises of our research questions and to make sure that the level of difficulty was appropriate for the participants so they neither feel underchallenged nor overstrained. Consequently, another goal of the pre-test in S1 was to identify and reject candidates with a clear background in computer science or software engineering. Since we accepted every candidate for S2, the pre-test in S2 only allowed us to compare this population with the one from S1 (and especially the candidates who took E2).

6.5 Experimental Material

Except for videos (which are available only upon request), all experimental material is available from our open data repository [24]. In the following, we describe the material briefly.

6.5.1 Instructions

All instructions during the experiments were given on paper sheets (11 pages for E1, 10 pages for E2e and E3e, 7 pages for E2c and E3c) in German[2]. During the experiments, the participants were asked to read the instructions and follow them. This involved watching video tutorials or reading text tutorials, applying the steps learned by doing tasks in spreadsheets and answering questions on paper. We reported more details about the experimental material in [20].

6.5.2 Pre-test

The pre-test was implemented as an online survey. The survey had 21 questions and was loosely based on the survey used in the study of Baker et al. [6]. However, our variant was much shorter to make it completable in 5 to 10 minutes so we would not risk losing candidates for S1. The questions were targeted at rating the experience of the participants and checking which spreadsheet activities they typically perform (e.g. just filling activities or also development?).

For the actual triage after the pre-test, we developed a simple scoring sheet which computed a weighted sum based on the answers.

6.5.3 Training

Understanding and learning SIF and Spreadsheet Guardian requires some basic training. To provide adequate training to all participants and keep them motivated during the training phase, we employed a training concept with short "learn-and-apply" cycles: after watching recorded videos or reading text tutorials, the participants had to do small practical tasks in spreadsheets.

The basic training started with an introduction video (duration: 9m 3s) to spreadsheets, anomalies in spreadsheets and design rules for spreadsheets. To avoid unnatural behavior of the participants regarding our experiment questions, the video tried to give the participants the false impression that the experiment's goal was to measure effects of different spreadsheet design rules.

In the second part of the basic training we explained SIF's user interface in a video tutorial (duration: 5m 42s) or an adequate text tutorial. The tutorials showed how rules can be configured, how inspections can be started, how findings are reported and how they can be managed. After the tutorials, participants were asked to replicate the steps on a small "playground spreadsheet" (spielwiese.xlsx) that contained seeded faults.

The advanced training started with a video tutorial (duration: 10m 1s) or text tutorial that explained the test scenario inspection technique and how to apply it in SIF. Again, the participants were asked to apply the technique to the playground spreadsheet afterwards. This involved marking cells, creating a test scenario, interpreting its results and finding the cause for a (seeded) failure.

6.6 Tasks

To produce a non-too artificial tarifvergleich.xlsx spreadsheet we asked an experienced spreadsheet user to produce the spreadsheet according to a list of requirements given. We then modified the spreadsheet's calculation worksheet to make it fit our experimental design and seeded in two faults and one "evil" imperfection as shown in Figure 5a:

- the sum formulas in cells K33 and K34 added a value twice (like in the example in section 2)
- the reference in cell J34 was pointing to the wrong cell, i.e. to J29 instead of J30 (Excel's built-in mechanism fails to detect this).
- The indexes for all VLOOKUP functions were stored in separate cells but with white color on white background, e.g. cell H28 contains a VLOOKUP function which reads the index from cell H25 that looks empty at first sight.

As generally known, seeding defects is not the best solution. However, we did not find any suitable real spreadsheets with the defects we wanted that would fit our experiment design.

[2]We translated the spreadsheets in the figures to ease understanding.

6.6.1 Main Part of E1

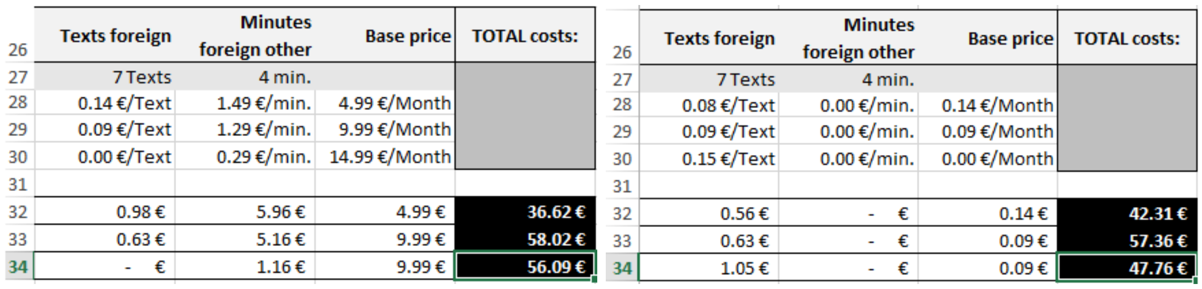
The main part of E1 consisted of four tasks. In the first task, the participants had to recompute the result for a given set of input values with the calculator of the operating system (not the spreadsheet). The second task was to create a new test scenario in SIF with the same values to show them that Spreadsheet Guardian would report the finding. In the third task, the participants were asked to look for the seeded faults in the spreadsheet and to try removing them so that the finding reported by the test scenario would be eliminated. They were given the hint to activate the fully automated inspection techniques (which gave hints for the first fault) but had to find the second fault manually. In the last task, they were asked to "invent" consumption values for a described cell phone user and to compute expected values using a calculator before using SIF to create and run a test scenario for it.

6.6.2 Main Part of E2/E3

Unlike in E1, in E2/E3 we wanted to simulate a situation on the "receiver side" of a user maintaining a spreadsheet. Thus, the focus here was less on corrective maintenance (tracing failures and finding faults) but on adaptive maintenance, i.e. developing the spreadsheet further by adding more features.

To tackle the experiment question, we divided the participants into two groups – an experiment group (E2e/E3e) and a control group (E2c/E3c). However, we think that it would have been too hard to isolate involved effects if we had one group with all the training and SIF's complete stack and one group with more or less nothing. Therefore, we decided to give both groups the same basic training and SIF's fully automated inspection techniques but to provide only the experiment group with Spreadsheet Guardian's test scenario inspection technique.

Both the experiment groups (E2e/E3e) and the control groups (E2c/E3c) received the same task and basically the same spreadsheet ("tarifvergleich.xlsx"). However, only the experiment groups' spreadsheet already had four test scenarios planted in. These test scenarios were the first four correct test scenarios that the participants of E1 produced (we chose this approach to avoid artificial scenarios).



26	Texts foreign	Minutes foreign other	Base price	TOTAL costs:
27	7 Texts	4 min.		
28	0.14 €/Text	1.49 €/min.	4.99 €/Month	
29	0.09 €/Text	1.29 €/min.	9.99 €/Month	
30	0.00 €/Text	0.29 €/min.	14.99 €/Month	
31				
32	0.98 €	5.96 €	4.99 €	36.62 €
33	0.63 €	5.16 €	9.99 €	58.02 €
34	- €	1.16 €	9.99 €	56.09 €

(a) Pristine

26	Texts foreign	Minutes foreign other	Base price	TOTAL costs:
27	7 Texts	4 min.		
28	0.08 €/Text	0.00 €/min.	0.14 €/Month	
29	0.09 €/Text	0.00 €/min.	0.09 €/Month	
30	0.15 €/Text	0.00 €/min.	0.00 €/Month	
31				
32	0.56 €	- €	0.14 €	42.31 €
33	0.63 €	- €	0.09 €	57.36 €
34	1.05 €	- €	0.09 €	47.76 €

(b) After erroneous maintenance

Figure 5: Excerpt of "tarifvergleich.xlsx"

The main part of E2/E3 had just three tasks. In the first task, the participants were asked to configure test rules, run them and fix the reported findings. While participants of E2e/E3e were given concrete findings pointing to the second seeded fault, the participants of E2c/E3c were not given any hints – but we also did not expect them to detect it. In the second task, the participants were asked to add a new tarrif and to include it in the comparison of the "Dashboard" worksheet.

The crux was the third task: here, we asked the participants to extend the spreadsheet by a new consumption category ("Texts network-internal"). The deliberately nasty trap here was that by inserting a new column in the Tarrifs worksheet, the hidden indexes used for the VLOOKUP functions in the Calculation worksheet would not be automatically updated - resulting in obvious failures as shown in Figure 5b.

6.6.3 Final Test (FT)

The purpose of the final test was to investigate how our participants liked the experiment and our SIF tool, how confident they felt about their modifications being correct (only E2/E3) and to gather more data about their habits and background. This was done by a paper survey which consisted of 33 questions. The questions were a mix of questions adapted from the survey by Baker et. al. [6] and our own.

6.7 Analysis Procedure

To judge the correctness of the "tarifvergleich.xlsx" spreadsheets in E2/E3, we designed eight simple test scenarios – four for regression testing and four for testing the functionality the participants were supposed to add. To make it fair for the control groups, we treated output values as correct even if they did not fix the second seeded fault (which was hard to find without the hint given by the test scenarios).

We executed the tests manually twice (without the help of SIF), i.e. we opened each spreadsheet, filled in the test scenario's input values and captured the result values. Our results were not normally distributed, therefore we decided to use a Wilcoxon signed-rank test for statistical analyses. Additionally, we computed Cohen's d for the effect size. To measure normal distribution as a t-test precondition we used a Shapiro-Wilk test.

7 Evaluation Execution and Analysis

All experiments of S1 were done in a time frame of six weeks. The execution of S2 was done in two weeks. However, designing, preparing and piloting the experiments as well as evaluating the results took us about seven months.

7.1 Instrumentation

We did not want to breathe down the participants' necks during the experiments. Therefore, we watched the experiments over VNC connections from the experimenters' machines. In E1 and E2, the environment was isolated and the participants had no internet access. In E3, the participants did have internet access.

For E1 and E2 we did not impose strict time limits. We only asked the participants to reserve about 90 minutes for the experiment. Due to university regulations, for E3 we had to limit the available time to 120 minutes. When time was up, we told the participants that time was over but they were free to continue the experiments if they wanted.

In general, we refrained from giving the participants hints and interfering with the experiments except for the following reasons:

- A participant did not understand the instructions.
- A participant encountered one of SIF's known bugs.
- A participant was not sure what the parameters of the VLOOKUP function stand for. In such cases, we explained the function to participants of E1 and E2. For E3, we motivated the participants to do research on the internet themselves.
- A participant of E1, E2e or E3e had no idea how to find the defects (we gave the advice to enter the input values of a test scenario manually and continue searching).

7.2 Observations

We analyzed the collected data and looked for interesting findings. In the following, we will present a few selected results that are relevant especially regarding our experiment questions and hypotheses.

7.2.1 Pre-test

Figure 6 shows the results of the triage. The horizontal lines mark the thresholds we used in S1 to distinguish between suitable and unsuitable participants for E1 and E2. There is a significant difference between the scores of E2 and E3 (p=0.0005, d=0.6910). It can be seen that of the participants we accepted for E3, we would have rejected many for E2 (and some even for E1).

7.2.2 Participants' Impressions

In all experiments, only five participants stated that they did not like the study too much. The remaining participants stated that it was very good (26) or good (39) (Figure 7a). We think that participants who like a study are more likely to give their best during an experiment whereas participants who do not like a study tend to rush through it to get it over with. A similar picture can be drawn for our tool SIF in general where eight participants stated not to like SIF too much, while the remaining participants stated that it was very good (26) or good (37) (Figure 7b).

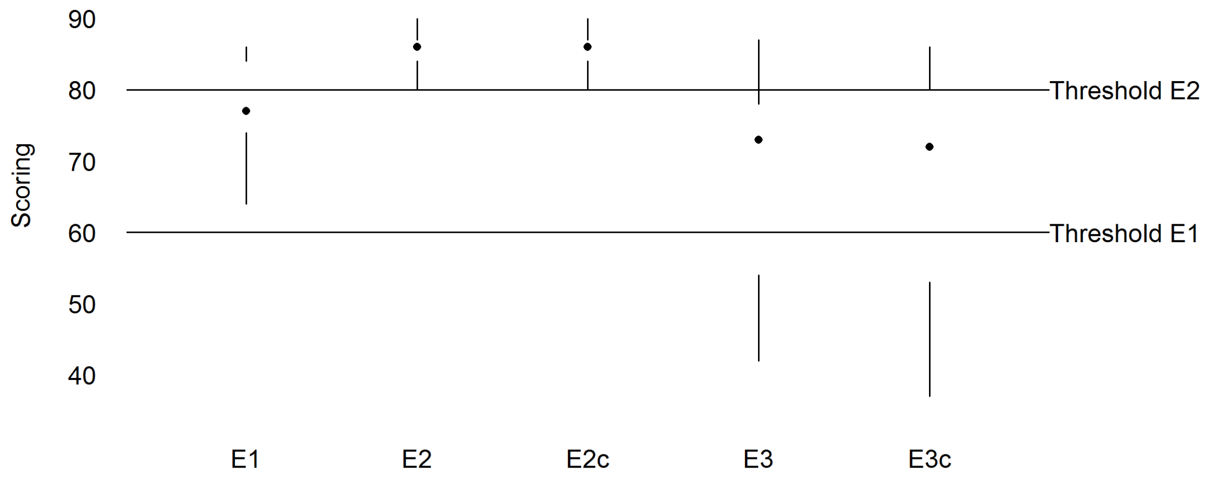
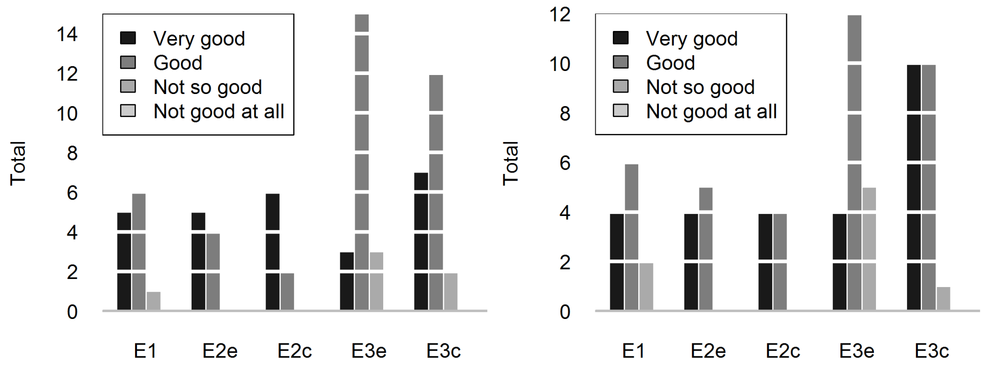


Figure 6: Participants' scores



(a) Q: How did you like our study? (b) Q: How did you like our tool SIF?

Figure 7: Participants' impressions

7.2.3 Hypothesis H_1 - Correctness of Specified Test Rules

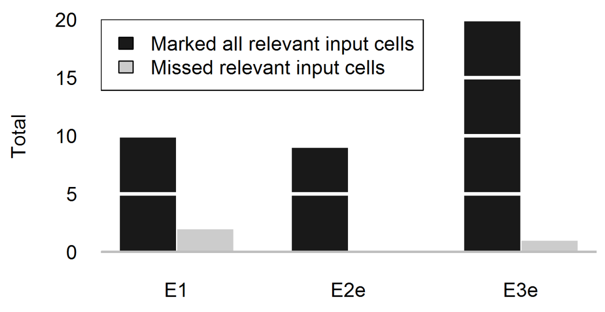
For creating scenarios, it is necessary to first mark input, intermediate and output cells. We therefore analyzed whether the participants succeeded to do so. While *all* participants successfully marked *all* intermediate and output cells correctly, a small percentage of the participants missed some input cells or marked irrelevant cells as input cells (Figure 8). We did a t-test to see if a significant amount of participants selected all relevant input cells and it confirmed that they did (p_{E1}=6.663e-06, p_{E2e}=constant, p_{E3e}=5.4e-15). The same is true for not marking irrelevant cells (p_{E1}=6.663e-06, p_{E2e}=constant, p_{E3e}=3.344e-10).

Only creating test scenarios is not sufficient for identifying faults. A test scenario must be also correct to unveil a fault. This requires populating the marked cells with reasonable values when creating scenarios. The analysis of all created scenarios showed that *all* participants provided reasonable values for all types of cells. In the cases where participants marked too many input cells, we observed that they simply filled irrelevant cells with zeroes. Therefore, we can reject H_{1_0}: it is possible to specify test rules correctly.

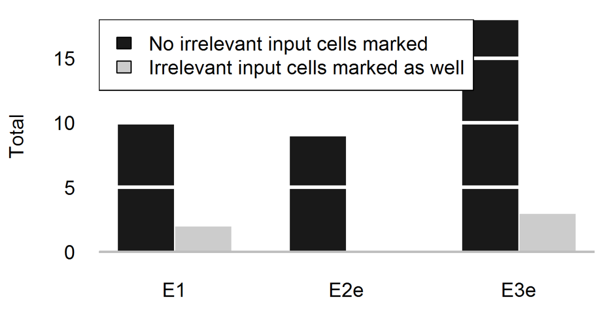
7.2.4 Hypothesis H_2 - Efficiency of Specified Test Rules

To analyze the efficiency for specifying test rules, we measured the duration of the playground tasks. We set 15 minutes as a reasonable time for playground 1 and 20 minutes for playground 2. Figure 9 shows the measured duration of the playground tasks.

As it can be seen, the participants solved the tasks quickly. A t-test confirmed that all groups on playground 2 were significantly faster than 15 minutes (p_{E1}=0, p_{E2e}=0, p_{E2c}=0, p_{E3e}=0.0002, p_{E3c}=0.0023). Also, for playground 2 a t-test confimed that the participants were significantly faster than 20 minutes

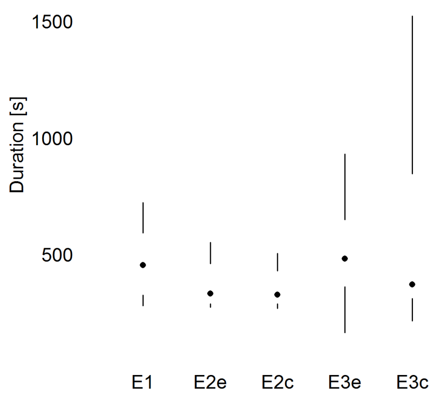


(a) Completeness

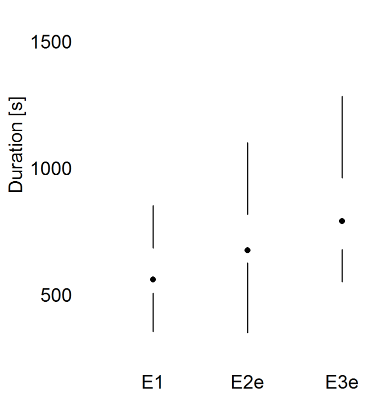


(b) Correctness

Figure 8: Marking of the input cells



(a) Playground 1



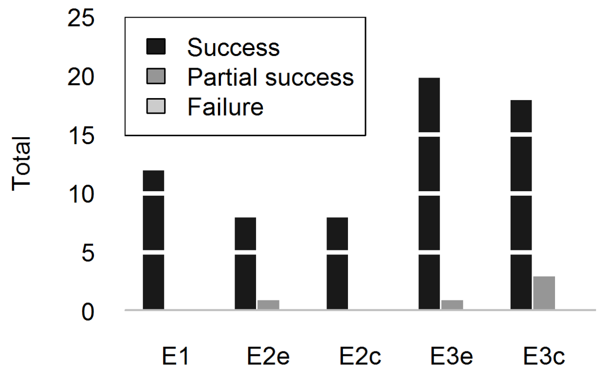
(b) Playground 2

Figure 9: Duration of the playground tasks in seconds

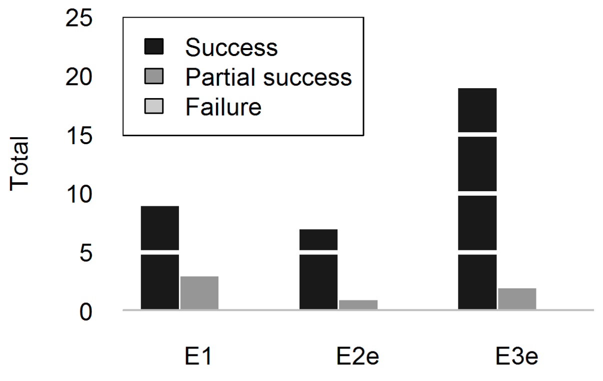
(p_{E1}=2.3148e-05, p_{E2e}=8.1019e-05, p_{E3e}=0.0002). Thus, we can reject H_{2_0}: it is possible to specify test rules efficiently.

7.2.5 Hypothesis H_3 - Learning Spreadsheet Guardian

One could argue that if an activity is more complex, the success rate of applying it is lower. Therefore, we examined the success rate of both playgrounds. As Figure 10 indicates, most participants were able to solve the tasks of both playgrounds.



(a) Playground 1



(b) Playground 2

Figure 10: Task-solving performance of the participants

In addition to the task solving success, we also analyzed the perceived difficulty to see if applying

Spreadsheet Guardian added to this. As shown in Figure 11, the majority of the participants of E1 perceived the difficulty level as adequate. For both, E2 and E3, some participants of the particular experiment group perceived the experiment to be slightly more difficult than in the particular control group. However, this difference is not significant ($p_{E2e-E2c}$=0.1438, $p_{E3e-E3c}$=0.0731).

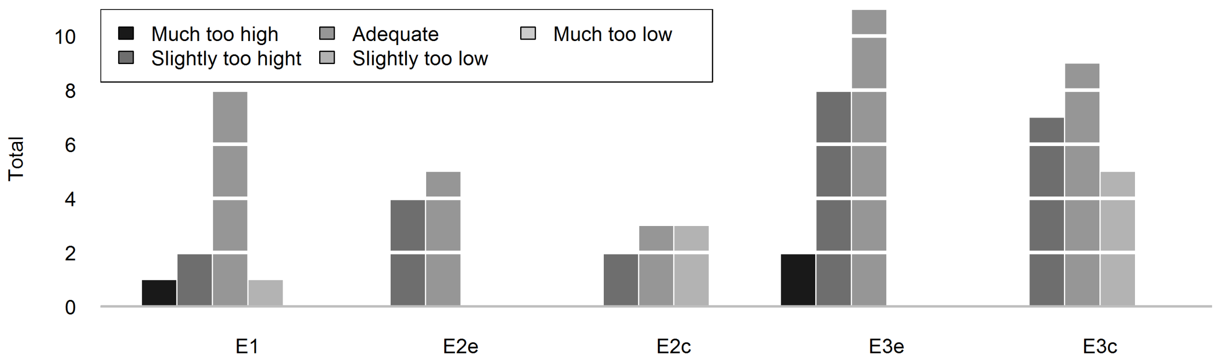


Figure 11: Q: For me, the difficulty level of the experiment was ...

After these analyzes we cannot reject H_{3_0}: the participants did not perceive the application of Spreadsheet Guardian as significantly more complex.

7.2.6 Hypothesis H_4 - Correctness

Figure 12 shows the results of the final task. As you can see, most participants could not solve the task in a way that produced correct results. In E3 the control group was significantly better than the experiment group (p=0.0401, d=0.6695). In E2 there was no significant difference (p=0.6625). Due to these mixed results, we cannot reject H_{4_0}: at least in our sample, spreadsheets protected by Spreadsheet Guardian did not contain less faulty cells than "unprotected" spreadsheets.

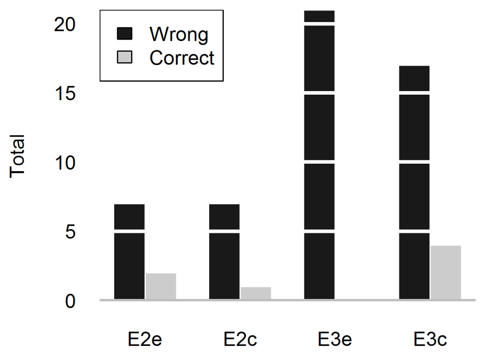


Figure 12: Correctness after maintenance activities in E2e/E3e and E2c/E3c

7.2.7 Further Exploratory Analysis

During our analysis, we noticed that some participants were more confident then others regarding the correctness of their spreadsheet after their modifications. Although this was covered neither in our initial propositions nor our hypotheses, we decided to analyze this as well.

In the final test, the participants had to state on a 7p. Likert scale how sure they were that the spreadsheet in the final task yielded correct results after their maintenance activities. Figure 13 shows their responses. As it can be seen, the participants of E2e and E2c were comparable confident whereas in E3 the participants of E3c were more confident than those of E3c.

We also compared the actual correctness of the maintained spreadsheets in relation to the participants' confidence about them being correct. As illustrated in Figure 14, many participants of the control groups were very sure about the spreadsheets' correctness while in fact most were wrong. On the other hand, the self-confidence of the participants in the experiment groups was more balanced.

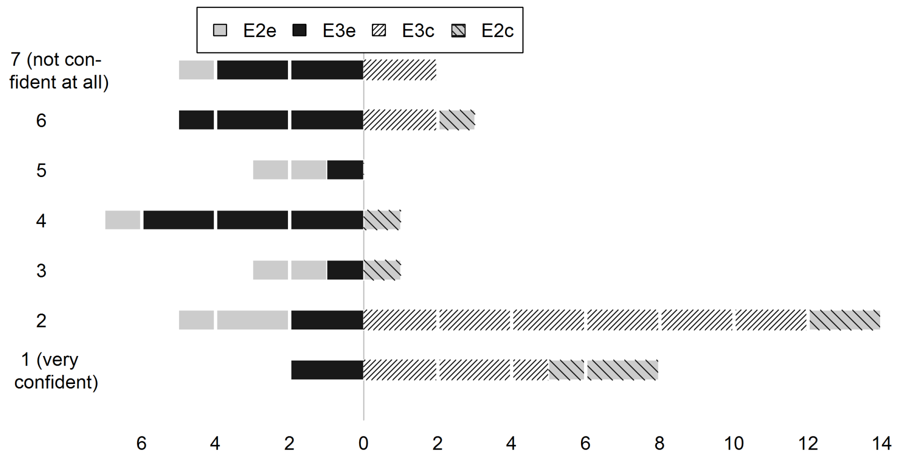
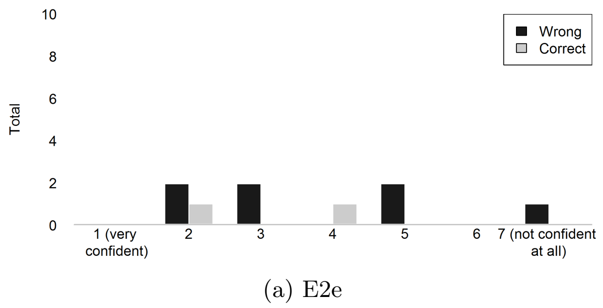
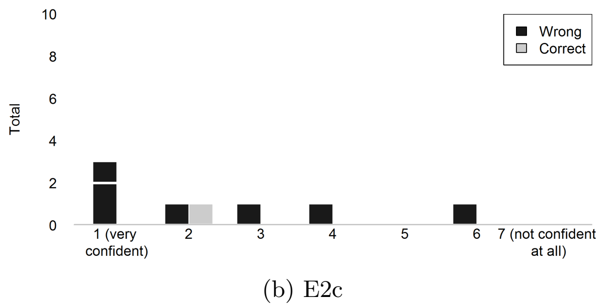


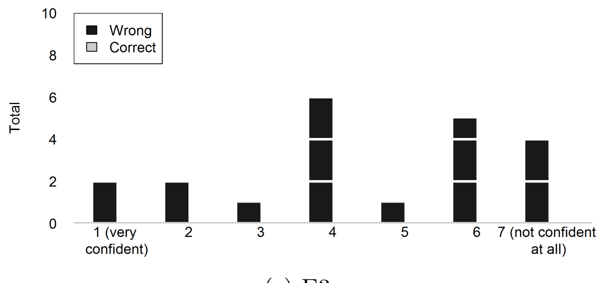
Figure 13: Q: How confident are you that your spreadsheet is correct?



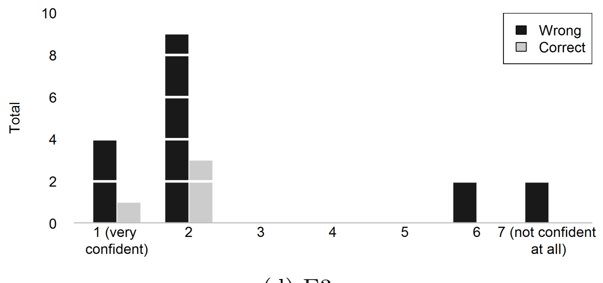
(a) E2e



(b) E2c



(c) E3e



(d) E3c

Figure 14: Distribution of confidence across wrong and correct solutions

Table 2: Ratio between confidence and actual correctness for E2 and E3

Experiment	Assessment correct?	# (Very) confident	# Not confident (at all)	Total	Ratio
E2e	Yes	1	1	2	1 (2:2)
	No	2	0	2	
E2c	Yes	1	1	2	0.5 (2:4)
	No	4	0	4	
E3e	Yes	0	9	9	2.25 (9:4)
	No	4	0	4	
E3c	Yes	4	4	8	0.62 (8:13)
	No	13	0	13	

We further investigated the correctness of the participants' assessments. We counted an assessment as being correct if a participant's spreadsheet was correct and the participant was confident about it being correct. Also, when a participant stated a very low confidence and the spreadsheet was indeed incorrect, we evaluated this as a correct assignment. Likewise, wrong spreadsheets but participants being confident that it was correct as well as correct spreadsheets but participants being unconfident were counted as incorrect assessments. For this measure, we only used very confident ratings (1 and 2 on our Likert scale) and very unconfident ratings (6 and 7 on our Likert scale) and disregarded the "unsure" ratings (3, 4 and 5 on our Likert scale).

Table 2 summarizes the results about the correctness of the participants' assessments. We also computed a ratio between correct and incorrect assessments for each group. The results show that participants of the control groups had much more realistic assessments about the correctness of their spreadsheets than participants of the experiment groups.

7.3 Interpretation

7.3.1 Specifying Test Rules

As the results of our E1 group show, specifying test rules using Spreadsheet Guardian is easy and feasible even for regular spreadsheet users. However, when comparing E2 and E3 it becomes evident that our approach works even better for experienced Excel users than for novices. For those who were not able to do so, we analyzed the screencasts and came to the conclusion that it was not because they could not use SIF but because they lacked the knowledge how to solve the specific spreadsheet fault. For example, most participants were able to create a scenario correctly, even if they did not mark all cells correctly. However, some did not understand the findings SIF raised for this scenario. Therefore, we conclude that for specifying test rules the major challenge is to actually convince users rather than make them actually apply our approach (which is not the problem).

While following a surprise-explain-reward strategy [47] could help to improve this aspect further, our assumption is that test rule producers need to be people who work with spreadsheets on a daily basis and who are not already overstrained by using slightly advanced functions such as IF or VLOOKUP.

7.3.2 Learning Spreadsheet Guradian

It is partly surprising that experienced Excel users seem to outperform undergraduate computer science students by far in both applying our approach and successfully maintaining spreadsheets in general (the overall success rate in E3 was 4/42 while in E2 it was 3/14). This is interesting, as the computer science students are likely to be already familiar with the concept of unit testing (which is taught in the first semester of our courses). Therefore, we conclude that previous knowledge of the concept of unit testing is not required when it comes to learning and applying Spreadsheet Guardian.

7.3.3 Effects on Correctness

When it comes to judging the usefulness of our approach for actually preserving correctness during maintenance, the results are not very clear. Be it with or without Spreadsheet Guardian – the vast majority of our participants broke the spreadsheets' correctness during maintenance. While in E2 the users who employed Spreadsheet Guardian performed slightly better, in E3 it was the other way round. We think that it would require a sample with a higher maintenance success rate (and, thus, possibly easier tasks) to make profound judgments on this aspect. Therefore, from the actual results, we cannot attest Spreadsheet Guardian a clearly positive impact on preserving correctness.

7.3.4 Effects on Overconfidence

While there are no doubts that the correctness of spreadsheets is important, we consider having a realistic perception about the correctness to be even more important. When comparing the results from E2/E3 between the experiment and control groups, we are really stunned that the "invisible gorilla" effect [9] was much stronger than we expected – we previously thought that most participants would recognize at first sight that something must be terribly wrong with the numbers in the base price column (see Figure 5).

From the results, we conclude that just deploying a tool with only fully automated inspection approaches might lead to a false perception about correctness (if no findings are reported by such a tool, users might be even more confident that the spreadsheet is correct). In contrast, applying Spreadsheet Guardian seems to successfully fight overconfidence. This encourages us to extend our original theory (described in section 4) as follows:

Propositions	
P3	By applying Spreadsheet Guardian, spreadsheet users can reasonably judge the actual correctness of their spreadsheets.
Explanations	
E5	Newly raised findings about violations of test rules specified by spreadsheet users before maintenance are serious indicators for newly introduced faults. (→P3)

8 Threats to Validity

When designing and executing the experiments, we followed general principles described in [39]. Still, the validity of this work might be affected by a number of threats. We split all identified threats into three groups: construct validity (CV), internal validity (IV) and external validity (EV). In the following, we provide a brief discussion:

- (CV) Our approach is targeted at typical spreadsheet users. The participants of E3 do not belong to this group. However, it remains questionable whether undergraduate computer science students are actually better at doing work with spreadsheets than typical end-users. We therefore rate this threat only as moderate.
- (IV) Because most of our participants for E1 and E2 were fully employed, we did the experiments in their homes at varying times of the day. Therefore, we were unable to control factors such as their tiredness, workplace suitability (eight participants wanted to work from their couch or on small kitchen tables) or lighting conditions. We don't regard this threat as critical because environmental factors are not perfect in workplaces either.
- (IV) Although the population for E3 had a similar background, using only randomization for shaping the control and experiment groups for E3 may have led to imbalance. While the gender distribution is definitely problematic, other factors like professional work experience and age were comparable. Yet, the differences in confidence ratings are too high to assume their origin in the imbalance between those groups. Therefore, we see this only as a moderate threat.
- (EV) We used spreadsheets with seeded defects instead of real spreadsheets. Also, the experiments were tailored to the type of faults our tool is able to detect. On the other hand, this allowed us to make the experiment mostly domain-independent, i.e. previous knowledge in a specific domain like finance was not necessary. For the aims of our study, we see this as an acceptable compromise.
- (EV) Although we have spent extensive time on recruiting participants for E1 and E2, the sample size remains rather small. This threat, combined with the low number of correctly maintained spreadsheets, hinders us from making statements about Spreadsheet Guardian's effect on maintaining correctness. For E2, we partly mitigated this threat by adding the participants from E3 (but, as discussed, this has brought up new issues).
- (EV) While in E1 and E2 we could deny participants, in E3 we had to take every student who registered for our experiment. The higher frustration rate can be clearly seen from our results. However, this also helped us to get a better understanding of sensible requirements for test rule producers. We see this threat to be harmless as long as we cast a skeptical eye on it.

9 Related Work

Spreadsheet Guardian is an approach which is both preventive and detective and employs partly automated inspection techniques – this is a unique combination. In the following, we want to compare it with other approaches which also employ partly automated detection techniques.

First of all, it is important to know that Spreadsheet Guardian focuses on detecting failures, not faults. Presuming that failures are known, there are several approaches that are able to detect cells which are likely to cause them [3, 17, 2]. These approaches could be used in conjunction with our approach. However, we think that detecting failures is the actual key challenge.

Among the partly automated inspection techniques, WYSIWYT ("What you see is what you test") is probably the most cited one. It was initially developed as a generic testing technique for visual programs [43] and later adapted for spreadsheets [8]. WYSIWYT works as follows: it automatically detects input cells and then populates them with values provided by users or a random generator. Then, users have to state whether they think that values in other cells (i.e. intermediate or result cells) are right or wrong, so WYSIWYT can flag failures and help locating corresponding faults. Spreadsheet Guardian works the other way round: it *first* asks the user to compute values outside of the spreadsheet and then compares these expectations with values computed by the spreadsheet. We expect that this tackles the overconfidence problem better since humans prefer to make judgments from memory instead of by recomputing facts [41].

In Ayalew's "Interval Testing" approach [4, 5], users specify ranges of plausible values for input, intermediate and output cells. In a subsequent symbolic execution of the spreadsheet, theoretical violations of these ranges are computed and reported as findings. In contrast to this approach, Spreadsheet Guardian recomputes the spreadsheet in a real and not just symbolic execution, thus, it is not limited to certain types of supported spreadsheet functions. Another difference is that Spreadsheet Guardian detects actual failures and not only theoretical disagreements between the spreadsheet and its test specification. On the other hand, Spreadsheet Guardian covers only the few specified test cases while Interval Testing is able to test a much broader range.

The "EXQUISITE" approach [18] is the only approach that implements unit tests for spreadsheets in a comparable fashion to Spreadsheet Guardian. However, it relies on automatic detection of cell types and does not allow users to manually specify them which can have certain disadvantages [23]. Another difference is that EXQUISITE does not execute the inspections continuously in the background. Also, just like WYSIWYT, it does not hide values already present in the spreadsheet at the time of specification which probably also makes it more prone to the overconfidence issue.

10 Conclusion and Future Work

In this work, we have proposed an approach and a theory for supporting the maintenance of spreadsheets in collaborative settings. By continuously applying partly automated inspection techniques that separate the specification of test rules from their execution, Spreadsheet Guardian is able to protect the semantic correctness of spreadsheets to some degree – at least indirectly.

The results from our study do not indicate that spreadsheets automatically have significantly higher chances of staying correct during maintenance when Spreadsheet Guardian is applied. Yet, our empirical evaluation gives definite indications that Spreadsheet Guardian considerably lowers overconfidence. Hence, it can be assumed that users applying Spreadsheet Guardian would be more careful when taking business-critical decisions based on such spreadsheets. After all, we should not forget that many of the widely reported "spreadsheet horror stories" did not result from ineffective inspections but simply from overconfidence. Therefore, we recommend to consider Spreadsheet Guardian for any business-critical spreadsheets that are maintained in a collaborative setting.

However, as discussed, one could certainly argue that the results obtained from our controlled experiments are not yet completely convincing. This is why we think that it would be advisable to investigate Spreadsheet Guardian in the field as well. Furthermore, conducting deductive research on our inductively introduced third proposition for the theory of Spreadsheet Guardian is necessary: This should provide more insight into the effects regarding the reduction of overconfidence.

References

[1] Eusprig horror stories. URL `http://www.eusprig.org/horror-stories.htm`. [2010-09-30].

[2] R. Abreu, A. Riboira, and F. Wotawa. Constraint-based debugging of spreadsheets. In *CIbSE*, pages 1–14, 2012.

[3] R. Abreu, J. Cunha, J. P. Fernandes, P. Martins, A. Perez, and J. Saraiva. Smelling faults in spreadsheets. In *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*, pages 111–120. IEEE, 2014. doi: 10.1109/icsme.2014.33.

[4] Y. Ayalew. *Spreadsheet testing using interval analysis*. PhD thesis, Klagenfurt University, 2001.

[5] Y. Ayalew. A user-centered approach for testing spreadsheets. *International Journal of Computing and ICT Research*, 1(1):77–85, 2007.

[6] K. R. Baker, L. Foster-Johnson, B. Lawson, and S. G. Powell. A survey of MBA spreadsheet users. *Retrieved April*, 30:2010, 2006.

[7] B. Bishop and K. McDaid. An empirical study of end-user behaviour in spreadsheet error detection & correction. *Proc. European Spreadsheet Risks Int. Grp. (EuSpRIG)*, 2007.

[8] M. Burnett, A. Sheretov, B. Ren, and G. Rothermel. Testing homogeneous spreadsheet grids with the ”what you see is what you test” methodology. *Software Engineering, IEEE Transactions on*, 28 (6):576–594, 2002. doi: 10.1109/tse.2002.1010060.

[9] C. Chabris and D. Simons. *The invisible gorilla: And other ways our intuitions deceive us*. Broadway Books, 2011.

[10] J. Cunha, J. P. Fernandes, H. Ribeiro, and J. Saraiva. Towards a catalog of spreadsheet smells. In *Computational Science and Its Applications–ICCSA 2012*, pages 202–216. Springer, 2012. doi: 10.1007/978-3-642-31128-4_15.

[11] K. Frühauf, J. Ludewig, and H. Sandmayr. *Software-Prüfung: eine Anleitung zum Test und zur Inspektion*. vdf Hochschulverlag AG, 2007. doi: 10.1007/978-3-322-94041-4.

[12] T. A. Grossman, V. Mehrotra, and Ö. Özlük. Lessons from mission-critical spreadsheets. *Communications of the Association for Information Systems*, 20(1):60, 2007.

[13] F. Hermans. Improving spreadsheet test practices. In *CASCON*, pages 56–69, 2013.

[14] F. Hermans, M. Pinzger, and A. Van Deursen. Supporting professional spreadsheet users by generating leveled dataflow diagrams. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 451–460. ACM, 2011. doi: 10.1145/1985793.1985855.

[15] F. F. J. Hermans. *Analyzing and visualizing spreadsheets*. TU Delft, Delft University of Technology, 2013.

[16] T. Herndon, M. Ash, and R. Pollin. Does high public debt consistently stifle economic growth? a critique of reinhart and rogoff. *Cambridge journal of economics*, 38(2):257–279, 2014. doi: 10.1093/cje/bet075.

[17] B. Hofer, A. Riboira, F. Wotawa, R. Abreu, and E. Getzner. On the empirical evaluation of fault localization techniques for spreadsheets. In *Fundamental Approaches to Software Engineering*, pages 68–82. Springer, 2013. doi: 10.1007/978-3-642-37057-1_6.

[18] D. Jannach and T. Schmitz. Model-based diagnosis of spreadsheet programs: a constraint-based debugging approach. *Automated Software Engineering*, pages 1–40, 2014. doi: 10.1007/s10515-014-0141-7.

[19] D. Jannach, T. Schmitz, B. Hofer, and F. Wotawa. Avoiding, finding and fixing spreadsheet errors–a survey of automated approaches for spreadsheet qa. *Journal of Systems and Software*, 94:129–150, 2014. doi: 10.1016/j.jss.2014.03.058.

[20] V. Käfer, D. Kulesz, and S. Wagner. What is the best way for developers to learn new software tools? an empirical comparison between a text and a video tutorial. Technical report, PeerJ Preprints, 2016.

[21] B. Kankuzi and J. Sajaniemi. A domain terms visualization tool for spreadsheets. In *Visual Languages and Human-Centric Computing (VL/HCC), 2014 IEEE Symposium on*, pages 209–210. IEEE, 2014. doi: 10.1109/vlhcc.2014.6883059.

[22] A. Kohlhase and M. Kohlhase. Compensating the computational bias of spreadsheets with mkm techniques. In *Intelligent Computer Mathematics*, pages 357–372. Springer, 2009. doi: 10.1007/978-3-642-02614-0_29.

[23] D. Kulesz. A spreadsheet cell-meaning model for testing. In *1st Workshop on Software Engineering Methods in Spreadsheets (SEMS)*, 2014.

[24] D. Kulesz and V. Käfer. Data for Spreadsheet Guardian, Nov. 2016. URL `https://doi.org/10.5281/zenodo.188896`.

[25] D. Kulesz and S. Zitzelsberger. Investigating effects of common spreadsheet design practices on correctness and maintainability. In *Proceedings of the EuSpRIG 2012 Conference*, Manchester, UK, 2012. ISBN 978-0-95699258-6-2.

[26] D. Kulesz, J. Scheurich, and F. Beck. Integrating anomaly diagnosis techniques into spreadsheet environments. In *Software Visualization (VISSOFT), 2014 Second IEEE Working Conference on*, pages 11–19. IEEE, 2014. doi: 10.1109/vissoft.2014.12.

[27] D. Kulesz, F. Toth, and F. Beck. Live inspection of spreadsheets. In *2nd Workshop on Software Engineering Methods for Spreadsheets (SEMS) (co-located with ICSE 2015)*, Florence, Italy, 2015.

[28] P. Liggesmeyer. *Software-Qualität: Testen, Analysieren und Verifizieren von Software.* Springer Science & Business Media, 2009. doi: 10.1007/978-3-642-77747-9_1.

[29] N. Marriott*. Using computerized business simulations and spreadsheet models in accounting education: a case study. *Accounting Education*, 13(sup1):55–70, 2004. doi: 10.1080/0963928042000310797.

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

[31] B. A. Nardi and J. R. Miller. Twinkling lights and nested loops: distributed problem solving and spreadsheet development. *International Journal of Man-Machine Studies*, 34(2):161–184, 1991. doi: 10.1016/0020-7373(91)90040-e.

[32] R. R. Panko. What we know about spreadsheet errors. *Journal of End User Computing*, 10:15–21, 1998. doi: 10.4018/joeuc.1998040102.

[33] R. R. Panko. Applying code inspection to spreadsheet testing. *Journal of Management Information Systems*, pages 159–176, 1999. doi: 10.1080/07421222.1999.11518250.

[34] R. R. Panko. Reducing overconfidence in spreadsheet development. *Reducing overconfidence in spreadsheet development*, 2003.

[35] R. R. Panko. Spreadsheets and sarbanes-oxley: Regulations, risks, and control frameworks. *Communications of the Association for Information Systems*, 17(1):29, 2006.

[36] R. R. Panko. Two experiments in reducing overconfidence in spreadsheet development. *Journal of Organizational and End User Computing*, 19(1):1, 2007. doi: 10.4018/9781599049458.ch062.

[37] R. R. Panko and D. N. Port. End user computing: The dark matter (and dark energy) of corporate it. In *System Science (HICSS), 2012 45th Hawaii International Conference on*, pages 4603–4612. IEEE, 2012. doi: 10.1109/hicss.2012.244.

[38] S. G. Powell, K. R. Baker, and B. Lawson. A critical review of the literature on spreadsheet errors. *Decision Support Systems*, 46(1):128–138, 2008. doi: 10.1016/j.dss.2008.06.001.

[39] L. Prechelt. *Kontrollierte Experimente in der Softwaretechnik: Potenzial und Methodik.* Springer, 2001.

[40] C. T. Ragsdale. *Spreadsheet modeling and decision analysis.* Thomson South-Western, 2004.

[41] L. M. Reder. Plausibility judgments versus fact retrieval: Alternative strategies for sentence verification. *Psychological Review*, 89(3):250, 1982. doi: 10.1037//0033-295x.89.3.250.

[42] C. M. Reinhart and K. S. Rogoff. Growth in a time of debt (digest summary). *American Economic Review*, 100(2):573–578, 2010. doi: 10.3386/w15639.

[43] G. Rothermel, L. Li, C. DuPuis, and M. Burnett. What you see is what you test: A methodology for testing form-based visual programs. In *Proceedings of the 20th international conference on Software engineering*, pages 198–207. IEEE Computer Society, 1998. doi: 10.1109/icse.1998.671118.

[44] C. Scaffidi, M. Shaw, and B. Myers. Estimating the numbers of end users and end user programmers. In *Visual Languages and Human-Centric Computing, 2005 IEEE Symposium on*, pages 207–214. IEEE, 2005. doi: 10.1109/vlhcc.2005.34.

[45] D. I. Sjøberg, T. Dybå, B. C. Anda, and J. E. Hannay. Building theories in software engineering. In *Guide to advanced empirical software engineering*, pages 312–336. Springer, 2008. doi: 10.1007/978-1-84800-044-5_12.

[46] L. Weitze, H. Söbke, and S. Berner. *Spreadsheet-based Decision Support Systems in Waste Management: A Collection of Working Examples.* Rhombos-Verlag, 2014.

[47] A. Wilson, M. Burnett, L. Beckwith, O. Granatir, L. Casburn, C. Cook, M. Durham, and G. Rothermel. Harnessing curiosity to increase correctness in end-user programming. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 305–312. ACM, 2003. doi: 10.1145/642611.642665.

[48] R. Zhang, C. Xu, S. Cheung, P. Yu, X. Ma, and J. Lu. How effective can spreadsheet anomalies be detected: An empirical study. *Journal of Systems and Software*, 2016. doi: 10.1016/j.jss.2016.03.061.