Avoiding, Finding and Fixing Spreadsheet Errors - A Survey of Automated Approaches for Spreadsheet QA

Dietmar Jannach[1a], Thomas Schmitz[a], Birgit Hofer[b], Franz Wotawa[b]

[a] *TU Dortmund, Germany*
[b] *TU Graz, Austria*

Abstract

Spreadsheet programs can be found everywhere in organizations and they are used for a variety of purposes, including financial calculations, planning, data aggregation and decision making tasks. A number of research surveys have however shown that such programs are particularly prone to errors. Some reasons for the error-proneness of spreadsheets are that spreadsheets are developed by end users and that standard software quality assurance processes are mostly not applied. Correspondingly, during the last two decades, researchers have proposed a number of techniques and automated tools aimed at supporting the end user in the development of error-free spreadsheets. In this paper, we provide a review of the research literature and develop a classification of automated spreadsheet quality assurance (QA) approaches, which range from spreadsheet visualization, static analysis and quality reports, over testing and support to model-based spreadsheet development. Based on this review, we outline possible opportunities for future work in the area of automated spreadsheet QA.

Keywords: Spreadsheet, Quality Assurance, Testing, Debugging

1. Introduction

Spreadsheet applications, based, e.g., on the widespread Microsoft Excel software tool, can nowadays be found almost everywhere and at all levels of

[1]Corresponding author: D. Jannach (dietmar.jannach@tu-dortmund.de), Postal address: TU Dortmund, 44221 Dortmund, Germany, T: +49 231 755 7272

organizations [1]. These interactive computer applications are often developed by non-programmers – that is, domain or subject matter experts – for a number of different purposes including financial calculations, planning and forecasting, or various other data aggregation and decision making tasks.

Spreadsheet systems became popular during the 1980s and represent the most successful example of the End-User Programming paradigm. Their main advantage can be seen in the fact that they allow domain experts to build their own supporting software tools, which directly encode their domain expertise. Such tools are usually faster available than other business applications, which have to be developed or obtained via corporate IT departments and are subject to a company's standard quality assurance (QA) processes.

Very soon, however, it became obvious that spreadsheets – like any other type of software – are prone to errors, see, e.g., the early paper by Creeth [2] or the report by Ditlea [3], which were published in 1985 and 1987, respectively. More recent surveys on error rates report that in many studies on spreadsheet errors at least one fault was found in every single spreadsheet that was analyzed [4]. Since in reality even high-impact business decisions are made, which are at least partially based on faulty spreadsheets, such errors can represent a considerable risk to an organization[2].

Overall, empowering end users to build their own tools has some advantages, e.g., with respect to flexibility, but also introduces additional risks, which is why Panko and Port call them both "dark matter (and energy) of corporate IT" [1]. In order to minimize these risks, researchers in different disciplines have proposed a number of approaches to avoid, detect or fix errors in spreadsheet applications. In principle, several approaches are possible to achieve this goal, beginning with better education and training of the users, over organizational and process-related measures such as mandatory reviews and audits, to better tool support for the user during the spreadsheet development process. In this paper, we focus on this last type of approaches, in which the spreadsheet developer is provided with additional software tools and mechanisms during the development process. Such tools can for example help the developer locate potential faults more effectively, organize the

[2]See `http://www.eusprig.org/horror-stories.htm` for a list of real-world stories or the recent article by Herndorn et al. [5] who found critical spreadsheet formula errors in the often-cited economic analysis of Reinhart and Rogoff [6].

test process in a better structured way, or guide the developer to better spreadsheet designs in order to avoid faults in the first place. The goals and contributions of this work are (A) an in-depth review of existing works and the state-of-the-art in the field, (B) a classification framework for approaches to what we term "automated spreadsheet QA", and (C) a corresponding discussion of the limitations of existing works and an outline of perspectives for future work in this area.

This paper is organized as follows. In Section 2, we will define the scope of our research, introduce the relevant terminology and discuss the specifics of typical spreadsheet development processes. Section 3 contains our classification scheme for approaches to automated spreadsheet QA. In the Sections 4 to 9, we will discuss the main ideas of typical works in each category and we will report how the individual proposals were evaluated. Section 10 reviews the current research practices with respect to evaluation aspects. In Section 11, we point out perspectives for future works and Section 12 summarizes this paper.

2. Preliminaries

Before discussing the proposed classification scheme in detail, we will first define the scope of our analysis and sketch our research method. In addition, we will briefly discuss differences and challenges of spreadsheet QA approaches in comparison with tool-supported QA approaches for traditional imperative programs.

2.1. Scope of the analysis, research method, terminology

Spreadsheets are a subject of research in different disciplines including the fields of Information Systems (IS) and Computer Science (CS) but also fields such as Management Accounting or Risk Management, e.g., [2] or [7].

Scope. In our work, we adopt a Computer Science and Software Engineering perspective, focus on tool support for the spreadsheet development process and develop a classification of automated spreadsheet QA approaches. Examples for such tools could be those that help the user locate faults, e.g., based on visualization techniques or by directly pointing them to faulty cells, or tools that help the user avoid making faults in the first place, e.g., by supporting complex refactoring work. Spreadsheet error reduction techniques from the IS field, see, e.g., [8], and approaches that are mainly based on "manual" tasks like auditing or code inspection will thus not be in the focus of our work.

Research on spreadsheets for example in the field of Information Systems often covers additional, more user-related, or fundamental aspects such as error types, error rates and human error research in general, the user interface, cognitive effort and acceptance issues of tools, user over-confidence, as well as methodological questions regarding the empirical evaluation of systems, see, e.g., [9, 10, 11, 12, 13, 4]. Obviously, these aspects and considerations should be the basis when designing an automated spreadsheet QA tool that should be usable and acceptable by end users. In our work and classification, we however concentrate more on the provided functionality and the algorithmic approaches behind the various tools. We will therefore discuss the underlying assumptions for each approach, e.g., with respect to user acceptance or evaluation, only as they are reported in the original papers. Still, in order to assess the overall level of research rigor in the field, we will report for each class of approaches how the individual proposals were evaluated or validated.

These insights will be summarized and reviewed in Section 10. In this section, we will also look at the difficulties of empirically evaluating the true value of spreadsheet error reduction techniques according to the literature from the IS field.

Regarding tool support in commercial spreadsheet environments, we will briefly discuss the existing functionality of MS Excel and comparable systems in the different sections. Specialized commercial auditing add-ons to MS Excel usually include a number of QA tools. As our work focuses more on advanced algorithmic approaches to spreadsheet QA, we see the detailed analysis of current commercial tools to be beyond the scope of this paper. Finally, we will also not cover fault localization or avoidance techniques for the imperative programming extensions that are typically part of modern spreadsheet environments.

Research method. For creating our survey, we conducted an extensive literature research. Papers about spreadsheets are published in a variety of journals and conference proceedings. However, there exists no publication outlet which is only concerned with spreadsheets, except maybe for the application-oriented EuSpRIG conference series[3]. In our research, we therefore followed an approach which consists both of a manual inspection of relevant journals

[3]`http://www.eusprig.org`

and conference proceedings as well as searches in the digital libraries of ACM and IEEE. Typical outlets for papers on spreadsheets which were inspected manually included both broad Software Engineering conferences and journals such as ICSE, ACM TOSEM, or IEEE TSE. At the same time, we reviewed publications at more focused events such as ICSM or the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). In addition, major IS journals and events such as Information Systems Research, ACM TOIS or ICIS were considered in our research.

When searching the digital libraries, we started by looking for papers containing the term "spreadsheet" in the title or abstract. From the 400 to 500 results returned by the search engines of the libraries, we manually inspected the abstracts. Provided their scope was relevant for our research, we categorized them according to the categorization framework described in Section 3, and followed the relevant references mentioned in the articles.

Terminology. Regarding the terminology used in the paper, we will use the terms "spreadsheet", "spreadsheet application", or "spreadsheet program" more or less interchangeably as often done in the literature. When we refer to the underlying software system to create spreadsheets (e.g., Microsoft Excel), we will use the term "spreadsheet environment" or "spreadsheet tool". In some papers, the term "form-based visual languages" is used [14] to describe the more general family of such systems. In our work, we will however rely on the more widespread term "spreadsheet".

There are a number of definitions of the terms "error", "fault", and "failure" in the literature. According to IEEE standards for Software Engineering an "error" is a misapprehension on side of the one developing a spreadsheet caused by a mistake or misconception occurring in the human thought process. A "fault" is the manifestation of an "error" within a spreadsheet which may be causing a "failure". A "failure" is the deviation of the observed behavior of the spreadsheet from the expectations. In the literature on spreadsheets, in particular the terms "fault" and "error" are often used in an interchangeable manner. Surveys and taxonomies of spreadsheet problems like [15], [16], or [17], for example, more or less only use the term "error". In our review, we will – in order to be in line with general Software Engineering research – use the term "fault" instead of "error" whenever appropriate.

2.2. Specifics of spreadsheets and their QA tools

The requirements for automated approaches for spreadsheet QA can be quite different from those of tools that are used with typical imperative languages. In [14], Rothermel et al. illustrated some of the major differences in the context of spreadsheet testing. Many of the aspects mentioned in their work however do not only hold for the testing domain, but should in general be taken into account when developing tools supporting the spreadsheet developer.

First, the way in which users interact with a spreadsheet environment is largely different from how programs in imperative languages are developed. In spreadsheets, for example, the user is often constructing a spreadsheet in an unstructured incremental process using some test data. For the given test data, the user continuously receives visual and immediate feedback. To increase the chances of being accepted by developers, any supporting tool should therefore be designed in a way that it supports such an incremental development process. In that context, trying to de-couple the actual implementation tasks from other tasks like testing or design could be problematic. At the same time, being able to provide immediate feedback in the incremental process appears to be crucial.

Second, the computation paradigm of spreadsheets is quite different from imperative programs. The basic nature of spreadsheets is that their computations – the "evaluation" of the program – are driven by data dependencies between cells and explicit control flow statements are only contained in formulas in the cells. When designing supporting QA mechanisms and tools, this aspect should be kept in mind. For example, when adapting existing QA approaches from imperative programs to spreadsheet development, there might be different characteristics and quality measures that have to be considered. Data dependencies can, for example, be more relevant than the control flow. At the same time, the conceptual model of the users might be rather based on the data and formula dependencies than on execution orders.

Third, spreadsheet programs are not only based on a simpler computational model than imperative programs, their "physical" layout – i.e., the spatial arrangement of the labels and formulas – is typically strongly determined by the intended computation semantics. This spatial information can be used by automatic QA tools, e.g., to detect inconsistencies between neighboring cells and to assess the probability of a formula being semantically correct, to automatically infer label information, or to rank change suggestions in goal-directed debugging approaches [18].

Finally, developers of spreadsheets are mostly non-professional programmers. Developers of imperative programs often have a formal training or education in software development and are generally aware of the importance of systematic QA processes. People developing spreadsheets are mostly non-programmers and may have limited interest and awareness when it comes to investing additional efforts in QA activities like testing or refactoring. Therefore, any QA methodology and the corresponding tool support should make it easy for a non-programmer to understand the value of investing the additional efforts. To cope with this, approaches for spreadsheet QA should not require special training or an understanding of the theory behind the approach. The used language should avoid special terminology from the underlying theory or technique. When discussing the different approaches in the next sections, we will therefore briefly discuss the approaches from the perspective of usability and what is expected from the end user.

3. A classification of automated approaches to spreadsheet QA

Generally, we classify the various spreadsheet QA approaches into two main categories depending on their role and use in the development lifecycle.

- "Finding and fixing errors" is about techniques and tools that are mainly designed to help the user detect errors and understand the reasons for the errors. These tools are typically used by the developer or another person, e.g., an auditor or reviewer, during or after the construction of the spreadsheet.

- "Avoiding errors" is about techniques and tools that should help the developer create spreadsheets that do not have errors in the first place. These approaches support the creation process of spreadsheets.

In our work, we however aim to develop a finer-grained categorization scheme to classify the existing approaches to automated spreadsheet QA. The main categories of our proposed scheme are shown in Table 1.

The categories (1) and (2) can serve both the purpose of finding and avoiding errors. A good visualization, for example, of cell dependencies, helps the user to spot a problem. At the same time, a visualization can be used to highlight cells or areas for which there is a high probability that an error will be made in the future, for example, when there are repetitive structures in the spreadsheet. Static analyses can both identify already existing problems such

	Finding errors	Avoiding errors
(1) Visualization-based approaches	x	x
(2) Static code analysis & reports	x	x
(3) Testing approaches	x	
(4) Automated fault localization & repair	x	
(5) Model-driven development approaches		x
(6) Design and maintenance support		x

Table 1: Overview of main categories of automated spreadsheet QA.

as references to empty cells and serve as indicators for potential problems, e.g., by listing formulas which are too complex. The techniques falling into the categories (3) and (4) mainly contribute to the problem of "Finding and fixing errors" as they either help the user to identify the existence of a problem or to localize the error causes. The methods in the categories (5) and (6) often provide means to avoid errors, e.g., by supporting the refactoring process or adding an additional layer of abstraction. In general, the schema shown in Table 1 serves as a rough guideline for the categorization. There might be individual techniques within certain subcategories, which can serve both the purposes of finding and avoiding errors.

We summarize the main idea of the individual families of approaches as follows.

- Visualization-based approaches: These approaches provide the user with a visually enhanced representation of some aspects of the spreadsheet to help him or her understand the interrelationships and dependencies between cells or larger blocks of the spreadsheet. These visualizations help the user to quickly detect anomalies and irregularities in the spreadsheet.

- Static analysis & reports: These approaches are based on static code analysis and aim to point the developer to potentially problematic areas of the spreadsheet. Examples of techniques include "code smells" or the detection of data clones but also the typical family of techniques found in commercial tools capable of detecting circular dependencies or reporting summaries about unreferenced cells.

- Testing-based techniques: The methods in this category aim to stimulate and support the developer to systematically test the spreadsheet application during or after construction. The supporting tools for example include mechanisms for test case management, the automated generation of test cases or analysis of the test coverage.

- Automated fault localization & repair: The approaches in this category rely on a computational analysis of possible causes of an error or unexpected behavior (algorithmic debugging). They rely on additional input by the developer such as test cases or statements about the correctness of individual cells. Some approaches are also capable of providing "repair" suggestions.

- Model-driven development approaches: Methods in this category mainly adopt the idea of using (object-oriented) conceptual models as well as model-driven software development techniques, which are nowadays quite common in the software industry. The typical advantages of such approaches include the introduction of additional layers of abstraction or the use of code-generation mechanisms.

- Design and maintenance support: The approaches in this category either help the spreadsheet developer to end up with better error-free designs or support him or her during spreadsheet construction. The mechanisms proposed in that context for example include automated refactoring tools, methods to avoid wrong cell references, and exception handling.

Table 2 outlines the structure of the main sections of the paper.

4. Visualization-based approaches

Visualization-based approaches are helpful in different ways. They can, for example, help a developer or reviewer understand a given spreadsheet and its formulas, so that he or she can check it more easily for potential errors or bad design. In addition, visualizations are a good starting point for a reviewer other than the original author of the spreadsheet to understand

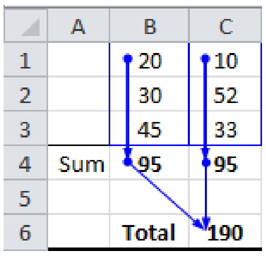
4. Visualization-based approaches	4.1. Dataflow and dependency visualization 4.2. Visualization of related areas 4.3. Semantic-based visualizations 4.4. Information Visualization approaches
5. Static code analysis & reports	5.1. Unit and type inference 5.2. Spreadsheet smells 5.3. Static analysis in commercial tools
6. Testing approaches	6.1. Test adequacy and test case management 6.2. Automated test case generation 6.3. Assertion-based testing 6.4. Test-driven spreadsheet development
7. Automated fault localization & repair	7.1. Trace-based candidate ranking 7.2. Constraint-based fault localization 7.3. Repair approaches
8. Model-driven development approaches	8.1. Declarative spreadsheet models 8.2. Spreadsheet templates 8.3. Object-oriented visual models 8.4. Relational spreadsheet models
9. Design and maintenance support	9.1. Reference management 9.2. Exception handling 9.3. Changes and spreadsheet evolution 9.4. Refactoring 9.5. Reuse

Table 2: Outline of the main parts of the paper.

its basic structure and the dependencies between the formulas. A typical application scenario is thus the use of visualizations in the auditing process, see, e.g., [19] or [20]. We categorize the different approaches for spreadsheet visualization from the literature as follows.

4.1. Dataflow and dependency visualization

A number of approaches visualize the dataflow in the spreadsheet and the corresponding dependencies of the formula cells, see, e.g., [21, 22, 23, 24, 25, 26, 27, 28] and [29]. In many cases, arrows are used to represent the usage of a cell in a formula, which is a standard feature of commercial spreadsheet environments like MS Excel as shown in Figure 1.



	A	B	C
1		20	10
2		30	52
3		45	33
4	Sum	95	95
5			
6		Total	190

Figure 1: Simple dependency visualization in MS Excel.

One of the earlier works going beyond simple dependency visualizations was presented by Davis in [19]. In addition to the use of arrows within the spreadsheet to visualize dependencies between cells, spreadsheets are visualized as graphs. The graph visualization is based on a spreadsheet description language proposed by Ronen et al. earlier in [30] to model the functionality of a spreadsheet. In these graphs, the cells correspond to the nodes and edges represent dependencies between cells. Two experiments with users were performed in which the new techniques – arrows and graphs – were compared with the existing features of MS Excel 3.0. At that time, MS Excel could only provide a listing of cell dependencies but had no graphical visualization. In the first experiment, 27 students had to find all dependent cells of a given cell. In the second experiment involving 22 students, the task was to correct an observed fault in a given cell. Overall, both new approaches were found to be more helpful for the given tasks than the standard functionality of MS Excel. Interestingly, the simple arrow-based approach was outperforming the more complex dependency graph approach.

Another more advanced method for dependency visualization was presented by Igarashi et al. in [21]. In their work, the authors rely on an animated and interactive visualization approach and "fluid interfaces" to give the user a better understanding of the data flow in the spreadsheet. Advanced animations are used to visualize which cells are input to other calculations. These animations made it possible to represent comparably complex dependencies in a visual form. In addition, users could interact with the visualizations and thereby manipulate the references through drag and drop operations, e.g., to move references, scale referenced arrays or interactively fill areas with formulas. To evaluate their approach, a prototype system was built, tested with comparably small spreadsheets and informally discussed in their paper. A study with real users and with large spreadsheets was however not done.

A different approach to visualize complex dependencies in spreadsheets is to represent parts of the spreadsheet in three-dimensional space as done, e.g., in [22] or [25]. In the approach proposed by Shiozawa et al. [22], for example, the spreadsheet is rendered in 3D and the user can interactively manipulate the visualization and lift cells or groups of cells. The connected cells are lifted to some extent based on a distance metric, allowing the user to better distinguish between cell dependencies, which are drawn as arrows in the 3D space. Similar to the work of Igarashi et al. mentioned above, the evaluation of the approach was limited to an informal discussion of a prototype system.

In [23], Chen and Chan proposed additional techniques for cell dependency visualization in spreadsheets. One of the main ideas is to visualize the dependencies between larger blocks of formulas in neighboring cells instead of displaying arrows between individual cells as done, e.g., in MS Excel. An alternative visualization of dependencies between larger blocks of the spreadsheet is proposed by Kankuzi and Ayalew in [26] and [27]. In their approach, the cells are first clustered and the resulting dependencies are then displayed as a tree map in an external window.

A more recent proposal to spreadsheet visualization was made by Hermans et al. in [28] and [29]. In their approach, the user can inspect the data flows within the spreadsheet on different levels of detail. On the global view, only the different worksheets of the spreadsheet and their dependencies are shown; on the lowest level, dependencies between individual cells are displayed. On an intermediate level, the spreadsheet is sliced down to smaller areas of geometrically adjacent cells. Beside the arrows that are used to indicate dependencies, their visualization method also displays the mathematical functions used in the calculations. In [28], Hermans et al. evaluate their dataflow visualization in two steps. In the first round, an interview involving 27 subjects about the general usefulness of such diagrams was conducted; the second part consisted of 9 observational case studies in which the task for the participants consisted in transferring or explaining their complex real-world spreadsheets to another person. The observations during the study and the qualitative feedback obtained in the post-experiment interviews indicated that the proposed techniques are well suited for the task of spreadsheet comprehension and helpful for auditing and validation purposes.

4.2. Visualization of related areas

Different methods and tools for identifying and visualizing semantically related or structurally similar blocks of cells were proposed in [31, 32, 33, 34]. These blocks can be highlighted using different colors to make it easier for the user to understand the logical structure of the spreadsheet or to identify irregularities as done in [35].

In the work of Mittermeir and Clermont [31], the concept of "logical areas" is introduced as a first step in their approach. Such areas can be automatically identified by looking for structurally similar (equivalent) formulas in different areas of the spreadsheet. Such areas are for example the result of a formula copy operation by the user during the construction process. Since a preliminary study with a prototype tool on 78 large real-world spreadsheets revealed that relying on logical areas alone reaches its limits for larger spreadsheets, the concept of "semantic classes" was introduced. In this semi-automated approach, the user manually specifies related areas in the spreadsheet. Based on this user-provided input and information about the spatial arrangement of potentially related cells, further reasoning about areas with high similarity in the spreadsheet can be performed. The work was later on improved in [32], where semantic classes were identified based on the information contained in label cells and a set of heuristics. An alternative method for decomposing a given spreadsheet for the purpose of visualization was presented in [33, 34]. In that work, the identification of areas is based on properties of the data flow in the spreadsheet.

Both the approaches proposed in [31] and [33, 34] were discussed in the corresponding papers using one artificial spreadsheet with a few dozen formulas as an example. In [36], Clermont et al. report that the approach from [31] was used to audit real-world spreadsheets, leading to detected error rates in spreadsheets that are in line with those reported in the literature[4]. An evaluation regarding the question to which extent the additional tool support increases the error detection rate or speeds up the inspection process was however not conducted.

In [20], Sajaniemi proposes two further visualization approaches called S2 and S3. The basic idea is to detect equivalent formulas in blocks of

[4]See, e.g., [4], for a discussion of error rates.

cells and visualize the dependencies between individual blocks. A theoretical comparison with the visualization techniques proposed in [19, 21, 30, 37, 38, 39] and the auditing functionality of MS Excel 7.0 was done, showing different advantages of their approaches. Beside the visualization approaches, Sajaniemi's work represents an interesting methodological contribution, as he proposes a systematic way of theoretically analyzing and comparing different visualization techniques.

4.3. Semantic-based visualizations

The missing semantics for the formulas of a spreadsheet, which is caused by the use of numbered cell references instead of information-carrying names, is a well-known problem in spreadsheet research. This was already discussed in an early work by Hendry et al. [37], where they proposed a system for annotating cells in order to describe their semantics.

The work by Chadwick et al. presented in [40] is based on the observation of two typical types of errors that are made by many spreadsheet users when creating formulas: (1) formulas sometimes reference the wrong cells as inputs; (2) formulas are sometimes copied incorrectly. To deal with the first problem, the authors propose different techniques to make the formulas more intuitively readable. One of the techniques is for example to transform a formula like `=SUM(F6:F9)` into `=SUM(Night Wages_Grade1:Night Wages_Grade4)` based on cell labels within the spreadsheet. Another idea is to represent complex formulas in a visual form. For this purpose, the formulas are decomposed, cell references are replaced with readable names and operators are translated into natural language such that the logic of the formula can be understood more easily. As a solution to problems arising from wrongly copied formulas, Chadwick et al. propose to use visual indicators and mark copied cells and their origin with the same color and add an additional comment to the original cell.

The different methods of the formula visualization were evaluated through a survey involving 63 students. The students had to rank the methods with respect to clarity and ease of understanding. The most visual method was ranked first in that survey; interestingly, however, the usual notation of Excel with cell references was ranked second and was better accepted than the above-described approach in which cell references were replaced with labels. The visualization that was used to highlight copied formulas was evaluated through a small user study with 5 students. The participants had to construct a spreadsheet and were provided with the additional visualization in that process. The results indicated that the participants liked the approach as they confirmed its usefulness to check a spreadsheet for correctness.

Nardi and Serrecchia [41] propose a more complex approach to reconstruct the underlying model of the spreadsheet, where a knowledge base is constructed and reasoning mechanisms are developed to describe calculation paths of individual cells with descriptive names. Although the approach was implemented prototypically, a systematic evaluation was not done.

4.4. Information Visualization approaches

In [42], Brath and Peters apply techniques from the field of Information Visualization to spreadsheet analysis. These visualizations support the developer in the process of detecting anomalies in the spreadsheet. In contrast to some of the approaches described so far, the aim is thus not to visualize the data flow or the structures of the spreadsheet but the data itself. To that purpose, a 3D representation is proposed, where the cell values are for example shown as bars instead of numbers. Higher numbers result in higher bars. Using the corresponding tool, the user can navigate through the 3D space, detect outliers or unexpected patterns in the data. The general feasibility of the method is informally discussed in the paper based on two case studies[5].

In general, a number of techniques from the field of Information Visualization, e.g., the "fisheye"-based approach described in [43], can in principle be applied to visualize large tabular data in spreadsheets for inspection purposes. The work of Ballinger et al. [24] is an example for such a work that relies, among others, on 3D diagrams and a fisheye view to visualize data dependencies. Overall, however, the number of similar works that rely on Information Visualization techniques appears to be limited.

4.5. Discussion

Quite a number of proposals have been made in the literature that aim to represent certain aspects of a spreadsheet in visual form. The purposes of the visualization include in particular spreadsheet comprehension, e.g., in an auditing context, and in particular anomaly and error detection.

Regarding the research methodology, only in very few and more recent papers a systematic and rigorous experimental evaluation of the proposed

[5]The general idea presented in [42] was later on implemented in a commercial tool by Oculusinfo Inc. `http://www.oculusinfo.com`.

methods has been done. In most cases, the validation is limited to qualitative interviews or surveys involving a comparably small set of participants or the informal discussion of prototype systems and individual use cases. The true applicability and usability for end users of many approaches is often unclear. Many works in that field would thus benefit if more systematic evaluations and user studies were performed as it is done for example in the fields of Human Computer Interaction and Information Systems. Possible evaluation approaches for visualization techniques include spreadsheet construction, inspection, or understanding exercises as done in IS spreadsheet research, e.g., in [9, 10, 44] or [45], but also observational approaches based, e.g., on think-aloud protocols or usage logs.

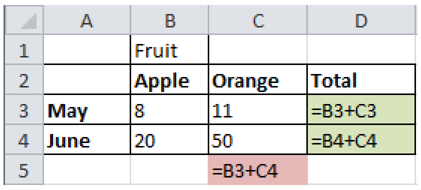
Regarding practical tools, the market-leading tool MS Excel incorporates only a small set of quite simple visualization features for spreadsheet analysis or debugging. Cell dependencies can be visualized as shown in Figure 1 or as colored rectangles highlighting the referenced cells of a formula. In addition, a small visual clue – a green triangle at the cell border – is displayed when some of the built-in error checking rules are violated. With respect to the idea of using "semantic" variable names instead of cell references, spreadsheet systems like MS Excel allow the developer to manually assign names to cells or areas to make the spreadsheets more comprehensible.

5. Static Analysis and Reports

Static code analysis or reporting-based approaches analyze the formulas of the spreadsheets and show possible faults or bad spreadsheet design that can lead to faults in the future. In contrast to automated fault localization approaches described in Section 7, the approaches in this category do not use the values in cells or information from test cases to find errors. Instead, they rather analyze the formulas themselves and the dependencies between them, look at static labels, and determine other structural characteristics of the spreadsheets.

5.1. Unit and type Inference

A major research topic in the last decade was related to "unit and type inference" [46, 47, 48, 49, 50, 51, 52, 53, 54]. The main idea of these approaches is to derive information about the units of the input cells and use this information to assess if the calculations in the formulas can be plausible with respect to the units of the involved cells. To obtain information about a cell's unit, its headers can be used. Figure 2 shows an example illustrating the idea [46]. The formulas in cell D3 and D4 can be considered legal. They combine apples with oranges, which are both of type fruit. In contrast, C4 could be considered illegal, as cells with incompatible units are combined, i.e., apples from May with oranges from June. With the help of such a unit inference mechanism, the semantics of a calculation can be checked for errors. The process of deriving the unit information from header cells is called header inference.



	A	B	C	D
1		Fruit		
2		**Apple**	**Orange**	**Total**
3	**May**	8	11	=B3+C3
4	**June**	20	50	=B4+C4
5			=B3+C4	

Figure 2: Unit inference example; adapted from [46].

The idea to use a unit inference system to identify certain kinds of potential errors was introduced by Erwig and Burnett in 2002 [46]. In their initial approach, a cell could have more than one unit. However, this first work provided no explicit procedures of how the header inference should be done. In addition, there were some limitations regarding certain operators. Later on, a header inference approach was proposed in [49], so that the system could work without or with limited user interaction. Other improvements to the basic approach including various forms of more sophisticated reasoning were put forward in [47, 49, 51, 52, 53, 54]. In [53] and [54], for example, the idea is to do a semantic analysis of the labels in order to map the labels to known units of measurements. Based on this information, more precise forms of reasoning about the correctness of the calculations become possible. In contrast to the latter approaches based on semantic analysis of labels, in the work described in [55] the assumption is that the user manually enters the units and labels for the input cells and the system is then able to make the appropriate inferences for the output cells.

The idea of considering relationships between headers ("is-a", "has-a"), a different reasoning strategy and a corresponding tool capable of processing Excel documents were presented in [48] and [50]. The first work for this approach [46] included no evaluation. A first small evaluation with 28 spreadsheets was done in [49] to test the header inference and the error detection mechanism. For both sets of spreadsheets the numbers of detected errors and incorrect header and unit inferences were counted. The header and unit inferences were checked by hand and the system showed good accuracy. Regarding error detection, the system was capable of finding errors in 7 student spreadsheets. Since the total number of errors is not reported, no information about the error detection rate is available. In later papers on the topic including [53, 54, 56, 57], the systems were evaluated by comparing them with previous approaches using the EUSES spreadsheet corpus [58]. Again, the evaluation was done by counting and comparing the detected errors using different approaches.

5.2. Spreadsheet smells

The term "spreadsheet smells" was derived from code smells in software maintenance [59], where it is used for referring to bad code design. These designs are not necessarily faults themselves, but can lead to faults during the future development of the software, for example, when the software is to be refactored or expanded. A typical example for a code smell is the *duplication* of code fragments. If the same part of code is contained several times in a program, it is usually better to place it into a function so that eventual changes to the code fragment have only to be done once. Duplicated code in addition makes the code harder to read.

Spreadsheet smells are a comparably recent topic in spreadsheet research. Hermans and colleagues were among the first to adapt the concept of code smells to the spreadsheet domain, see, e.g., [60, 61, 62]. Similar ideas have already been proposed earlier in the context of spreadsheet visualization, where heuristics were used to identify irregularities in spreadsheets [35, 42].

In general, spreadsheet smells are heuristics which describe bad designs that can lead to errors when the spreadsheet is changed or when a new instance of it is created with new input data. In [60], Hermans et al. propose so-called "inter-worksheet smells". These smells indicate bad spreadsheet design based on the analysis of dependencies between different worksheets. If, for example, a formula has too many references to another worksheet, it probably should be moved to that worksheet. In addition to adapting the code smells to the spreadsheet domain, Hermans et al. also introduced metrics to discover these smells and a means to visualize them in their own worksheet dependency visualization approach [29] (see Section 4.1). "Formula smells" were discussed in [61]. These smells represent bad designs of individual formulas, e.g., when a formula is too complex. Later on, Hermans et al. propose a method for finding data clones in a spreadsheet [62]. To evaluate their spreadsheet smell approach, Hermans et al. performed both a quantitative and a qualitative evaluation in [60]. For the quantitative evaluation, the EUSES spreadsheet corpus was searched for the different types of inter-worksheet spreadsheet smells to understand how frequent these smells occur. In the qualitative evaluation, they identified smells in the spreadsheets of 10 professional spreadsheet developers and discussed the smells with the developers. The evaluation proved that the detected smells point to potential weaknesses in the spreadsheet designs. The same type of evaluation was done for the formula smells in [61].

The work of Cunha et al. in [63] is also based on the idea of spreadsheet smells. In contrast to the works by Hermans et al., they did not aim at adapting known code smells but rather tried to identify spreadsheet-specific smells by analyzing a larger corpus of spreadsheets.

5.3. Static analysis in commercial tools

Static analysis techniques are often part of commercial spreadsheet environments and spreadsheet auditing tools. As mentioned in Section 4.5, MS Excel, for example, is capable of visualizing "suspicious" formulas. A pre-defined set of rules is checked to determine if a formula is suspicious, e.g., when it refers to an empty cell or when a formula omits cells in a region. Typical spreadsheet auditing tools such as the "Spreadsheet Detective" [64][6] also strongly rely on the identification of such suspicious formulas using static analyses and produce corresponding reports. To identify these formulas, different heuristics are used, which can take the formula complexity into account, e.g., by checking if there are multiple IF-statements. Other indicators include duplicated named ranges or numbers quoted as text. Some audit tools also comprise mechanisms to support spreadsheet evolution and versioning, e.g., by listing the differences between two variants of the same spreadsheet [65]. An in-depth analysis of these tools is however beyond the scope of our work.

5.4. Discussion

The goal of static analysis techniques usually is to identify formulas or structural characteristics of spreadsheets which are considered to be indicators for potential problems. The accuracy of these methods depends on

[6]http://www.spreadsheetdetective.com

the quality of the error detection heuristics or metrics that are used to define a smell. Generally, such static analysis tools represent a family of error detection methods which can be found in commercial tools.

Type and unit systems go beyond pure analysis approaches and try to apply additional inferencing to detect additional types of potential problems and can be considered a lightweight semantic approach. While such inferencing techniques have the potential of identifying a different class of errors, there is also some danger that they detect too many “false positives”.

From the perspective of the research methodology, both quantitative and qualitative methods are applied in particular in the more recent works. Evaluations are done using existing document corpora and spreadsheets created by students or professionals. However, a potential limitation when using the EUSES corpus in that context is that the intended semantics of the formulas which are considered faulty by a certain technique are for most formulas not known. Thus, we cannot determine with certainty if the formula is actually wrong and the technique was successful. From the end-user perspective, many results of a static analysis, e.g., code smells, can be quite easily communicated and explained to the user.

With respect to “smell” detection based on complex unit inference, Abraham et al. [66] conducted a think-aloud study involving 5 subjects. One goal of the study was to evaluate if the users would understand the underlying concepts well enough to correct the errors reported by their tool. Their observations indicate that the subjects, who were trained on the topic before the experiment, did understand how to interpret the feedback by the tool and correct the unit errors without the need to understand the underlying reasoning process.

6. Testing approaches

In professional software development processes, systematic testing is crucial for ensuring a high quality level of the created software artifacts. Typically, testing activities are performed by different groups of people in the various phases of the process; both manual as well as automated test procedures are common. As non-professional spreadsheet developers mostly have no proper education in Software Engineering, the testing process is assumed to be much less structured and systematic. In addition, the developer in many cases might be the only person that performs any tests.

Given the immediate-feedback nature of spreadsheets, testing can be done by simply typing in some inputs and checking if the intermediate cells and output cells contain the expected values. Commercial spreadsheet tools such as MS Excel do not provide any specific mechanisms to the user for storing such test cases or running regression tests. Furthermore, these tools do not help the developer assess if a sufficient number of tests has been made. In the following, we review approaches that aim at transferring and adapting ideas, concepts, tools and best-practice approaches from standard software testing to the specifics of spreadsheet development.

6.1. Test adequacy and test case management

A number of pioneering works in this area have been done by the research group of Burnett, Rothermel and colleagues at Oregon State University. Already in 1997, they discussed test strategies and test-adequacy criteria for form-based systems and later on proposed a visual and incremental spreadsheet testing methodology called "What You See Is What You Test" (WYSIWYT) [67, 14, 68]. During the construction of the spreadsheet, the user can interactively mark the values of some derived cells to be correct for the currently given inputs. Based on these tests, the system determines the "testedness" of the spreadsheet. This is accomplished through an automatic evaluation of a test adequacy criterion which is based on an abstract model of the spreadsheet, spreadsheet-specific "definition-use" (du) associations and dynamic execution traces. Later on, several improvements to this approach were proposed, such as scaling it up to large homogeneous spreadsheets that are often found in practice, adding support for recursion, or dealing with questions of test case reuse [69, 70, 71, 72]. The approach was ported from Forms/3 to Microsoft Excel with additional support of special features such as higher-order-functions and user defined functions [73]. In [74], Randolph et al. presented an alternative implementation of the WYSIWYT approach, which was designed in a way that it can be used in combination with different spreadsheet environments.

In [68], the results of a detailed experimental evaluation of the basic approach are reported whose aim was to assess the efficacy of the approach and how "du-adequate" test suites compare to randomly created tests. In their evaluations, they used 8 comparably small spreadsheets, in which experienced users manually injected a single fault. Then, a number of du-adequate and random pools of test cases were created. The analysis of applying these tests among other things revealed that the du-adequate pools outperformed random pools of the same size in all cases with respect to their ability of detecting the errors. A further study involving 78 subjects in which the efficiency and effectiveness of the approach was tested is described in [75].

6.2. Automated test case generation

When using the WYSIWYT approach, the spreadsheet developer receives feedback about how well his or her spreadsheet is tested. Still, the developer has to specify the test cases manually. To support the user in this process, Fisher et al. proposed techniques for the automated generation of test cases [76, 77].

In these works, two methods for generating values for a test case were evaluated. The "Random" method randomly generates values and checks if their execution uses a path of a so far unvalidated definition-use pair. The second, goal-oriented method called "Chaining" iterates through the unvalidated definition-use pairs and tries to modify the input values in a way that both the definition and the use are executed. If the generation of input values for a new test case is successful, the user only has to validate the output value to obtain a complete test case. To assess the effectiveness and efficiency of their approach an offline simulation-based study without real users based on 10 comparably small spreadsheets containing only integer type cells was performed [77]. The results clearly showed that the "Chaining" method was more effective than the "Random" method.

In [78], the AutoTest tool was presented, which implements a different strategy for automatic test case generation and uses constraint solving to search for values that lead to the execution of the desired definition-use pairs. This method is guaranteed to generate test cases for all feasible definition-use pairs. The method was compared with the previously described method from [77] using the same experimental setup and showed that AutoTest was both more effective and could generate the test cases faster.

6.3. Assertion-based testing

A very different approach for users to test and ensure the validity of their spreadsheets was presented by Burnett et al. in [79]. In this work, the concept of assertions, which can be found in some imperative languages, was transferred to spreadsheets. Assertions in the spreadsheet domain (called "guards" here) correspond to statements about pre- and post-conditions about allowed cell values in the form of Boolean expressions. The assertions are provided by the end user through a corresponding user-oriented tool and automatically checked and partially propagated through the spreadsheet in the direction of the dataflow. Whenever a conflict between an assertion and a cell value or between a user given and a propagated assertion is detected, the user is pointed to this problem through visual feedback.

In [79] and [80] different controlled experiments were performed to evaluate the approach. The experimental setup in [79] consisted of a spreadsheet testing and debugging exercise in which 59 subjects participated. About half of the subjects were using an “assertion-enabled” development environment, whereas the other group used the same system without this functionality. The analysis revealed that assertions helped users to find errors both more effectively and efficiently across a range of different error types. A post-experiment questionnaire furthermore showed that the users not only understood and liked using the assertions but that assertions are also helpful to reduce the users’ typical over-confidence about the correctness of their program. Ways of how to extend the concept of guards to multiple cells were discussed in [81]; a small think-aloud study indicated that such mechanisms must be carefully designed as the expectations around the reasoning behind such complex guards was not consistent across users.

6.4. Test-driven spreadsheet development

Going beyond individual techniques for test case management and test case generation, McDaid et al. in [82] address the question if the principle of test-driven development (TDD), which received increased attention in the Software Engineering community, is applicable in spreadsheet development processes. Following this principle, the user iteratively creates test cases first that define the intended spreadsheets functionality and writes or changes formulas afterwards to fulfill the test. This continuing and systematic form of testing shall help to minimize the number of faults that remain in the final spreadsheet.

In their work, the authors argue that spreadsheets are well suited for the TDD principle and present a prototype tool. To evaluate the approach, 4 users with different background in spreadsheet expertise and TDD were asked to develop different spreadsheets and corresponding test cases. From the subsequent interviews, the authors concluded that the approach is easy to use and most of the participants stated that the approach is beneficial, even if the required time for the initial development increased measurably.

6.5. Discussion

One of the major problems of end-user programs is that they are usually not rigorously tested. As demonstrated through various experimental studies, better tool support during the development process helps users to develop spreadsheets with fewer errors. However, commercial spreadsheet systems contain limited functionality in that direction. MS Excel only provides a very basic data validation tool for describing allowed types and values for individual cells, which can be seen as a form of assertions.

One problem in that context lies in the design of user interfaces for test tools that are suitable for end users. While in-depth evaluations of the effectiveness of the test case generation or test adequacy were performed as described above, the number of experiments regarding usability aspects with real users is still somewhat limited. Another main issue is the limited awareness of end users regarding the importance and value of thorough testing and their overconfidence in the correctness of the programs. More research about how to stimulate users to provide more information to the QA process in the sense of [80] is therefore required.

In that context, a better understanding is required in which ways spreadsheet developers actually test their spreadsheets or would be willing to at least partially adopt a test-driven development principle. In [83], Hermans made an analysis based on the EUSES corpus which revealed that there are a number of users who add additional assertions in the form of regular formulas to their spreadsheets. These assertions or tests are however often incomplete and have a low coverage, which led the author to the development of an add-on tool that automatically points the user to possible improvements for such user-specified assertions.

The application of *mutation testing* techniques to spreadsheet programs was discussed in [84]. Mutation testing consists of introducing small changes to a given program and checking how many of these mutants can be eliminated by a given set of tests. In their work, Abraham and Erwig propose a set of mutation operators for spreadsheets where some of them are based on operators that are used for mutating general-purpose languages and some of them are spreadsheet-specific. Generally, mutations can be used to test the coverage or adequacy of manually or automatically created test suites. In the broader context of fault detection and removal, they can however also be used to evaluate debugging approaches as was done in the spreadsheet literature, e.g., in [85, 86, 87] or [88].

7. Automated Fault Localization & Repair

The approaches in this category address scenarios in the development process, in which the spreadsheet developer enters some test data in the spreadsheet and observes unexpected calculation results in one or more cells. Such situations arise either during the initial development or when one of the above-mentioned test methodologies is applied. Already for medium-sized spreadsheets, the set of possible "candidates" that could be the root cause of the unexpected behavior can be large, in particular when the spreadsheet consists of longer chains of calculations that involve many of the spreadsheet's cells. Without tool support, the user would have to inspect all formulas on which the cell with the erroneous value depends and check them for correctness. The goal of most of the approaches in this category therefore is to assist the user in locating the true cause of the problem more efficiently, in many cases by ranking the possible error sources (*candidates*). Some of the approaches even go beyond that and try to compute a set of possible "repairs", i.e., changes to some of the formulas to achieve the desired outcomes. In contrast to static code analysis and inspection approaches, the basis for the required calculations usually comprises a specification of input values and expected output values or test cases.

7.1. Trace-based candidate ranking

An early method for candidate ranking which has some similarities to spectrum-based fault localization methods for imperative programs was presented by Reichwein et al. in [89] and [90]. In their method, they first propose to transfer the concept of program slicing to spreadsheets in order to eliminate impossible candidates in an initial step. Their technique uses user-specified information about correct and incorrect cell values and considers those cells that theoretically contribute to an erroneous cell value to be possibly faulty. A cell's formula is more likely to be faulty, if it contributes to more values that are marked as erroneous. Similarly, a formula is more likely to be correct if it contributes to more correct cell values. If a cell contributes to an incorrect cell value but the path to it is "blocked" by a cell with a correct value, its fault likelihood is assumed to be somewhere in between. In later works [91, 92], in which besides two further heuristics for fault localization a deeper analysis of the method's effectiveness factors were discussed, this technique is called "Blocking Technique". The "Blocking Technique" was evaluated in a user study involving 20 subjects in [90].

The task of the participants, which were split into two groups of equal size, was to test a given spreadsheet. Both groups were using a tool that implemented the WYSIWYT approach. One group additionally had the described fault localization extension activated. An interview after the experiment and the analysis of the experiment data showed – among other aspects – that most users appreciated the possibility to use the fault localization and they considered it to be particularly useful to locate the "harder" faults.

A similar technique was proposed by Ayalew and Mittermeir in [93], where for a faulty cell value the cells are highlighted that have the most influence on it. Later on, Hofer et al. in [87] explicitly proposed to adapt spectrum-based fault-localization from the traditional programming domain to spreadsheets. In contrast to previous works, they use a more formal approach with similarity coefficients to calculate the fault probabilities of the spreadsheet cells. They evaluated their version of spectrum-based fault localization for spreadsheets on a subset of the EUSES spreadsheet corpus and compared the fault localization capabilities of spectrum-based fault localization to those of two model-based debugging approaches (see Section 7.2).

7.2. Constraint-based fault localization

The following approaches translate a spreadsheet into a constraint-based representation, such that additional inferences about possible reasons for an unexpected value in some of the cells can be made.

In [94], Jannach and Engler presented an approach in which they first translated the spreadsheet into a Constraint Satisfaction Problem (CSP)[95]. Then, based on user-specified test cases and information about unexpected values in some of the cells, they used the principle of Model-Based Diagnosis (MBD) to determine which cells can theoretically be the true cause for the observed and unexpected calculation outcomes. With their work, they continue a line of research in which the MBD-principle, which was originally designed to find problems in hardware artifacts, is adapted for software debugging, see, e.g., [96] or [97]. Technically, an approach similar to [97] was adopted, which is capable of dealing with multiple "positive" and "negative" test cases and at the same time supports the idea of user-provided assertions.

In a first evaluation with relatively small artificial spreadsheets containing a few dozen formulas, it was shown that the approach is – depending on the provided test cases – able to significantly reduce the number of fault candidates. Later on, the method was further improved and optimized and embedded in the Exquisite debugging tool for MS Excel [98]. An evaluation of the enhanced version on similar examples showed significant enhancements with respect to the required calculation time. Mid-sized spreadsheets containing about 150 formulas and one injected fault could for example be diagnosed within 2 seconds on a standard laptop computer.

In a later work [88], different algorithmic improvements were proposed which helped to increase the scalability of the approach. The method was evaluated using a number of real-world spreadsheets in which faults where artificially injected. Furthermore, a small user study in the form of a debugging exercise was conducted, which indicated that the users working with the Exquisite tool were both more efficient and effective than the group that did a manual inspection. The size of the study was however quite small and involved only 24 participants.

A similar approach for finding an explanation for unexpected values using a CSP representation and the MBD-principle has been proposed by Abreu et al. in [99] and [100]. While the general idea is similar to the approach of Jannach and Engler, the technical realization is slightly different. Instead of using the Hitting-Set Algorithm [101], they encode the reasoning about the correctness of individual formulas directly into the constraint representation. Therefore, they make use of an auxiliary boolean variable for each cell representing the correctness of the cell's formula. Another difference of this approach compared to the work of Jannach and Engler is that Abreu et al. rely on a single test case only. The method was evaluated using four comparably small and artificial spreadsheets for which the algorithm could find a manually injected fault very quickly (taking at most 0.17 seconds). In [102], Außerlechner et al. evaluated this constraint-based approach using different SMT[7] and constraint solvers. For their evaluation, they created a special document corpus which both comprised spreadsheets that contain only integer calculations as well as a subset of the EUSES corpus with real number calculations. Their evaluation showed that the debugging approach of Abreu et al. can be used to find faults in medium-sized spreadsheets in real-time and that the approach is capable of debugging spreadsheets containing real numbers.

In [87], Hofer et al. propose to combine their spectrum-based fault localization approach with a light-weight Model-Based Software Debugging tech-

[7]Satisfiability Modulo Theories

nique. In particular, Hofer et al. suggest to use the coefficients obtained from the SFL technique as initial probabilities for the model-based debugging procedure. To evaluate the effectiveness of their hybrid method, they compared their approach to a pure spectrum-based approach and a constraint-based diagnosis approach. In their experiments, spreadsheets from the EUSES spreadsheet corpus were mutated using a subset of the mutation operators proposed in [84]. Overall, 227 mutated spreadsheets containing from 6 to over 4,000 formulas were used in the comparison. The results showed that the combined approach led to a better ranking of the potentially faulty cells, but was slightly slower than the pure SFL method.

7.3. Repair approaches

Repair-based approaches do not only point the users to potentially problematic formulas, but also aim to additionally propose possible corrections to the given formulas in a way that unexpected values in cells can be changed to the expected ones.

A first method for automatically determining such change suggestions ("goal-directed debugging") was presented by Abraham and Erwig in [85]. In their approach, the user states the expected value for an erroneous cell and the method computes suitable change suggestions by recursively changing individual formulas and propagating the change back to preceding formulas using spreadsheet-specific change inference rules. The possible changes that yield the desired results are then ranked based on heuristics. A revised and improved version of the method ("GoalDebug") that is better suited to address different (artificial) spreadsheet fault types discussed in [84], was presented in [103]. Later on, GoalDebug was combined with the AutoTest approach (see Section 6) to further improve the debugging results with the help of more test cases and other testing-related information [104].

To test the usefulness of their initial proposal [85], a user study with 51 subjects inspecting 2 spreadsheets with seeded faults was conducted. During the study, the subjects had to locate the faults using the WYSIWYT approach (see Section 6.1) but without the goal-directed method. The experiment revealed that the users made many mistakes when testing the spreadsheets and that the proposed approach could have prevented these mistakes. Furthermore, all the seeded faults were located with their approach.

GoalDebug was evaluated later on using an offline experiment where faults were injected into spreadsheets using a set of defined mutation operators. For the experiment, 7500 variants of 15 different spreadsheets with up to 54 formulas and 100 cells [84] were created and analyzed. The baseline for their evaluation was their own previous version of the method. The evaluation showed that GoalDebug was able to deal with all 9 defined mutation types and had a "success rate" of finding a correct repair of above 90 %, which was much better than the original version of the method.

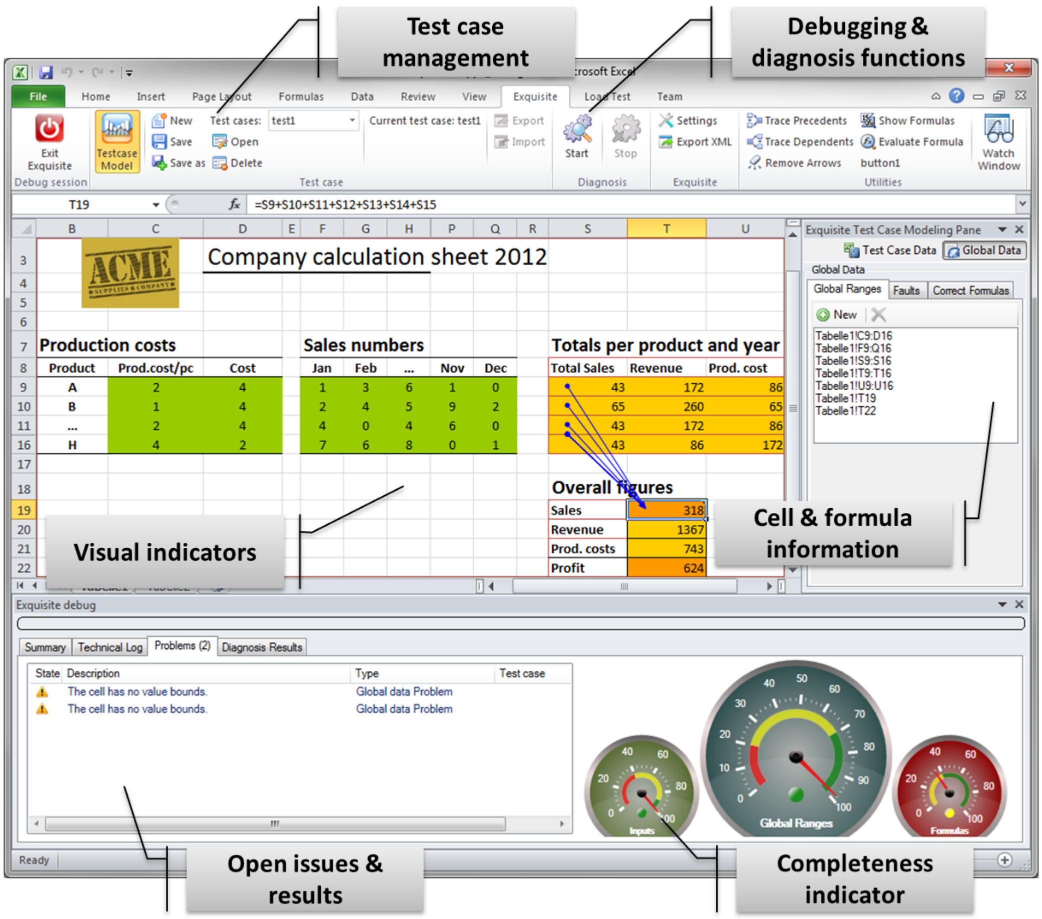
7.4. Discussion

The effectiveness of the debugging techniques reviewed in this section was mostly assessed using evaluation protocols in which certain types of faults were artificially seeded into given spreadsheets. The evaluations showed that the proposed techniques either lead to good rankings (of candidates or repair suggestions) or are able to compute a set of possible explanations. However, the spreadsheets used in the experiments often were small and the scalability of many approaches remains unclear. Besides, most of the approaches were evaluated with a non-public set of spreadsheets. Therefore, a direct comparison of the approaches is difficult. In addition, the constraint-based approaches are often limited to small spreadsheets and integer calculations.

Unfortunately, "oracle faults" are usually not discussed in the described approaches: For all approaches, the spreadsheet developer has to provide some information, e.g., the expected outcomes or which cells produce a correct output and which cells are erroneous. Most of the approaches are evaluated assuming a perfect user knowing every expected value. However, the spreadsheet developer often does not know all the required information or might accidentally provide wrong values. Further empirical evaluations should therefore consider vague or partly wrong user input.

For some of the proposed techniques, plug-in components for MS Excel have been developed, including [103] and [98], see Figure 3. Usability aspects of such tools have however not been systematically explored so far and it is unclear if they are suitable for an average or at least ambitious spreadsheet developer. More research in the sense of [105], where Parnin and Orso evaluated how and to which extent developers actually use debugging tools for imperative languages, is thus required in the spreadsheet domain.

Debugging support in commercial spreadsheet systems is very limited. Within MS Excel, one of the few features that support the user in the debugging process is the "Watch Window" as shown in Figure 4. Similar to debuggers for imperative programs, the spreadsheet developer can define watchpoints – in this case by selecting certain cells – and the current values of the cells are constantly updated and displayed in a compact form.



T19 =S9+S10+S11+S12+S13+S14+S15

Company calculation sheet 2012

Production costs

Product	Prod.cost/pc	Cost
A	2	4
B	1	4
...	2	4
H	4	2

Sales numbers

Jan	Feb	...	Nov	Dec
1	3	6	1	0
2	4	5	9	2
4	0	4	6	0
7	6	8	0	1

Totals per product and year

Total Sales	Revenue	Prod. cost
43	172	86
65	260	65
43	172	86
43	86	172

Overall figures

Sales	318
Revenue	1367
Prod. costs	743
Profit	624

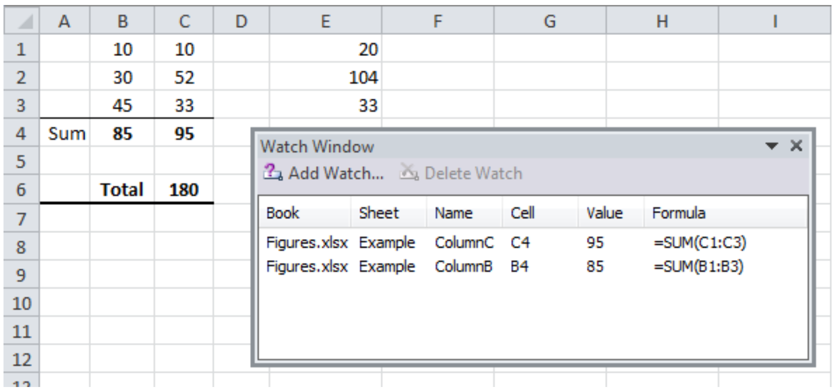
Tabelle1!C9:D16
Tabelle1!F9:Q16
Tabelle1!S9:S16
Tabelle1!T9:T16
Tabelle1!U9:U16
Tabelle1!T19
Tabelle1!T22

State	Description	Type	Test case
⚠	The cell has no value bounds.	Global data Problem	
⚠	The cell has no value bounds.	Global data Problem	

Figure 3: Debugging workbench of the Exquisite system [98]

8. Model-driven development approaches

In contrast to the approaches described in previous chapters, model-driven development approaches were not primarily designed to support the user in finding potential errors, but rather to improve the quality and structure of the spreadsheets and to prevent errors in the first place. Similar to model-driven approaches in the area of general software development, the main idea of these approaches is to introduce another layer of abstraction in the development process. Typically, the spreadsheet models in this intermediate layer introduce more abstract conceptualizations of the problem and thus serve as a bridge between the implicit idea, which the developer had of



	A	B	C	D	E	F	G	H	I
1		10	10		20				
2		30	52		104				
3		45	33		33				
4	Sum	**85**	**95**						
5									
6		**Total**	**180**						
7									
8									
9									
10									
11									
12									

Book	Sheet	Name	Cell	Value	Formula
Figures.xlsx	Example	ColumnC	C4	95	=SUM(C1:C3)
Figures.xlsx	Example	ColumnB	B4	85	=SUM(B1:B3)

Figure 4: MS Excel's watch window for cell value inspection

the spreadsheet, and the actual implementation. This way, the semantic gap between the intended idea and the spreadsheet implementation, which can become large in today's business spreadsheets [106], can be narrowed.

The abstract spreadsheet models proposed recently in the literature are used in two different phases in the development process. First, they are used as a form of "code-generators". In this scenario, parts of the spreadsheets are automatically generated from the models, thus reducing the risk of mechanical errors. Second, they are used to recover the underlying conceptual structures from an existing spreadsheet, which is similar to existing reverse engineering approaches in general software development. Following our classification scheme from Section 3, model-driven development approaches are therefore usually related to design and maintenance approaches, which we will discuss later on.

8.1. Declarative and object-oriented spreadsheet models

Isakowitz et al. were among the first to look at spreadsheet programs from a modeling perspective [38]. In their work, their main premise is that spreadsheet programs can be viewed at from a physical and a logical viewpoint, the physical being the cell's formulas and values and the logical being a set of functional relations describing the spreadsheets functionality. In their approach, spreadsheets consist of four principal components, among them the "schema" which captures the program's logic and the "data" which holds the values of the input cells. With the help of tools, this logic can be automatically extracted from a given spreadsheet and represented in a tool-independent language. In addition, the proposed system is capable of synthesizing spreadsheets from such specifications.

A similar object-oriented conceptualization of spreadsheet programs was presented later on by Paine in [107] and [108]. In the *Model Master* approach, spreadsheets are specified in a declarative way as text programs. These programs can then be passed to a compiler, which generates spreadsheets from these specifications. The logic of a spreadsheet is organized in the sense of object-oriented programming in the form of classes which encapsulate attributes and the calculation logic, see Figure 5. The comparably simple modeling language comprises a number of features including inheritance or multi-dimensional arrays to support tabular calculations.

```
company = attributes <
  incomings [ 1995:2004 ]
  outgoings [ 1995:2004 ]
  profit    [ 1995:2004 ]
>
where
  profit[ all t ] = incomings[ t ] - outgoings[ t ]
```

Figure 5: A class specification in *Model Master* [107].

Beside the automatic generation of spreadsheets from these models, the system also supports the extraction (reconstruction) of models from spreadsheets, which however requires the user to provide additional hints. The extracted models can be checked for errors or used as a standard for spreadsheet interchange. Particular aspects of structure discovery are discussed in [109][8].

To validate the general feasibility of the approach, different experiments were made in which small-sized spreadsheets were generated. The test cases used for model reconstruction were even smaller. Unfortunately and similar to the earlier work of Isakowitz et al. [38], no studies with real users were performed so far to assess the general usability of the approach at least for

[8]The work of Lentini et al. [39] is also based on the automatic extraction of the mathematical model of a given spreadsheet and a Prolog-based representation. However, their work rather focuses at the generation of a tutoring facility for a given spreadsheet and is thus only marginally relevant for our review.

advanced spreadsheet developers.

Paine described a different approach for a declarative modeling language in [110]. *Excelsior* is a spreadsheet development system which comprises a programming language built on *Prolog* and which is designed for the modular and re-usable specification of Excel spreadsheets. In addition to the standard functionality of *Prolog*, the programming language comprises specific constructs and operators to model the logic of a spreadsheet in a modular form. An example for such a specification is given in Figure 6. Based on such a design, the layout of the spreadsheet can be separated from its functionality and a compiler can be used to automatically generate a spreadsheet instance from these specifications.

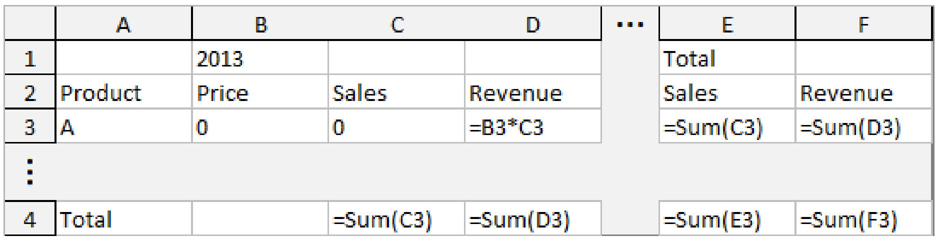
```
Year[2000] = 2000
Year[2001] = 2001
Sales[2000] = 971
Sales[2001] = 1803
Expenses[2000] = 1492
Expenses[2001] = 1560
Profit[2000] = Sales[2000] - Expenses[2000]
Profit[2001] = Sales[2001] - Expenses[2001]
Layout Year[2000:] as A2 downwards
Layout Expenses[2000:] as B2 downwards
Layout Sales[2000:] as C2 downwards
Layout Profit[2000:] as D2 downwards
```

Figure 6: A spreadsheet specification in *Excelsior* [110].

In [111], the functionality of the *Excelsior* system was tested on a larger spreadsheet. The task was to extract a model, i.e., the logical structure, of a spreadsheet with 10,000 cells and then apply several changes to it with the help of *Excelsior*. After model extraction, refactoring was found to be very easy in *Excelsior*, as only parameters had to be changed to generate a refactored and adapted spreadsheet. However, the extraction of the model was only semi-automatic and according to the authors took 2 days to complete. Moreover, no systematic evaluation to test the usability of this approach for average spreadsheet users was done.

8.2. Spreadsheet templates

In contrast to the works of Paine and colleagues, Erwig et al. proposed to rely on a *visual* and template-based method to capture certain aspects of the underlying model of a spreadsheet [112, 113, 114]. A "template" in their *Gencel* approach can in particular be used to specify repetitive areas in a spreadsheet. Figure 7 shows an example of a template specification. The design of the template can be done using a visualization that is similar to the typical UI paradigm of spreadsheet systems like MS Excel. In the example, the contents below the column headers B,C, and D are marked as being repetitive. In the model, this is indicated by the missing vertical separator lines between the column headers and the "..."-symbols between column and row headers.



	A	B	C	D	⋯	E	F
1		2013				Total	
2	Product	Price	Sales	Revenue		Sales	Revenue
3	A	0	0	=B3*C3		=Sum(C3)	=Sum(D3)
⋮							
4	Total		=Sum(C3)	=Sum(D3)		=Sum(E3)	=Sum(F3)

Figure 7: Spreadsheet template example; adapted from [114].

Similar to Paine's work, spreadsheet instances can be automatically generated from models. The generated spreadsheets can furthermore be altered later on in predefined ways. The supported operations include the addition or removal of groups of repetitive areas and value updates. Another feature of their approach is the use of a type system. The template-based approach also supports a reverse engineering process and the automatic reconstruction of templates from a given spreadsheet using certain heuristics [115]. To evaluate their approach, the authors discussed it in terms of the "Cognitive Dimensions of Notations" framework [116, 117] and conducted a small think-aloud study with 4 subjects [112]. Unfortunately, two of the subjects could not complete the spreadsheet development exercise because of technical difficulties; the spreadsheet created by the other participants were however error-free.

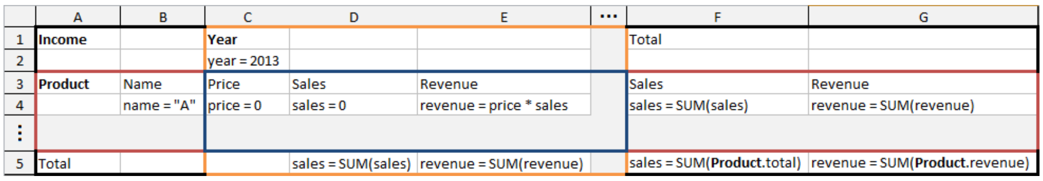
The template extraction method was evaluated in [115] with the help of a user study and a sample of 29 randomly selected spreadsheets of the EUSES spreadsheet corpus. The 23 participating users – 19 novice and 4 expert users – were asked to manually create templates for the selected spreadsheets. These manually created templates were then compared with the automatically extracted ones with respect to their correctness. The analysis revealed that the automatically generated templates were of significantly higher quality than the manually created ones and that even expert users had problems to correctly identify the underlying patterns of the spreadsheets.

8.3. Object-oriented visual models

As a continuation and extension to the template-based approach and in order to address a wider range of error types, Engels and Erwig later on proposed the concept of “ClassSheets” [118], which is similar to the work of Paine [107] mentioned above in the sense that the paradigm of object-orientation is applied to the spreadsheet domain.

Figure 8 shows an example of a ClassSheet specification, which uses a visualization similar to MS Excel. The different classes are visually separated by colored rectangles and represent semantically related cells. In contrast to the pure templates, the classes are not only syntactic structures but rather represent real-world objects or business objects in the sense of object-oriented software development. Beside the visual notation, the modeling approach comprises mechanisms to address the modeled objects rather through symbolic class names than through direct cell references.

Similar to the template-based approach described above, prototype tools were developed that support both the automated generation of spreadsheets from the models and the extraction of ClassSheet models from existing spreadsheets [119].



	A	B	C	D	E	...	F	G
1	**Income**		**Year**				Total	
2			year = 2013					
3	**Product**	Name	Price	Sales	Revenue		Sales	Revenue
4		name = "A"	price = 0	sales = 0	revenue = price * sales		sales = SUM(sales)	revenue = SUM(revenue)
⋮								
5	Total			sales = SUM(sales)	revenue = SUM(revenue)		sales = SUM(**Product**.total)	revenue = SUM(**Product**.revenue)

Figure 8: ClassSheet example; adapted from [118].

In the original paper in which ClassSheets were proposed and formalized [118], no detailed evaluation of the approach was performed. The automated extraction approach proposed in [119] was evaluated using a set of 27 spreadsheets, which contained 121 worksheets and 176 manually identified tables. According to their analysis, their tool was able to extract models from all but

13 tables. The 163 extracted models were then manually inspected. Only 12 of the models were categorized as being "bad‘" and 27 as being "acceptable". The remaining 124 models were found to be "good".

A number of extensions to the basic ClassSheet approach were later on proposed in the literature. In [106], Luckey et al. addressed the problem of model evolution and how such updates can be automatically transferred to already generated spreadsheets to better support a round-trip engineering process. The same problem of model evolution and the co-evolution of the model and the spreadsheet instances was addressed by Cunha et al. in [120, 121]. In [122], Cunha et al. proposed an approach to support the other update direction – the automatic transfer of changes made in the spreadsheet instances back to the spreadsheet model. Further extensions to the ClassSheet approach comprise the support of primary and foreign keys as used in relational designs, the generation of UML diagrams from ClassSheet models to support model validation or mechanisms to express constraints on allowed values for individual cells [123, 124]. For most of these extensions, no systematic evaluation has been done so far.

A different, in some sense visual approach to re-construct the underlying (object-oriented) model was proposed by Hermans et al. in [125]. Their approach is based on a library of typical patterns, which they try to locate in spreadsheets with the help of a two-dimensional parsing and pattern matching algorithm. The resulting patterns are then transformed into UML class diagrams, which can be used to better understand or improve a given spreadsheet. For the evaluation of their prototype tool, they first checked the plausibility of their patterns by measuring how often they appear in the EUSES corpus. Then, for a sample of 50 random spreadsheets, they compared the quality of generated class models with manually created ones, which led to promising results.

8.4. Relational spreadsheet models

One of the main principles of most spreadsheets is that the data is organized in tabular form. An obvious form of trying to obtain a more abstract model of the structure of a spreadsheet is to rely on approaches and principles from the design of relational databases. With the goal of ending up in higher-quality and error-free spreadsheets, Cunha et al. in [126] proposed to extract a relational database schema from the spreadsheet, which shall help the user to better understand the spreadsheet and which can consequently be used to improve the design of the spreadsheet. The main outcome of such a refactoring process should be a spreadsheet design which is more modular, has no data redundancies and provides suitable means to prevent wrong data inputs. With respect to the last aspect, Cunha et al. in [127, 128] proposed to use the underlying (extracted) relational schema to provide the user with advanced editing features including the auto-completion of values, non-editable cells and the safe deletion of rows.

In the original proposal of Cunha et al. [126], no formal evaluation of the approach was performed. An evaluation of the model-based approaches proposed in [127] and [126] was however done later on in [129]. In this user study, the goal was to assess if relying on the proposed methods can actually help to increase the effectiveness and efficiency of the spreadsheet development process. The participants of the study had to complete different development tasks and these tasks had to be done either on the original spreadsheet designs or on one of the assumedly improved ones. The results of the experiment unfortunately remained partially inconclusive and the results were not consistently better when relying on the model-based approaches.

To evaluate the advanced editing features mentioned in [128], a preliminary experiment using a subset of the EUSES spreadsheet corpus was done in that work. The initial results indicate that the tool is suited to provide helpful editing assistance for a number of spreadsheets; a more detailed study about potential productivity improvements and error rates has so far not been done.

8.5. Discussion

The model-driven development approaches discussed in this section aim to introduce additional syntactic or semantic abstraction layers into the spreadsheet development process. Overall, these additional mechanisms and conceptualizations shall help to close the semantic gap between the final spreadsheet and the actual problem in the real world, lead to higher quality levels in terms of better designs and fewer errors, and allow easier maintenance. Going beyond many model-driven approaches for standard software artifacts, automated "code" generation and support for round-trip engineering are particularly in the focus of spreadsheet researchers.

However, following a model-based approach comes with a number of challenges, which can also be found in standard software development processes. These challenges for example include the problem of the co-evolution of models and programs. Furthermore, the design of the modeling language plays an important role and often a compromise between expressivity and comprehensibility has to be made. A particular problem in that context certainly lies in the fact that the spreadsheet designers usually have no formal IT education and might have problems understanding the tools or the long-term advantages of better abstractions and structures. Furthermore, one of the main reasons of the popularity of spreadsheets lies in the fact that no structured or formal development process is required and people are used to develop spreadsheets in an ad-hoc, interactive and incremental prototyping process.

From a research perspective, many of the discussed papers only contain a preliminary evaluation or no evaluation at all. Thus, a more systematic evaluation and more user studies are required to obtain a better understanding if the proposed models are suited for typical spreadsheet developers and if they actually help them to develop spreadsheets of higher quality.

In current spreadsheet environments like MS Excel, only very limited support is provided to visually or semantically enrich the data or the calculations. One of the few features of adding semantics in a light-weight form is the assignment of symbolic names to individual cells or areas, which increases the readability of formulas. In addition, MS Excel provides some features for data organization including the option to group data cells and hide and display them as a block.

9. Design and maintenance support

The following approaches support the user in the development and maintenance processes. These approaches range from tools whose goal is to avoid wrong references over the handling of exceptional behavior to tools supporting the long-term use of spreadsheets (e.g., change-monitoring tools, add-ins for automatic refactoring and approaches that handle the reuse of formulas). All these tools play an important role in spreadsheet quality assurance as their goal is to avoid faults either by means of a clear and simple representation, by automation or by dealing with certain types of exceptional behavior.

9.1. Reference management

A major drawback of common commercial spreadsheet tools is that they provide limited support to ensure the correctness of cell references across the spreadsheet, e.g., because names of referenced cells do not carry semantic information about the content. Users often reference the wrong cells because they make off-by-one mistakes when selecting the referenced cell or accidentally use relative references instead of absolute references. Identifying such wrong references can be a demanding task. Even though systems like MS Excel support named cells and areas, most spreadsheet developers use the numbered and thus abstract cell names consisting of the row and column index.

Early approaches to address this problem – including NOPumpG [130, 131] and Action Graphics [132] – propose to give up the grid-based paradigm of spreadsheets and force the user to assign explicit names to the "cells". WYSIWYC ("What you see is what you compute") [133] is an alternative approach which retains the grid-based paradigm and proposes a new visual language for spreadsheets. The approach shall help to make the spreadsheet structures, calculations and references better visible and thus lead to a better correspondence of a spreadsheet's visual and logical structure. This should help to avoid errors caused by wrong cell references.

Unfortunately, while prototype systems have been developed, none of the above mentioned techniques have been systematically evaluated, e.g., through user studies. Therefore, it remains unclear if end users would be able to deal with such alternative development approaches and to which extent the problem of wrong cell references would actually be solved.

Finally, note that some problems of wrong cell references can be guaranteed to be avoided when (parts of) the spreadsheets are automatically generated from templates or visual models as done in the Gencel [112] and ClassSheet [118] approaches, see Section 8.2 and 8.3. In these systems, certain types of errors including reference errors can be avoided as only defined and correct update operations are allowed.

9.2. Exception Handling

The term exception handling refers to a collection of mechanisms supporting the detection, signaling and after-the-fact handling of exceptions [134]. Exceptions are defined as any unusual event that may require special processing [135]. Being aware of possible exceptional situations and handling them accordingly is an important factor to improve the quality of spreadsheets and making them more robust.

In [134], Burnett et al. propose such an approach to exception handling for spreadsheets. In their paper, they show that the *error value model* can be used for easy and adequate exception handling in spreadsheets. In the error value model, error messages (like `#DIV/0` in MS Excel) are returned instead of the expected values. The advantage of the approach using error values is that no changes to the general evaluation model in the spreadsheet paradigm are necessary. Exception handling approaches for imperative paradigms, in contrast, usually alter the execution sequence, which is not the case for spreadsheets with their static evaluation order. In addition, no special skills are required by the spreadsheet developers for exception prevention and exception handling as they can use the standard language operators (e.g., the if-then-else construct). What makes the approach of Burnett et al. different from the typical error value model in systems like MS Excel is that it supports customizable error types the end user can define to handle *application-specific* errors. Burnett et al. implemented their exception handling approach in the research system Forms/3. However, no evaluation with real users was done.

9.3. Changes and spreadsheet evolution

Spreadsheets often undergo changes and, unfortunately, changes often come with new errors that are introduced. FormulaDataSleuth [136] is a tool aimed to help the spreadsheet developer to immediately detect such errors when the spreadsheet is changed. Once the developer has specified which data areas and cells should be monitored by the tool, the system will automatically detect a number of potential problems. For the defined data areas, the tool can for example detect empty cells or input values that have a wrong data type or exceed the predefined range of allowed values. For monitored formula cells, accidentally overwritten formulas as well as range changes leading to wrong references can be identified. The authors demonstrate the usefulness of their approach by means of a running example. A deeper experimental investigation is however missing.

Understanding how a given spreadsheet evolved over time and seeing the difference between versions of a spreadsheet is often important when a spreadsheet is reused in a different project. In [137], Chambers et al. propose the *SheetDiff* algorithm, which is capable of detecting and visualizing certain types of non-trivial differences between two versions of a spreadsheet. To evaluate the approach, a number of spreadsheets from the EUSES corpus were selected. Some of them were considered to be modified versions of each other. For a number of additional spreadsheets, pre-defined change types (e.g., row insertion) were applied. The proposed algorithm was then compared with two commercial products. As measures, the correct change detection rate and the compactness of the result presentation were used. The results indicate that the new method is advantageous when compared with existing tools.

Later on, Harutyunyan et al. in [138] proposed a dynamic-programming based algorithm for difference detection called *RowColAlign*, which addressed existing problems of the greedy *SheetDiff* procedure described above. Instead of relying on manually selected or modified spreadsheets, a parameterizable test case generation technique was chosen, which allowed the authors to evaluate their method in a more systematic way.

9.4. Refactoring

Refactoring is defined as the process of changing the internal structure of a program without changing the functionality [139]. Refactoring contributes to the quality of spreadsheets in different ways, for example, by simplifying formulas and thus making them easier to understand, and by removing duplicate code thereby supporting easier and less error-prone maintenance. Refactoring in the context of spreadsheets is often concerned with the rearrangement of the columns and rows, i.e., the transformation of the design of the spreadsheet. Doing this transformation manually can be both time-intensive and prone to errors. Accordingly, different proposals have been made in the literature to automate this quality-improving maintenance task and to thereby prevent the introduction of new errors.

Badame and Dig [140] identify seven refactoring measures for spreadsheets and provide a corresponding plug-in for Microsoft Excel called RefBook. The plugin automatically detects the locations for which refactoring is required and supports the user in the refactoring process. Examples for possible refactoring steps include “Make Cell Constant”, “Guard Cell”, or “Replace Awkward Formula”. Badame and Dig evaluated their approach in different ways. In a survey involving 28 Excel users, the users preferred the refactored formula versions. In addition, a controlled lab experiment showed that people introduce faults during manual refactoring which could be avoided through automation. A retrospective analysis of spreadsheets from the EUSES corpus was finally done to validate the applicability of refactoring operators for real-world spreadsheets.

Harris and Gulwani [141] present an approach that supports complicated table transformations using user-specified examples. Their approach is based on a language for describing table transformations, called *TableProg*, and the algorithm *ProgFromEx* that takes as input a small example of the current spreadsheet and desired output spreadsheet. *ProgFromEx* automatically infers a program that implements the desired transformation. In an empirical evaluation, Harris and Gulwani applied their algorithm to 51 pairs of spreadsheet examples taken from online help forums for spreadsheets. This empirical evaluation proved that the required transformation programs could be generated for all example spreadsheets. However, sometimes a more detailed example spreadsheet than the one provided by the users was necessary.

In principle, the *Excelsior* tool mentioned in Section 8 is also suited to support spreadsheet restructuring tasks [111]. *Excelsior* supports flip and resize operations for tables. In addition, users can create several variants of a given spreadsheet. In [111], a case study was performed using one spreadsheet with several thousand cells to show the general feasibility of the approach. The depth of the evaluation thus considerably differs from the other refactoring approaches discussed in this section, which rely both on user studies and analyses based on real-world spreadsheets.

9.5. Reuse

In general, reusing existing and already validated software artefacts saves time, avoids the risk of making faults and supports maintainability [142]. This obviously also applies for spreadsheet development projects. Individual spreadsheets or parts of them are often reused in other projects. At a micro-level, even individual formulas are often used several times within a single spreadsheet. The standard solution for the reuse of formulas is to simply copy and paste the formulas. However, changing the original formula does not change its copies and forgotten updates of copied formulas thus can easily lead to faults.

The problem of reuse within spreadsheet programs was addressed by Djang and Burnett [143] and by Montigel [144]. In the approach of Djang and Burnett [143], reuse is mainly achieved through the concept of inheritance, a reuse approach that is common in object-oriented programming. Their "similarity inheritance" approach is however specifically designed to match the spreadsheet paradigm. In principle, it allows the developer to specify dependencies between (copied) spreadsheet cells in the form of multiple and mutual inheritance both on the level of individual cells and on a more coarse-grained level. The approach is illustrated based on a number of examples; an empirical evaluation is mentioned as an important next step.

Montigel [144] proposes the spreadsheet language *Wizcell.* In particular, *Wizcell* aims at facilitating reuse by making the possible semantics of copy

& paste and drag & drop actions more explicit. In particular, he sees four possible outcomes of such actions: (1) Either the copied formula is duplicated or there is a reference to the original formula. (2) Either the formulas in the copied cells refer to the cells mentioned in the original cells, or the references are changed according to the relative distance of the copy and the original. The proposed *Wizcell* language correspondingly allows the developer to specify the intended semantics, thus reducing the probability of introducing a fault. Similar to the reuse approach presented in [143], no report on an empirical evaluation is provided in [144].

9.6. Discussion

Many of the techniques and approaches presented in this section adapt existing techniques from traditional Software Engineering to the spreadsheet domain. In some cases, the authors explicitly address the problem that the basic spreadsheet development paradigm should not be changed too much and that the comprehensibility for the end user has to be maintained. However, some approaches require that the developer has a certain understanding of non-trivial programming concepts. As end users are usually non-professional programmers, the question of the applicability in practice arises.

Excelsior [111], for example, requires the user to understand concepts from logic programming. Djang and Burnett [143] build their work upon the concept of inheritance. While this term might not be used in the tool and these details are hidden from the UI through a visual representation, understanding the underlying semantics might be important for the developer to use the tools properly. NOPumpG [130, 131] and Action Graphics [132] use the concept of variables, which might not be known to a spreadsheet user. It therefore remains partially open whether all of these approaches are suited for end users without programming experience even if comparably simple visual representations are used.

The exception handling approach of Burnett et al. [134] requires no extended programming skills (except for example simple if-then-else constructs). Also Harris and Gulwani [141] consider the often limited capabilities of spreadsheet developers in their method and propose an example-based approach. Badame and Dig [140] rely on a semi-automatic approach and a plug-in to a wide-spread tool like MS Excel.

10. Discussion of current research practice

Our review showed that the way the different approaches from the literature are evaluated varies strongly. This can be partially attributed to the fact that research is carried out in different sub-fields of Computer Science as well as in Information Systems, each having their own standards and protocols.

The following major types of evaluation approaches can be found in the literature.

1. User studies: The proposed techniques and tools were evaluated in laboratory or field studies.
2. Empirical studies without users: The approaches were empirically evaluated, e.g., by applying them on operational spreadsheets or spreadsheets containing artificially injected errors. Such forms of evaluation can for example show that certain types of faults will be found when applying a given method, e.g., [88].
3. Theoretical analyses: Some researchers show by means of theoretical analyses that their approaches prevent certain types of errors, e.g., reference errors [114, 118].
4. No systematic evaluation: In some sub-areas and in particular for some older proposals, the evaluation was limited to an informal discussion based on example problems, based on unstructured feedback from a small group of users, or there was no real evaluation done at all.

Traditionally, research in various sub-fields of Computer Science is often based on offline experimental designs and simulations, whereas user studies are more common in Information Systems research, see, e.g., [145] for a review of evaluation approaches in the area of recommendation systems. In more recent proposals in particular from the Computer Science field, which is the focus of this work, theoretical or simulation-based analyses are now more often complemented with laboratory studies, e.g., in [60] or [88].

Generally, while we observe improvements with respect to research rigor and more systematic evaluations over the last years, in our view the research practice in the field can be further improved in different aspects.

10.1. Challenges of empirical evaluation approaches without users

The sample data sets used in offline experimental designs are often said to be (randomly) taken from the huge and very diverse EUSES corpus. Which documents were actually chosen and which additional criteria were applied is often not well justified. The choice can be influenced for example by the scalability of the proposed method or simply by the capabilities of some parser. Other factors that may influence the observed "success" of a new method can be the types or positions of the injected errors. These aspects are often not well documented and even when the benchmark problems are made publicly available as in [87], they may have special characteristics that are required or advantageous for a given method and, e.g., contain only one single fault or use only a restricted set of functions or cell data types.

We therefore argue that researchers should report in more detail about the basis of their evaluations. Otherwise, comparative evaluations are not easily possible in the field, in particular as source codes or the developed Excel plug-ins are usually not shared in the community. Even though different types of spreadsheets might be required for the different research proposals, one future goal could therefore be to develop a set of defined *benchmark spreadsheets*. These can be used and adapted by the research community and serve as a basis for comparative evaluations, which are barely found in current spreadsheet literature.

10.2. Challenges of doing user studies

The more recent works in the field often include reports on different types of laboratory studies to assess, for example, if users are actually capable of using a new tool or, more importantly, if the tool actually helps the users in the fault identification or removal process. Such studies can be considered to be the main evaluation instrument in IS research and the typical experimental designs of such studies include tasks like code inspection and fault localization, error detection and removal, and formula or spreadsheet construction.

Conducting reliable user studies, which are done usually in laboratory settings, is in general a challenging task even though various standard designs, procedures, and statistical analysis methods exist that are also common, e.g., in sociobehavioral sciences [146]. A discussion of general properties of valid experimental designs is beyond the scope of this work. However, in our review we observed some typical limitations in the context of spreadsheet research.

First, the number of participants in each "treatment group" – e.g., one group with and another group without tool assistance – is often quite small. Various ways including statistical power analysis exist to determine the minimum number of participants, which can however depend on the goal and type of the study, the statistical significance criterion used, or the desired confidence level and interval. Typical sample sizes in the literature are for example 61 participants assigned in two groups [147] or 90 participants that were distributed to two groups of different sizes [148].

Additional questions in that context are whether the study participants are representative for a larger population of spreadsheet users – in [149], students are considered as good surrogates – and how it can be made sure that the participants are correctly assigned to the different groups, e.g., based on their experience or a random procedure. Finally, the question arises if doing the experiment in a laboratory setting is not introducing a bias making the evaluation unrealistic. As for the latter aspect, also studies exist in which the participants accomplish the tasks at home [150]. In these cases, it is however easier for the participants to cheat. In particular for spreadsheet construction exercises, it has to be considered that the developed spreadsheets can be quite different from real operational spreadsheets, e.g., with respect to their complexity [11].

10.3. General remarks

In general, both for user studies and offline experiments in which we use artificially injected errors, the problem exists that we cannot be sure that the introduced types of faults are always representative or realistic. While a number of studies on error rates exist, Powell et al. [11] argue that it is often unclear which fault categorization scheme was used or how faults were counted that were corrected during the construction of the spreadsheet. It can thus be dangerous to make inferences about the general efficacy of a method if it was only evaluated on certain types of faults.

Field studies based on operational spreadsheets and real spreadsheet developers would obviously represent a valuable means to assess the true efficacy, e.g., of a certain fault reduction approach. Such reports are however rare as they are costly to conduct. The work presented in [148] is an example of such a study, in which experienced business managers participated and accomplished a spreadsheet construction exercise. In such settings, however, additional problems arise, e.g., that the participants could not be assigned to different treatment groups randomly as their geographical location had to be taken into account.

Finally, as in many other research fields, experimental studies are barely reproduced by other research groups to validate the findings. In addition, the reliability of the reported results can be low, e.g., because of biases by the researchers, weak experimental designs, or questions of the interpretation of the outcomes of statistical tests [151, 152].

Overall, the evaluation of tools and techniques to localize and remove faults in spreadsheets remains a challenging task as it not only involves algorithmic questions but at the same time has to be usable by people with a limited background in IT. In many cases, a comprehensive evaluation approach is therefore required which combines the necessary theoretical analysis with user studies whose design should incorporate the insights from the existing works, e.g., in the area of IS research or Human Computer Interaction.

11. Perspectives for future works

The literature review has pointed out some interesting new fields of research for spreadsheet quality assurance. In the following, we will sketch a subjective selection of possible directions to future works. In the discussion, we will limit ourselves to broader topics and not focus on specific research opportunities within the different sub-areas.

11.1. Life cycle coverage

Our review shows that a number of proposals have been made to support the developer in various stages of the spreadsheet life cycle including application design and development, testing, debugging, maintenance, comprehension and reuse. For the early development phases – like domain analysis, requirements specification and the initial design – we have however not found any proposals for automated tool support. Ko et al. in [153] argue that these early phases and tasks are mostly not explicitly executed in typical spreadsheet development activities, or more generally, end user programming scenarios. In their work, a detailed discussion and analysis of general differences between professional software engineering processes and end user software engineering can be found. Regarding requirements specification, Ko et al. for example mention that in contrast to professional software development, the source of the requirements is the same person as the programmer, e.g., because people often develop spreadsheets for themselves. With respect to design processes, one assumption is that end user programmers might not see the benefits of making the design an explicit step when translating the requirements into a program.

How to provide better tool support for the very early phases – which should ultimately lead to higher-quality spreadsheets in the end – is in our view largely open. Such approaches probably have to be accompanied with organizational measures and additional training for the end user programmer to raise the awareness of the advantages of a more structured development process, even if this process is exploratory and prototyping-based in nature. Alternative development approaches such as Example-Driven Modeling [154] or programming by example could be explored as well.

Beside tool support for the early development phases, we see a number of other areas where existing quality-ensuring or quality-improving techniques can be applied or further adapted to the spreadsheet domain. This includes better quality metrics, formal analysis methods, or techniques for spreadsheet evolution, versioning and "product lines", which in our view have not been explored deeply enough so far.

11.2. Combination of methods

We see a lot of potential for further research in the area of combining different specific techniques in hybrid systems. In [86], for example, the authors propose methods to combine the feedback of the UCheck type checking system with the results from the WYSIWYT fault localization technique based on heuristics. An evaluation using various mutations of a spreadsheet showed that the combination is advantageous, e.g., because different types of faults can be detected by the two techniques. Other works that integrate different types of information or reasoning strategies include [57, 54, 104] and more recently [87], who combine declarative debugging with trace-based candidate ranking.

Beside the integration of methods to fulfill one particular task, one possible direction of future research is to explore alternative hybridization designs, e.g., to combine methods in a sequential or parallel manner. In such a scenario, one computationally cheap method could be used to identify larger regions in the spreadsheet which most probably contain an error. More sophisticated and computationally demanding techniques could afterwards be applied within this local area to determine the exact location of the problem. Alternatively, there might be situations in which multiple techniques are available for a certain task, e.g., to rank the error candidates. Whether or not a specific technique works well for a given problem setting depends on a number of factors including the structure and the size of the spreadsheets or the types of the formulas. A possible future research direction could therefore lie in the development of algorithms which – based on heuristics, past observations, and a concise characterization of the capabilities and requirements of the different techniques – can automatically assess which of the available techniques will be the most promising given a specific spreadsheet and task.

11.3. Toward integrated user-friendly tools

Individual research efforts often aim at one particular problem, for example test support and test case management, propose one particular technique and focus on one single optimization criterion such as maximizing the test coverage. While keeping the work focused is appropriate in the context of individual scientific contributions, in reality, the different QA tasks are often related: a debugging activity, for example, can be initiated by a test activity or a maintenance task. Therefore, to be applicable in practice, one of the goals of future research is to better understand how integrated tools should be designed that support the developer in the best possible way. Such a research could for example include the discussion of suitable user interface (UI) designs, the choice of comprehensible terminology and metaphors, the question of the appropriate level of user guidance, the choice of adequate supporting visualizations, or even questions of how to integrate the tools smoothly into existing spreadsheet environments.

An example for such an end-user oriented interaction pattern for spreadsheets can be found in [80]. Using a so-called "surprise-explain-reward strategy", the goal of the work is to entice the user to make increased use of the assertion feature of the spreadsheet environment without requiring the user to change his or her usual work process. This is accomplished by automatically generating assertions about cell contents, presenting violations in the form of passive feedback, and then relying on the user's curiosity to explore the potential problems. Beckwith et al. later on continued this work in [155] and investigated gender-specific differences in the adoption of such new tool features and proposed different variations of the UI for risk-averse or low confidence users. Finally, another work that builds on psychological phenomena to increase tool adoption (and effectiveness) is presented in [156]. In this work, the authors focus on the role of the *perceivable rewards* and experiment with UI variants in which the tool's functionality is identical but the visual feedback, e.g., in terms of cell coloring is varied.

11.4. Toward a formal spreadsheet language

In the literature, a number of different *intermediate* representations are used to formally and precisely describe the logic of a given spreadsheet application. Some of these are based on standard formalisms with defined semantics including logic- and constraint-based approaches [110, 88, 99]; other papers introduce their own formalisms supporting a specific methodology or various forms of reasoning on it [85].

In particular in the latter cases, a precise definition of what can be expressed in these intermediate representations is sometimes missing, for example, if it is possible to reason about real-valued calculations or which of the more complex functions of systems like MS Excel can be expressed when using a certain intermediate representation.

In order to be able to better compare and combine different spreadsheet QA techniques in hybrid approaches as discussed above, a unified formal spreadsheet representation, problem definition language, or even a "theory of spreadsheets" could be useful. It would furthermore help making research efforts independent of specific environments or tool versions and at the same time allow for formal reasoning, e.g., about the soundness and completeness of individual fault localization techniques. Such problem definition languages are for example common in other domains such as Artificial Intelligence based planning or Constraint Satisfaction.

11.5. Provision of better abstraction mechanisms

In [157], Peyton Jones et al. argue that spreadsheets in their basic form can be considered as functional programs that only consist of statements comprising built-in functions. Thus, spreadsheet developers have no means to define reusable *abstractions* in the form of parameterizable functions. To implement the desired functionality, users therefore have to copy the formulas multiple times, which however leads to poor maintainability and lower spreadsheet quality. As a potential solution, the authors propose a user-oriented approach to design user-defined functions. A main goal of the design is to stay within the spreadsheet paradigm, which for example means that the function implementations should be specified as spreadsheets ("function sheets") and not in the form of imperative programs as done in MS Excel. The work presented in [157] was mostly based on theoretical considerations. In their evaluation, the authors mainly focus on the expressiveness of the language and performance aspects. So far, no evaluation investigating if users are able to understand the concepts of how to define function sheets or to interpret error messages has been done.

Later on, Sestoft [158] presented a practical realization of the approach that includes recursion and higher-order functions. To design and use new function sheets in their prototype system called “Funcalc”, the spreadsheet developer has to learn only three new built-in functions.

Both function sheets and the more recent ClassSheets as described in Section 8.3 represent approaches to empower spreadsheet developers with better abstraction mechanisms within the spreadsheet paradigm. As a result, these approaches should help users avoid making different types of faults and increase the general quality of the spreadsheets. Overall, we see the provision of such advanced concepts for spreadsheet design and implementation as a promising area for future research, where in particular the questions of understandability for the end user should be further investigated.

12. Summary

Errors in spreadsheet programs can have a huge impact on organizations. Unfortunately, current spreadsheet environments like MS Excel only include limited functionality to help developers create error-free spreadsheets or support them in the error detection and localization process. Over the last decades, researches in different subfields of Computer Science and Information Systems have therefore made a substantial number of proposals aimed at better tool support for spreadsheet developers during the development lifecycle.

With our literature review and the presented classification scheme we aimed to provide a basis to structure and relate the different strands of research in this area and critically reflected on current research practices. At the same time, the review and classification scheme should help to identify potential directions for future research and opportunities for combining different proposals, thereby helping to move from individual techniques and tools to integrated spreadsheet QA environments.

Acknowledgements

This work was partially supported by the European Union through the programme “Europäischer Fonds für regionale Entwicklung - Investition in unsere Zukunft” under contract number 300251802.

References

[1] R. R. Panko, D. N. Port, End User Computing: The Dark Matter (and Dark Energy) of Corporate IT, in: Proceedings of the 45th Hawaii

International Conference on System Sciences (HICSS 2012), Wailea, HI, USA, 2012, pp. 4603–4612.

[2] R. Creeth, Micro-Computer Spreadsheets: Their Uses and Abuses, Journal of Accountancy 159 (6) (1985) 90–93.

[3] S. Ditlea, Spreadsheets can be hazardous to your health, Personal Computing 11 (1) (1987) 60–69.

[4] R. R. Panko, What We Know About Spreadsheet Errors, Journal of End User Computing 10 (2) (1998) 15–21.

[5] T. Herndon, M. Ash, R. Pollin, Does High Public Debt Consistently Stifle Economic Growth? A Critique of Reinhart and Rogoff, Working Paper 322, Political Economy Research Institute, University of Massachusetts, Amherst (April 2013).

[6] C. M. Reinhart, K. S. Rogoff, Growth in a Time of Debt, American Economic Review 100 (2) (2010) 573–578.

[7] D. F. Galletta, D. Abraham, M. E. Louadi, W. Lekse, Y. A. Pollalis, J. L. Sampler, An empirical study of spreadsheet error-finding performance, Accounting, Management and Information Technologies 3 (2) (1993) 79–95.

[8] S. Thorne, A review of spreadsheet error reduction techniques, Communications of the Association for Information Systems 25, Article 24.

[9] T. Reinhardt, N. Pillay, Analysis of Spreadsheet Errors Made by Computer Literacy Students, in: Proceedings of the IEEE International Conference on Advanced Learning Technologies (ICALT 2004), Joensuu, Finland, 2004, pp. 852–853.

[10] D. F. Galletta, K. S. Hartzel, S. E. Johnson, J. L. Joseph, S. Rustagi, Spreadsheet Presentation and Error Detection: An Experimental Study, Journal of Management Information Systems 13 (3) (1996) 45–63.

[11] S. G. Powell, K. R. Baker, B. Lawson, A critical review of the literature on spreadsheet errors, Decision Support Systems 46 (1) (2008) 128–138.

[12] H. Howe, M. G. Simkin, Factors Affecting the Ability to Detect Spreadsheet Errors, Decision Sciences Journal of Innovative Education 4 (1) (2006) 101–122.

[13] J. R. Olson, E. Nilsen, Analysis of the Cognition Involved in Spreadsheet Software Interaction, Human-Computer Interaction 3 (4) (1987) 309–349.

[14] G. Rothermel, L. Li, C. Dupuis, 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 (ICSE 1998), Kyoto, Japan, 1998, pp. 198–207.

[15] R. R. Panko, R. P. Halverson, Spreadsheets on Trial: A Survey of Research on Spreadsheet Risks, in: Proceedings of the 29th Hawaii International Conference on System Sciences (HICSS 1996), Wailea, HI, USA, 1996, pp. 326–335.

[16] K. Rajalingham, D. R. Chadwick, B. Knight, Classification of Spreadsheet Errors, in: Proceedings of the European Spreadsheet Risks Interest Group 2nd Annual Conference (EuSpRIG 2001), Amsterdam, Netherlands, 2001.

[17] R. R. Panko, S. Aurigemma, Revising the Panko-Halverson taxonomy of spreadsheet errors, Decision Support Systems 49 (2) (2010) 235–244.

[18] M. Erwig, Software engineering for spreadsheets, IEEE Software 26 (5) (2009) 25–30.

[19] J. Davis, Tools for spreadsheet auditing, International Journal of Human-Computer Studies 45 (4) (1996) 429–442.

[20] J. Sajaniemi, Modeling Spreadsheet Audit: A Rigorous Approach to Automatic Visualization, Journal of Visual Languages & Computing 11 (1) (2000) 49–82.

[21] T. Igarashi, J. Mackinlay, B.-W. Chang, P. Zellweger, Fluid Visualization of Spreadsheet Structures, in: Proceedings of the IEEE Symposium on Visual Languages (VL 1998), Halifax, NS, Canada, 1998, pp. 118–125.

[22] H. Shiozawa, K. Okada, Y. Matsushita, 3D Interactive Visualization for Inter-Cell Dependencies of Spreadsheets, in: Proceedings of the IEEE Symposium on Information Visualization (Info Vis 1999), San Francisco, CA, USA, 1999, pp. 79–82, 148.

[23] Y. Chen, H. C. Chan, Visual Checking of Spreadsheets, in: Proceedings of the European Spreadsheet Risks Interest Group 1st Annual Conference (EuSpRIG 2000), London, United Kingdom, 2000.

[24] D. Ballinger, R. Biddle, J. Noble, Spreadsheet visualisation to improve end-user understanding, in: Proceedings of the Asia-Pacific Symposium on Information Visualisation - Volume 24 (APVIS 2003), Adelaide, Australia, 2003, pp. 99–109.

[25] K. Hodnigg, R. T. Mittermeir, Metrics-Based Spreadsheet Visualization: Support for Focused Maintenance, in: Proceedings of the European Spreadsheet Risks Interest Group 9th Annual Conference (EuSpRIG 2008), London, United Kingdom, 2008, pp. 79–94.

[26] B. Kankuzi, Y. Ayalew, An End-User Oriented Graph-Based Visualization for Spreadsheets, in: Proceedings of the 4th International Workshop on End-User Software Engineering (WEUSE 2008), Leipzig, Germany, 2008, pp. 86–90.

[27] Y. Ayalew, A Visualization-based Approach for Improving Spreadsheet Quality, in: Proceedings of the Warm Up Workshop for ACM/IEEE ICSE 2010 (WUP 2009), Cape Town, South Africa, 2009, pp. 13–16.

[28] F. Hermans, M. Pinzger, A. van Deursen, Supporting Professional Spreadsheet Users by Generating Leveled Dataflow Diagrams, in: Proceedings of the 33rd International Conference on Software Engineering (ICSE 2011), Waikiki, Honolulu, HI, USA, 2011, pp. 451–460.

[29] F. Hermans, M. Pinzger, A. van Deursen, Breviz: Visualizing Spreadsheets using Dataflow Diagrams, in: Proceedings of the European Spreadsheet Risks Interest Group 12th Annual Conference (EuSpRIG 2011), London, United Kingdom, 2011.

[30] B. Ronen, M. A. Palley, H. C. Lucas, Jr., Spreadsheet Analysis and Design, Communications of the ACM 32 (1) (1989) 84–93.

[31] R. Mittermeir, M. Clermont, Finding High-Level Structures in Spreadsheet Programs, in: Proceedings of the 9th Working Conference on Reverse Engineering (WCRE 2002), Richmond, VA, USA, 2002, pp. 221–232.

[32] S. Hipfl, Using Layout Information for Spreadsheet Visualization, in: Proceedings of the European Spreadsheet Risks Interest Group 5th Annual Conference (EuSpRIG 2004), Klagenfurt, Austria, 2004.

[33] M. Clermont, Analyzing Large Spreadsheet Programs, in: Proceedings of the 10th Working Conference on Reverse Engineering (WCRE 2003), Victoria, BC, Canada, 2003, pp. 306–315.

[34] M. Clermont, A Toolkit for Scalable Spreadsheet Visualization, in: Proceedings of the European Spreadsheet Risks Interest Group 5th Annual Conference (EuSpRIG 2004), Klagenfurt, Austria, 2008.

[35] M. Clermont, Heuristics for the Automatic Identification of Irregularities in Spreadsheets, in: Proceedings of the 1st Workshop on End-User Software Engineering (WEUSE 2005), St. Louis, MO, USA, 2005, pp. 1–6.

[36] M. Clermont, C. Hanin, R. T. Mittermeir, A Spreadsheet Auditing Tool Evaluated in an Industrial Context, in: Proceedings of the European Spreadsheet Risks Interest Group 3rd Annual Conference (EuSpRIG 2002), Cardiff, United Kingdom, 2002.

[37] D. Hendry, T. Green, CogMap: a Visual Description Language for Spreadsheets, Journal of Visual Languages & Computing 4 (1) (1993) 35–54.

[38] T. Isakowitz, S. Schocken, H. C. Lucas, Jr., Toward a Logical / Physical Theory of Spreadsheet Modeling, Transactions on Information Systems 13 (1) (1995) 1–37.

[39] M. Lentini, D. Nardi, A. Simonetta, Self-instructive spreadsheets: an environment for automatic knowledge acquisition and tutor generation, International Journal of Human-Computer Studies 52 (5) (2000) 775–803.

[40] D. Chadwick, B. Knight, K. Rajalingham, Quality Control in Spreadsheets: A Visual Approach using Color Codings to Reduce Errors in Formulae, Software Quality Control 9 (2) (2001) 133–143.

[41] D. Nardi, G. Serrecchia, Automatic Generation of Explanations for Spreadsheet Applications, in: Proceedings of the 10th Conference on Artificial Intelligence for Applications (CAIA 1994), San Antonio, TX, USA, 1994, pp. 268–274.

[42] R. Brath, M. Peters, Excel Visualizer: One Click WYSIWYG Spreadsheet Visualization, in: Proceedings of the 10th International Conference on Information Visualisation (IV 2006), London, United Kingdom, 2006, pp. 68–73.

[43] R. Rao, S. K. Card, The Table Lens: Merging Graphical and Symbolic Representations in an Interactive Focus+Context Visualization for Tabular Information, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI 1994), Boston, MA, USA, 1994, pp. 318–322.

[44] P. S. Brown, J. D. Gould, An Experimental Study of People Creating Spreadsheets, ACM Transactions on Information Systems 5 (3) (1987) 258–272.

[45] S. Aurigemma, R. R. Panko, The Detection of Human Spreadsheet Errors by Humans versus Inspection (Auditing) Software, in: Proceedings of the European Spreadsheet Risks Interest Group 11th Annual Conference (EuSpRIG 2010), London, United Kingdom, 2010.

[46] M. Erwig, M. M. Burnett, Adding Apples and Oranges, in: Proceedings of the 4th International Symposium on Practical Aspects of Declarative Languages (PADL 2002), Portland, OR, USA, 2002, pp. 173–191.

[47] M. Burnett, M. Erwig, Visually Customizing Inference Rules About Apples and Oranges, in: Proceedings of the IEEE Symposia on Human Centric Computing Languages and Environments (HCC 2002), Arlington, VA, USA, 2002, pp. 140–148.

[48] Y. Ahmad, T. Antoniu, S. Goldwater, S. Krishnamurthi, A Type System for Statically Detecting Spreadsheet Errors, in: Proceedings of the

18th IEEE/ACM International Conference on Automated Software Engineering (ASE 2003), Montreal, Canada, 2003, pp. 174–183.

[49] R. Abraham, M. Erwig, Header and Unit Inference for Spreadsheets Through Spatial Analyses, in: Proceedings of the IEEE Symposium on Visual Languages and Human Centric Computing (VL/HCC 2004), Rome, Italy, 2004, pp. 165–172.

[50] T. Antoniu, P. Steckler, S. Krishnamurthi, E. Neuwirth, M. Felleisen, Validating the Unit Correctness of Spreadsheet Programs, in: Proceedings of the 26th International Conference on Software Engineering (ICSE 2004), Edinburgh, United Kingdom, 2004, pp. 439–448.

[51] R. Abraham, M. Erwig, Type Inference for Spreadsheets, in: Proceedings of the 8th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming (PPDP 2006), Venice, Italy, 2006, pp. 73–84.

[52] R. Abraham, M. Erwig, UCheck: A Spreadsheet Type Checker for End Users, Journal of Visual Languages & Computing 18 (1) (2007) 71–95.

[53] C. Chambers, M. Erwig, Automatic Detection of Dimension Errors in Spreadsheets, Journal of Visual Languages & Computing 20 (4) (2009) 269–283.

[54] C. Chambers, M. Erwig, Reasoning About Spreadsheets with Labels and Dimensions, Journal of Visual Languages & Computing 21 (5) (2010) 249–262.

[55] M. J. Coblenz, A. J. Ko, B. A. Myers, Using objects of measurement to detect spreadsheet errors, in: Proceedings of the IEEE Symposium on Visual Languages and Human Centric Computing (VL/HCC 2005), 2005, pp. 314–316.

[56] C. Chambers, M. Erwig, Dimension Inference in Spreadsheets, in: Proceedings of the IEEE Symposium on Visual Languages and Human Centric Computing (VL/HCC 2008), Herrsching am Ammersee, Germany, 2008, pp. 123–130.

[57] C. Chambers, M. Erwig, Combining Spatial and Semantic Label Analysis, in: Proceedings of the IEEE Symposium on Visual Languages

and Human Centric Computing (VL/HCC 2009), Corvallis, OR, USA, 2009, pp. 225–232.

[58] M. Fisher, G. Rothermel, The EUSES Spreadsheet Corpus: A shared resource for supporting experimentation with spreadsheet dependability mechanisms, SIGSOFT Software Engineering Notes 30 (4) (2005) 1–5.

[59] M. Fowler, Refactoring: Improving the Design of Existing Code, Addison-Wesley Professional, 1999.

[60] F. Hermans, M. Pinzger, A. van Deursen, Detecting and Visualizing Inter-Worksheet Smells in Spreadsheets, in: Proceedings of the 34th International Conference on Software Engineering (ICSE 2012), Zurich, Switzerland, 2012, pp. 441–451.

[61] F. Hermans, M. Pinzger, A. van Deursen, Detecting Code Smells in Spreadsheet Formulas, in: Proceedings of the 28th IEEE International Conference on Software Maintenance (ICSM 2012), Riva del Garda, Trento, Italy, 2012, pp. 409–418.

[62] F. Hermans, B. Sedee, M. Pinzger, A. v. Deursen, Data Clone Detection and Visualization in Spreadsheets, in: Proceedings of the 35th International Conference on Software Engineering (ICSE 2013), San Francisco, CA, USA, 2013, pp. 292–301.

[63] J. Cunha, J. a. P. Fernandes, H. Ribeiro, J. a. Saraiva, Towards a Catalog of Spreadsheet Smells, in: Proceedings of the 12th International Conference on Computational Science and Its Applications (ICCSA 2012), Salvador de Bahia, Brazil, 2012, pp. 202–216.

[64] D. Nixon, M. O'Hara, Spreadsheet Auditing Software, in: Proceedings of the European Spreadsheet Risks Interest Group 2nd Annual Conference (EuSpRIG 2001), Amsterdam, Netherlands, 2001.

[65] J. Hunt, An approach for the automated risk assessment of structural differences between spreadsheets (diffxl), in: Proceedings of the European Spreadsheet Risks Interest Group 10th Annual Conference (EuSpRIG 2009), Paris, France, 2009.

[66] R. Abraham, M. Erwig, S. Andrew, A type system based on end-user vocabulary, in: Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2007), Coeur d'Alene, Idaho, USA, 2007, pp. 215–222.

[67] G. Rothermel, L. Li, M. Burnett, Testing Strategies for Form-Based Visual Programs, in: Proceedings of the 8th International Symposium on Software Reliability Engineering (ISSRE 1997), Albuquerque, NM, USA, 1997, pp. 96–107.

[68] G. Rothermel, M. Burnett, L. Li, C. Dupuis, A. Sheretov, A Methodology for Testing Spreadsheets, ACM Transactions on Software Engineering and Methodology 10 (1) (2001) 110–147.

[69] M. Burnett, A. Sheretov, G. Rothermel, Scaling Up a "What You See Is What You Test" Methodology to Spreadsheet Grids, in: Proceedings of the IEEE Symposium on Visual Languages (VL 1999), Tokyo, Japan, 1999, pp. 30–37.

[70] M. Burnett, A. Sheretov, B. Ren, G. Rothermel, Testing Homogeneous Spreadsheet Grids with the "What You See Is What You Test" Methodology, IEEE Transactions on Software Engineering 28 (6) (2002) 576–594.

[71] M. Burnett, B. Ren, A. Ko, C. Cook, G. Rothermel, Visually Testing Recursive Programs in Spreadsheet Languages, in: Proceedings of the IEEE Symposia on Human-Centric Computing Languages and Environments (HCC 2001), Stresa, Italy, 2001, pp. 288–295.

[72] M. Fisher, II, D. Jin, G. Rothermel, M. Burnett, Test Reuse in the Spreadsheet paradigm, in: Proceedings of the 14th International Symposium on Software Reliability Engineering (ISSRE 2003), Denver, CO, USA, 2002, pp. 257–268.

[73] M. Fisher, G. Rothermel, T. Creelan, M. Burnett, Scaling a Dataflow Testing Methodology to the Multiparadigm World of Commercial Spreadsheets, in: Proceedings of the 17th International Symposium on Software Reliability Engineering (ISSRE 2006), Raleigh, NC, USA, 2006, pp. 13–22.

[74] N. Randolph, J. Morris, G. Lee, A Generalised Spreadsheet Verification Methodology, in: Proceedings of the 25th Australasian Conference on Computer Science (ACSC 2002), 2002, pp. 215–222.

[75] K. Rothermel, C. Cook, M. Burnett, J. Schonfeld, T. R. G. Green, G. Rothermel, WYSIWYT Testing in the Spreadsheet Paradigm: An Empirical Evaluation, in: Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000), Limerick, Ireland, 2000, pp. 230–239.

[76] M. Fisher, II, M. Cao, G. Rothermel, C. Cook, M. Burnett, Automated Test Case Generation for Spreadsheets, in: Proceedings of the 24th International Conference on Software Engineering (ICSE 2002), Orlando, FL, USA, 2002, pp. 141–151.

[77] M. Fisher, II, G. Rothermel, D. Brown, M. Cao, C. Cook, M. Burnett, Integrating Automated Test Generation into the WYSIWYT Spreadsheet Testing Methodology, ACM Transactions on Software Engineering and Methodology 15 (2) (2006) 150–194.

[78] R. Abraham, M. Erwig, AutoTest: A Tool for Automatic Test Case Generation in Spreadsheets, in: Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2006), Brighton, United Kingdom, 2006, pp. 43–50.

[79] M. Burnett, C. Cook, O. Pendse, G. Rothermel, J. Summet, C. Wallace, End-User Software Engineering with Assertions in the Spreadsheet Paradigm, in: Proceedings of the 25th International Conference on Software Engineering (ICSE 2003), Portland, Oregon, 2003, pp. 93–103.

[80] A. Wilson, M. Burnett, L. Beckwith, O. Granatir, L. Casburn, C. Cook, M. Durham, G. Rothermel, Harnessing Curiosity to Increase Correctness in End-User Programming, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI 2003), 2003, pp. 305–312.

[81] L. Beckwith, M. Burnett, C. Cook, Reasoning about Many-to-Many Requirement Relationships in Spreadsheets, in: Proceedings of the

IEEE Symposia on Human Centric Computing Languages and Environments (HCC 2002), Arlington, VA, USA, 2002, pp. 149–157.

[82] K. McDaid, A. Rust, B. Bishop, Test-Driven Development: Can it Work for Spreadsheets?, in: Proceedings of the 4th International Workshop on End-User Software Engineering (WEUSE 2008), Leipzig, Germany, 2008, pp. 25–29.

[83] F. Hermans, Improving Spreadsheet Test Practices, in: Proceedings of the 23rd Annual International Conference on Computer Science and Software Engineering (CASCON 2013), Markham, Ontario, Canada, 2013, pp. 56–69.

[84] R. Abraham, M. Erwig, Mutation Operators for Spreadsheets, IEEE Transactions on Software Engineering 35 (1) (2009) 94–108.

[85] R. Abraham, M. Erwig, Goal-Directed Debugging of Spreadsheets, in: Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2005), Dallas, TX, USA, 2005, pp. 37–44.

[86] R. A. Joseph Lawrance, Margaret Burnett, M. Erwig, Sharing reasoning about faults in spreadsheets: An empirical study, in: Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2006), 2006, pp. 35–42.

[87] B. Hofer, A. Riboira, F. Wotawa, R. Abreu, E. Getzner, On the Empirical Evaluation of Fault Localization Techniques for Spreadsheets, in: Proceedings of the 16th International Conference on Fundamental Approaches to Software Engineering (FASE 2013), Rome, Italy, 2013, pp. 68–82.

[88] D. Jannach, T. Schmitz, Model-based diagnosis of spreadsheet programs - A constraint-based debugging approach, Automated Software Engineering to appear.

[89] J. Reichwein, G. Rothermel, M. Burnett, Slicing Spreadsheets: An Integrated Methodology for Spreadsheet Testing and Debugging, in: Proceedings of the 2nd Conference on Domain-Specific Languages (DSL 1999), Austin, Texas, 1999, pp. 25–38.

[90] J. R. Ruthruff, S. Prabhakararao, J. Reichwein, C. Cook, E. Creswick, M. Burnett, Interactive, Visual Fault Localization Support for End-User Programmers, Journal of Visual Languages & Computing 16 (1-2) (2005) 3–40.

[91] J. R. Ruthruff, M. Burnett, G. Rothermel, An Empirical Study of Fault Localization for End-User Programmers, in: Proceedings of the 27th International Conference on Software Engineering (ICSE 2005), St. Louis, MO, USA, 2005, pp. 352–361.

[92] J. R. Ruthruff, M. Burnett, G. Rothermel, Interactive Fault Localization Techniques in a Spreadsheet Environment, IEEE Transactions on Software Engineering 32 (4) (2006) 213–239.

[93] Y. Ayalew, R. Mittermeir, Spreadsheet Debugging, in: Proceedings of the European Spreadsheet Risks Interest Group 4th Annual Conference (EuSpRIG 2003), Dublin, Ireland, 2003.

[94] D. Jannach, U. Engler, Toward model-based debugging of spreadsheet programs, in: Proceedings of the 9th Joint Conference on Knowledge-Based Software Engineering (JCKBSE 2010), Kaunas, Lithuania, 2010, pp. 252–264.

[95] E. Tsang, Foundations of Constraint Satisfaction, Academic Press, 1993.

[96] C. Mateis, M. Stumptner, D. Wieland, F. Wotawa, Model-Based Debugging of Java Programs, in: Proceedings of the Fourth International Workshop on Automated Debugging (AADEBUG 2000), Munich, Germany, 2000.

[97] A. Felfernig, G. Friedrich, D. Jannach, M. Stumptner, Consistency-based diagnosis of configuration knowledge bases, Artificial Intelligence 152 (2) (2004) 213–234.

[98] D. Jannach, A. Baharloo, D. Williamson, Toward an integrated framework for declarative and interactive spreadsheet debugging, in: Procedings of the 8th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2013), Angers, France, 2013, pp. 117–124.

[99] R. Abreu, A. Riboira, F. Wotawa, Constraint-based Debugging of Spreadsheets, in: Proceedings of the 15th Ibero-American Conference on Software Engineering (CIbSE 2012), Buenos Aires, Argentina, 2012, pp. 1–14.

[100] R. Abreu, A. Riboira, F. Wotawa, Debugging Spreadsheets: A CSP-based Approach, in: Proceedings of the 23rd IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW 2012), Dallas, TX, USA, 2012, pp. 159–164.

[101] R. Reiter, A Theory of Diagnosis from First Principles, Artificial Intelligence 32 (1) (1987) 57–95.

[102] S. Außerlechner, S. Fruhmann, W. Wieser, B. Hofer, R. Spörk, C. Mühlbacher, F. Wotawa, The Right Choice Matters! SMT Solving Substantially Improves Model-Based Debugging of Spreadsheets, in: Proceedings of the 13th International Conference on Quality Software (QSIC 2013), Nanjing, China, 2013, pp. 139–148.

[103] R. Abraham, M. Erwig, GoalDebug: A Spreadsheet Debugger for End Users, in: Proceedings of the 29th International Conference on Software Engineering (ICSE 2007), Minneapolis, MN, USA, 2007, pp. 251–260.

[104] R. Abraham, M. Erwig, Test-driven goal-directed debugging in spreadsheets, in: Proceedings of the IEEE Symposium on Visual Languages and Human Centric Computing (VL/HCC 2008), Herrsching am Ammersee, Germany, 2008, pp. 131–138.

[105] C. Parnin, A. Orso, Are Automated Debugging Techniques Actually Helping Programmers?, in: Proceedings of the 2011 International Symposium on Software Testing and Analysis (ISSTA 2011), Toronto, Canada, 2011, pp. 199–209.

[106] M. Luckey, M. Erwig, G. Engels, Systematic Evolution of Model-Based Spreadsheet Applications, Journal of Visual Languages & Computing 23 (5) (2012) 267–286.

[107] J. Paine, Model Master: an object-oriented spreadsheet front-end, in: Proceedings of the CALECO Conference on Using Computer Technology in Economics and Business (CALECO 1997), Bristol, United Kingdom, 1997, pp. 84–92.

[108] J. Paine, Ensuring Spreadsheet Integrity with Model Master, in: Proceedings of the European Spreadsheet Risks Interest Group 2nd Annual Conference (EuSpRIG 2001), Amsterdam, Netherlands, 2001.

[109] J. Paine, Spreadsheet Structure Discovery with Logic Programming, in: Proceedings of the European Spreadsheet Risks Interest Group 5th Annual Conference (EuSpRIG 2004), Klagenfurt, Austria, 2004.

[110] J. Paine, Excelsior: Bringing the Benefits of Modularisation to Excel, in: Proceedings of the European Spreadsheet Risks Interest Group 6th Annual Conference (EuSpRIG 2005), London, United Kingdom, 2005.

[111] J. Paine, E. Tek, D. Williamson, Rapid Spreadsheet Reshaping with Excelsior: multiple drastic changes to content and layout are easy when you represent enough structure, in: Proceedings of the European Spreadsheet Risks Interest Group 7th Annual Conference (EuSpRIG 2006), Cambridge, United Kingdom, 2006.

[112] M. Erwig, R. Abraham, I. Cooperstein, S. Kollmansberger, Automatic Generation and Maintenance of Correct Spreadsheets, in: Proceedings of the 27th International Conference on Software Engineering (ICSE 2005), St. Louis, MO, USA, 2005, pp. 136–145.

[113] R. Abraham, M. Erwig, S. Kollmansberger, E. Seifert, Visual Specifications of Correct Spreadsheets, in: Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2005), Dallas, TX, USA, 2005, pp. 189–196.

[114] M. Erwig, R. Abraham, S. Kollmansberger, I. Cooperstein, Gencel: A Program Generator for Correct Spreadsheets, Journal of Functional Programming 16 (3) (2006) 293–325.

[115] R. Abraham, M. Erwig, Inferring Templates from Spreadsheets, in: Proceedings of the 28th International Conference on Software Engineering (ICSE 2006), Shanghai, China, 2006, pp. 182–191.

[116] T. R. G. Green, M. Petre, Usability Analysis of Visual Programming Environments: a 'cognitive dimensions' framework, Journal of Visual Languages & Computing 7 (2) (1996) 131–174.

[117] A. Blackwell, T. R. G. Green, Notational Systems – the Cognitive Dimensions of Notations Framework, HCI Models, Theories, and Frameworks: Toward a Multidisciplinary Science (2003) 103–134.

[118] G. Engels, M. Erwig, ClassSheets: Automatic Generation of Spreadsheet Applications from Object-Oriented Specifications, in: Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005), Long Beach, CA, USA, 2005, pp. 124–133.

[119] J. Cunha, M. Erwig, J. Saraiva, Automatically Inferring ClassSheet Models from Spreadsheets, in: Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2010), Madrid, Spain, 2010, pp. 93–100.

[120] J. Cunha, J. Visser, T. Alves, J. a. Saraiva, Type-Safe Evolution of Spreadsheets, in: Proceedings of the 14th International Conference on Fundamental Approaches to Software Engineering: Part of the Joint European Conferences on Theory and Practice of Software (FASE 2011/ETAPS 2011), Saarbrücken, Germany, 2011, pp. 186–201.

[121] J. Cunha, J. Mendes, J. Saraiva, J. Fernandes, Embedding and Evolution of Spreadsheet Models in Spreadsheet Systems, in: Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2011), Pittsburgh, PA, USA, 2011, pp. 179–186.

[122] J. Cunha, J. Fernandes, J. Mendes, H. Pacheco, J. Saraiva, Bidirectional Transformation of Model-Driven Spreadsheets, in: Proceedings of the 5th International Conference on Theory and Practice of Model Transformations (ICMT 2012), Springer Lecture Notes in Computer Science, Prague, Czech Republic, 2012, pp. 105–120.

[123] J. Cunha, J. Fernandes, J. Mendes, J. Saraiva, Extension and Implementation of ClassSheet Models, in: Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2012), Innsbruck, Austria, 2012, pp. 19–22.

[124] J. Cunha, J. a. P. Fernandes, J. a. Saraiva, From Relational ClassSheets to UML+OCL, in: Proceedings of the 27th Annual ACM Symposium on Applied Computing (SAC 2012), Trento, Italy, 2012, pp. 1151–1158.

[125] F. Hermans, M. Pinzger, A. van Deursen, Automatically Extracting Class Diagrams from Spreadsheets, in: Proceedings of the 24th European Conference on Object-Oriented Programming (ECOOP 2010), Maribor, Slovenia, 2010, pp. 52–75.

[126] J. Cunha, J. a. Saraiva, J. Visser, From Spreadsheets to Relational Databases and Back, in: Proceedings of the 2009 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation (PEPM 2009), Savannah, GA, USA, 2009, pp. 179–188.

[127] J. Cunha, J. Saraiva, J. Visser, Discovery-Based Edit Assistance for Spreadsheets, in: Proceedings of the IEEE Symposium on Visual Languages and Human Centric Computing (VL/HCC 2009), Corvallis, OR, USA, 2009, pp. 233–237.

[128] J. Cunha, J. a. Saraiva, J. Visser, Model-Based Programming Environments for Spreadsheets, in: Proceedings of the 16th Brazilian Conference on Programming Languages (SBLP 2012), Natal, Brazil, 2012, pp. 117–133.

[129] L. Beckwith, J. Cunha, J. Fernandes, J. Saraiva, End-Users Productivity in Model-Based Spreadsheets: An Empirical Study, in: Proceedings of the 3rd International Symposium on End-User Development (IS-EUD 2011), Springer Lecture Notes in Computer Science, Torre Canne, Italy, 2011, pp. 282–288.

[130] N. Wilde, C. Lewis, Spreadsheet-based interactive graphics: from prototype to tool, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI 1990), Seattle, WA, USA, 1990, pp. 153–160.

[131] C. Lewis, NoPumpG: Creating Interactive Graphics with Spreadsheet Machinery, Visual Programming Environments: Paradigms and Systems (1990) 526–546.

[132] C. Hughes, J. Moshell, Action Graphics: A Spreadsheet-based Language for Animated Simulation, Visual Languages and Applications (1990) 203–235.

[133] N. P. Wilde, A WYSIWYC (What You See Is What You Compute) Spreadsheet, in: Proceedings of the IEEE Symposium on Visual Languages (VL 1993), Bergen, Norway, 1993, pp. 72–76.

[134] M. M. Burnett, A. Agrawal, P. van Zee, Exception Handling in the Spreadsheet Paradigm, IEEE Transactions on Software Engineering 26 (10) (2000) 923–942.

[135] R. W. Sebesta, Concepts of Programming Languages (4th ed.), Addison-Wesley-Longman, 1999.

[136] B. Bekenn, R. Hooper, Reducing Spreadsheet Risk with FormulaDataSleuth, in: Proceedings of the European Spreadsheet Risks Interest Group 9th Annual Conference (EuSpRIG 2008), London, United Kingdom, 2008.

[137] C. Chambers, M. Erwig, M. Luckey, SheetDiff: A Tool for Identifying Changes in Spreadsheets, in: Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2010), Madrid, Spain, 2010, pp. 85–92.

[138] A. Harutyunyan, G. Borradaile, C. Chambers, C. Scaffidi, Planted-model evaluation of algorithms for identifying differences between spreadsheets, in: Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2012), Innsbruck, Austria, 2012, pp. 7–14.

[139] P. O'Beirne, Spreadsheet Refactoring, in: Proceedings of the European Spreadsheet Risks Interest Group 11th Annual Conference (EuSpRIG 2010), London, United Kingdom, 2010.

[140] S. Badame, D. Dig, Refactoring meets Spreadsheet Formulas, in: Proceedings of the 28th IEEE International Conference on Software Maintenance (ICSM 2012), Riva del Garda, Trento, Italy, 2012, pp. 399–409.

[141] W. R. Harris, S. Gulwani, Spreadsheet Table Transformations from Examples, in: Proceedings of the 32nd ACM SIGPLAN conference on Programming language design and implementation (PLDI 2011), San Jose, CA, USA, 2011, pp. 317–328.

[142] Y. Ye, G. Fischer, Reuse-Conducive Development Environments, Automated Software Engineering 12 (2) (2005) 199–235.

[143] R. W. Djang, M. M. Burnett, Similarity Inheritance: A New Model of Inheritance for Spreadsheet VPLs, in: Proceedings of the IEEE Symposium on Visual Languages (VL 1998), Halifax, NS, Canada, 1998, pp. 134–141.

[144] M. Montigel, Portability and Reuse of Components for Spreadsheet Languages, in: Proceedings of the IEEE CS International Symposium on Human-Centric Computing Languages and Environments (HCC 2002), Arlington, VA, USA, 2002, pp. 77–79.

[145] D. Jannach, M. Zanker, M. Ge, M. Gröning, Recommender systems in computer science and information systems - a landscape of research, in: Proceedings of the 13th International Conference on E-Commerce and Web Technologies (EC-WEB 2012), Vienna, 2012, pp. 76–87.

[146] L. P. S. Elazar J. Pedhazur, Measurement Design and Analysis: An Integrated Approach, Lawrence Erlbaum Assoc Inc, 1991.

[147] Using a structured design approach to reduce risks in end user spreadsheet development, Information and Management 37 (1) (2000) 1–12.

[148] F. Karlsson, Using two heads in practice, in: Proceedings of the 4th International Workshop on End-user Software Engineering (WEUSE 2008), Leipzig, Germany, 2008, pp. 43–47.

[149] R. R. Panko, Applying Code Inspection to Spreadsheet Testing, Journal of Management Information Systems 16 (2) (1999) 159–176.

[150] R. R. Panko, R. H. S. Jr., Hitting the wall: errors in developing and code inspecting a 'simple' spreadsheet model, Decision Support Systems 22 (4) (1998) 337–353.

[151] J. P. A. Ioannidis, Why most published research findings are false, PLoS Medizine 2 (8).

[152] R. Nuzzo, Scientific method: Statistical errors, Nature 506 (2014) 150–152.

[153] A. J. Ko, R. Abraham, L. Beckwith, A. Blackwell, M. Burnett, M. Erwig, C. Scaffidi, J. Lawrance, H. Lieberman, B. Myers, M. B. Rosson, G. Rothermel, M. Shaw, S. Wiedenbeck, The State of the Art in End-User Software Engineering, ACM Computing Surveys 43 (3) (2011) 21:1–21:44.

[154] S. R. Thorne, D. Ball, Z. Lawson, A Novel Approach to Formulae Production and Overconfidence Measurement to Reduce Risk in Spreadsheet Modelling, in: Proceedings of the European Spreadsheet Risks Interest Group 5th Annual Conference (EuSpRIG 2004), Klagenfurt, Austria, 2004.

[155] L. Beckwith, S. Sorte, M. Burnett, S. Wiedenbeck, T. Chintakovid, C. Cook, Designing features for both genders in end-user programming environments, in: Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2005), Dallas, TX, 2005, pp. 153–160.

[156] J. R. Ruthruff, A. Phalgune, L. Beckwith, M. M. Burnett, C. R. Cook, Rewarding "good" behavior: End-user debugging and rewards, in: Proceedings of the IEEE Symposium on Visual Languages and Human Centric Computing (VL/HCC 2004), 2004, pp. 115–122.

[157] S. P. Jones, A. Blackwell, M. Burnett, A user-centred approach to functions in excel, in: Proceedings of the 8th ACM SIGPLAN International Conference on Functional Programming (ICFP 2003), Uppsala, Sweden, 2003, pp. 165–176.

[158] P. Sestoft, J. Z. Sørensen, Sheet-defined functions: Implementation and initial evaluation, in: End-User Development, Vol. 7897 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2013, pp. 88–103.