DOI: 10.51984/JOPAS.V21I4.2140

جامعة سبها للعلوم البحتة والتطبيقية مجلة
Sebha University Journal of Pure & Applied Sciences

Journal homepage: www.sebhau.edu.ly/journal/index.php/jopas

A Comparative Study of Automated Testing Tools for Spreadsheets

Ali Aburas

Department of Computer Science, Faculty of Science, University of Tripoli, Tripoli, Libya

Keywords:
Automated Tools
End-user
Errors
Spreadsheets
Testing

ABSTRACT

Organizations and industries use spreadsheet programs, such as MS Excel, for various tasks, such as accounting, financial calculations, and reporting. However, the number of spreadsheets that contain errors is very high. One primary reason is that different end-users develop spreadsheets without programming or software development training. Thus, researchers have proposed a variety of automated tools and techniques to support the end-users in finding and fixing errors in spreadsheets. The main objective of this paper is to study different automated tools in spreadsheet testing and classify them according to two criteria: how they work and the ability to find spreadsheet faults. For this purpose, we discuss various spreadsheet testing automated tools, techniques, and strategies. In addition, we highlight their performance and where they can be used so that they can be beneficial to both end-users and researchers. We then set out key directions for potential future works.

دراسة مقارنة لأدوات الاختبار الآلي للجداول البيانات

علي أبوراس

قسم علوم الحاسب الآلي، كلية العلوم، جامعة طرابلس، طرابلس، ليبيا

الكلمات المفتاحية:
الأدوات الآلية
المستخدم النهائي
الأخطاء
جداول البيانات
اختبارات

الملخص

تستخدم قطاعات المؤسسات والصناعات برامج جداول البيانات، مثلا ميكروسوفت اكسل، لمختلف المهام، مثلا في المحاسبة والحسابات المالية وإعداد التقارير. ومع ذلك، فإن عدد جداول البيانات التي تحتوي على أخطاء مرتفع للغاية. أحد الأسباب الأساسية هو أن المستخدمين النهائيين المختلفين يطورون جداول بيانات بدون تدريب على البرمجة أو تطوير البرامج. وبالتالي، اقترح الباحثون مجموعة متنوعة من الأدوات والتقنيات الآلية لدعم المستخدمين النهائيين في العثور على الأخطاء وإصلاحها في جداول البيانات. الهدف الرئيسي من هذه الورقة هو دراسة الأدوات الآلية المختلفة في اختبار جداول البيانات وتصنيفها وفقًا لمعيارين: كيفية عملها والقدرة على العثور على أخطاء جداول البيانات. لهذا الغرض، نناقش مختلف الأدوات والتقنيات والاستراتيجيات الآلية لاختبار جداول البيانات. بالإضافة إلى ذلك، نسلط الضوء على أدائها وأين يمكن استخدامها حتى تكون مفيدة لكل من المستخدمين النهائيين والباحثين. ثم وضعنا الاتجاهات الرئيسية للأعمال المستقبلية المحتملة.

Introduction

The flexibility afforded by spreadsheet programs allow end-users and many organizations to use spreadsheets for various purposes such as accounting, financial calculations, reporting, and decision supporting tasks [1]. Spreadsheet programs are one of the most well-known programming systems among end-user programmers. End-users can easily use them without requiring training in programming [2]. In general, end-users often write spreadsheet programs to support their work or for personal use, and they do not have a solid background in computer science [2]. For example, a teacher can create a spreadsheet file to calculate grades, track attendances, and class schedules during the school year.

Since spreadsheets are, in most cases, created by end-users who are self-taught and have no education in software engineering, the error rate is assumed higher than in traditional software [3]. An experiment-based study on spreadsheet development reports that more than 95% of spreadsheets contain at least one error [4]. The implications of the errors in a spreadsheet are severe, and the consequences could be very costly for organizations, e.g., massive financial loss or loss of

Corresponding author:
E-mail addresses: al.aburas@uot.edu.ly
Article History : Received 31 May 2022 - Received in revised form 01 September 2022 - Accepted 03 October 2022

reputation*. To minimize these risks, it is essential to support end-users with appropriate methods and techniques to find and fix such faults.

Testing is essential for ensuring the correctness of spreadsheets [5]. Testing can be performed either manually or automatically. Manual testing is time-consuming and requires more effort [6]. A study shows that end-users typically underestimate the required testing effort, generating tests that are largely manually and generally not comprehensive [7]. Due to the need to ensure spreadsheet quality, researchers have proposed various automated approaches to prevent, detect, and remove spreadsheet errors.

This research intends to study and compare automated testing spreadsheet tools' concepts, features, and errors detection ability. Furthermore, we will classify a set of automated spreadsheet testing tools and distribute them over the type of approaches for preventing, detecting, and correcting spreadsheet errors. The fundamental goal is to analyze the features supported by these automated spreadsheet testing tools that aid in minimizing the resources in creating and testing spreadsheet files and increasing efficiency for maintenance and reuse of spreadsheet files.

The rest of this paper is organized as follows: Section II presents a variety of effective approaches for preventing, detecting and correcting spreadsheet errors. Then, Section III contains a full discussion of potential future works. Finally, a conclusion statement about the studied approaches is given.

TYPES OF APPROACHES

Spreadsheet programs like MS Excel lack any higher-level abstractions and include limited functionality to help end-users to create error-free spreadsheets that implement complex models. As a result, end-users make errors when creating or reusing complex spreadsheet files. Researchers have developed different approaches that engage various software engineering activities to support end-users in the error detection and fault localization process. These approaches are generally focused on preventing, detecting, and removing errors from spreadsheets. This section will illustrate some error prevention, auditing, testing, and automatic consistency checking approaches to help end-users create error-free spreadsheets.

1. Preventing errors

Spreadsheets are error-prone because they do not impose any restrictions on the kinds of updates that can be carried out. For example, end-users copy and paste formulas or even drag on a cell to fill another column, which can introduce severe errors and make it difficult to find them if references are not correctly updated [8]. This problem has led to the development of approaches, which are called preventing errors, that help end-users to create spreadsheets that do not have errors in the first place [9].

Researchers found that programmers make fewer syntax errors and focus on the logic of a program when they use a block-based language instead of a textual-based one [10]. As a result, researchers use visual-based approaches to provide end-users with a visually enhanced representation of some aspects of the spreadsheet to understand its basic structure and dependencies between the formulas [9]. Visual-based approaches aim to reduce formula errors and increase end-users efficiency while end-users edit and construct formulas. It does so by interactively visualizing hidden data and formulas. Thus, end-users can immediately see relevant data and results. In addition, visualizing referenced data and formulas is especially helpful for end-users working with large spreadsheets when navigating between different cells, formulas, and multiple sub-sheets.

Jansen and Herman [11] developed a block-based formula editor, called XLBlocks, for spreadsheets to support end-users while developing or maintaining formulas. XLBlocks aims to create formulas with a block-based language instead of the default textual formula language and translate them automatically into valid spreadsheet formulas. XLBlocks also introduces new, more straightforward functions to use than some built-in Excel functions such as SUM, SUMIFS, IFERROR, INDEX, MATCH, VLOOKUP, IF, −, /, >, and <.

As shown in Fig. 1, the user defines the formula in (a), and gives the formula a name (b). The user also has to specify cells in the spreadsheet that will receive this formula (c), and the functions that are used in the formula. The user can add comments, which are not transferred to the spreadsheet [11].

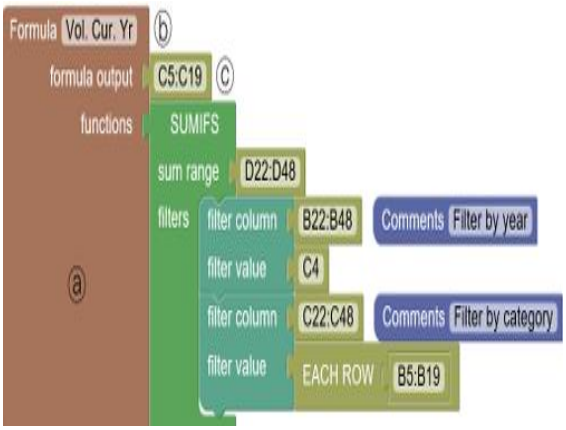


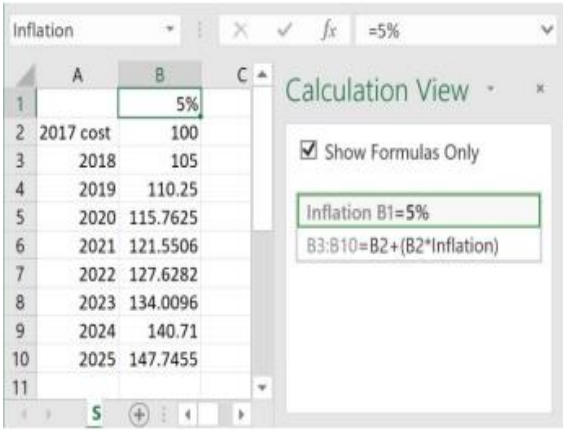
Fig. 1: Example of a block definition of a SUMIFS formula.

A think-aloud study [12] investigated the feasibility of this approach, and it showed that XLBlocks helps end-users create formulas without considering the syntax of the formulas. In addition, editing a formula part is easier in XLBlocks than the default text-based formula editor because it can easily drag and drop different parts of the formula.

Research on the showing computation in spreadsheets has shown that they positively affect the comprehension of the spreadsheet's formulas. Another advantage of showing the spreadsheet's formulas is that it minimizes the risk of introducing new errors during debugging and maintenance tasks.

Sarkar et. al. [13] also introduced Calculation View, a multiple-representation approach for viewing formulas and their groupings. Their approach uses a simple textual syntax for copying a formula into a block of cells and naming cells or ranges and referring to those names in other formulas. Calculation View allows end-users to give domain-relevant names for grid cell references; for example, they can refer to cell B2 as TaxRate, which is easier to read, makes formulas faster to type, debug and easy to update. Figure 2 depicts the ability of assign names and assign formulas to individual cells.

A user study was conducted with 22 participants who had prior experience with spreadsheets to assess the effectiveness of Calculation View. One goal of the study was to evaluate if the subjects would create and reason about spreadsheets with less manual and cognitive effort. The study shows that Calculation View improves the subjects' performance when performing copy/paste and debugging tasks. The end-users also tend to make fewer errors in editing a block of formulas and can focus on the logic of the formulas when they use Calculation View instead of a textual one.



* http://www.eusprig.org/horror-stories.htm

Fig. 2: Calculation View views the formulas in abstract operations such as range assignment and cell naming.

The prevention error approaches require additional programming in a new language that end-users must learn, which can be very time-consuming. Therefore, researchers have explored different approaches to visualize the computation hiding and identify groups of related cells in spreadsheets. The following section shows some automated techniques that aim to assist end-users with detecting and debugging spreadsheet formulas via visualization.

2. Detecting and Fixing errors

Researchers have developed different static checkers approaches and tools for automated fault localization. Such approaches are based on static code analysis that analyzes the tables, the formulas, and the dependencies between them [9]. They are designed to help end-users detect or remove errors, particularly during maintenance and debugging activities [1].

Schmitz et al. [14] proposed an approach that automatically partitions a large spreadsheet into a set of semantically equivalent calculation areas called "fragments." Once such areas are identified, the end-user is only required to check the correctness of a smaller part (i.e., one copy of the calculations) of the fragment. If this part is considered correct, the other areas containing identical calculations are also assumed to be correct. Their approach starts by defining a base fragment that only contains cells that share the same column or row with another fragment cell. The base fragment comprises either a single formula or a set of copy-equivalent formulas, i.e., they do the same calculations but use different inputs. Then, the base fragments are merged into larger ones in an evolutionary process to reduce the complexity when searching for the possible causes of the problem.

Barowy et al. [8] also proposed a learning-based approach, called ExceLint, that identifies likely spreadsheet errors based on discrepancies with other nearby formulas. ExceLint extracts data dependencies for every formula in the given spreadsheet by parsing a sheet's formulas and building the program's data-flow graph. Then, ExceLint exploits the intrinsically rectangular layout of spreadsheets to identify homogeneous, rectangular regions that contain formulas with identical reference behavior. Finally, ExceLint identifies suspect formulas by comparing cells to adjacent rectangular regions. To determine whether a formula is likely to be an error, ExceLint uses the cell's position in the layout and identifies formulas that are especially surprising disruptions to nearby rectangular regions.

Recently, a lot of effort has been put into exposing end-users to machine learning (ML), such as spam filtering and recommendation systems. ML is a subset of artificial intelligence where statistical algorithms process a large of data to build a model to become more accurate at predicting outcomes without being explicitly programmed. There are two types of ML-based learning algorithms: supervised and unsupervised algorithms. In supervised learning algorithms, systems are able to predict based on past data. The learning algorithms require labeled input and output data to train and build a model. Then, they identify patterns in data, learn from observations, and make predictions. On the other hand, unsupervised algorithms identify hidden patterns and determine the correlations and relationships from unlabeled data provided. The algorithms try to group the data into clusters, and the anomalies become more evident. The rise in data availability has encouraged researchers to apply ML-based techniques to make inferences and predict faults in spreadsheets.

Cheung et al. [15] proposed CUSTODES which uses an unsupervised learning-based approach to automatically detect faults in spreadsheets. First, it computes different features of spreadsheets and then uses cluster algorithm to put all cells with similar formulas in one group. Then, CUSTODES predicts if a cell is faulty if it is not in the identified group. CUSTODES utilizes strong features to extract similar cells and put them in one cluster. For examples, cell formulas and cell reference relations those cells in [F11:F16] in Fig 3. CUSTODES also refines each cluster by using weak features, such as cell labels, standard layouts, and fonts. For example, the header "Total" in F9 can be used to cluster cells in F11 to F19. CUSTODES identifies outlier cells as smells if they cannot be clustered or grouped. For example, cell F17 might be faulty since it contains a value, whereas cells from F11 to F19 comprise formulas.

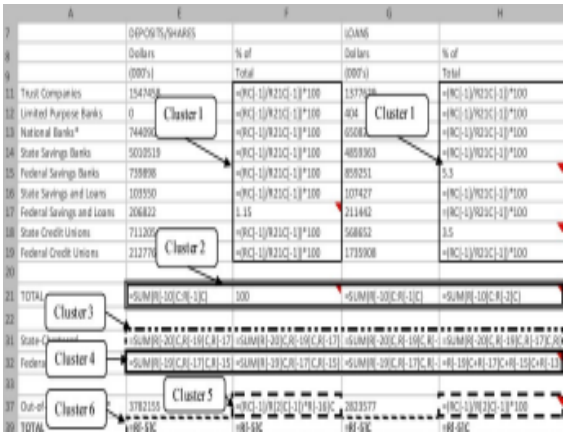


Fig. 3: An example for illustrating CUSTODES's whole cluster technique.

Regarding the CUSTODES clustering technique evaluation, the experiment showed that CUSTODES was able to detect up to 78% of the faulty cells in 70 spreadsheets files randomly sampled from the EUSES corpus.

Koch et al. [16] proposed an approach that uses spreadsheet metrics with Random Forests (RF) as an ML-based technique to predict faults in spreadsheet formulas. The main idea of their approach is to use a set of collection of spreadsheets containing labeled faulty formulas, incorrect or correct, to train an ML model for fault prediction. They selected a set of 64 metrics, such as the cell's column number (position), the number of any references to empty cells, and the number of references to other spreadsheets, as a learning dataset. The learning set is obtained by taking the labeled spreadsheet as input and computing a value for each formula of the spreadsheet. Their approach utilizes Random Forests (RF) to return the likelihood of unseen a formula is faulty. An evaluation of their approach on different datasets containing faulty spreadsheets showed that Random Forests (RF) could predict spreadsheets' faults.

A spreadsheet file is a table, a rectangle block of cells describing a business process. A table contains two elements: the header region and the data region. Spreadsheet programs, such as MS Excel, do not provide documentation to show clear table structure information. Identifying spreadsheet semantic table structures can help end-users understand the spreadsheet, which is a fundamental step in detecting errors. Therefore, Zhang et al. [17] proposed an automated approach called TasiError (Table Structure Identification Error Detection). TasiError uses a multi-classifier, e.g., Random Forest, Logistic Regression, and Decision Tree, to train the model and automatically identifies semantic table structures in spreadsheets. TasiError detects two types of most common errors: missing formulas and formula errors. They evaluated their approach and compared it with other existing tools, CUSTODES [15], and [8]. TasiError outperforms the other existing tools. These results show that semantic table structures can help error detection in spreadsheets.

Testing is an important approach to identify errors, reveal failures, and to assure the quality of software systems [5]. That is why developers test their programs and spend a large amount of time identifying and correcting errors within their programs [9]. The following section shows some testing approaches in the context of spreadsheets.

3. Testing Spreadsheets

Recent studies highlighted several challenges of testing in spreadsheets via interviewing spreadsheet end-users and conducting an online survey with industrial spreadsheet end-users [5] [18]. Roy et al. [5] report that end-users do manual and informal tests to ensure the correctness of their spreadsheets, and the most popular technique was ad-hoc testing.

Spreadsheet systems such as MS Excel do not support systematic testing of spreadsheets. Smith et al. [18] reported that among their respondents, 87% reported examining cells individually or using a calculator to check selected cells and 79% reported using common sense. As a result, various automated testing techniques and tools were proposed for spreadsheets to help end-users identify and correct errors by providing them with better spreadsheet testing.

The most notable of these approaches is a visual and incremental spreadsheet testing methodology called "What You See Is What You Test" (WYSIWYT) [19]. Whenever the end-user enters values into the spreadsheet, the end-user marks the output cells, which contain formulas, with a symbol ✓ if the output is correct and an × symbol if incorrect. WYSIWYT uses definitions-use (du) adequacy for keeping track of which cells in the spreadsheet have been tested and determines the "testedness" of the spreadsheet. The testedness is shown to the end-user by coloring the borders of the non-input cells from untested (red color) to tested (blue color) and through a progress bar, which ranges from 0 to 100 percent. Several enhancements to this approach were proposed, such as automatically generating test cases [20], and later WYSIWYT approach was integrated into Microsoft Excel [21].

Sohon et al. proposed an approach that aims to implement an invariant-based testing approach [22]. In the context of spreadsheets, invariants are proprieties that remain true after operations throughout the spreadsheet's usage. The properties are usually indicated by the labels of the individual cells or based on rows or columns. Sohon et al. explored the effectiveness of automatically inferred invariants to help end-users in ensuring the validity of their spreadsheets. They used the Daikon tool to infer invariants from the spreadsheets automatically. Daikon takes spreadsheets data saved in .CSV format and infers several invariants from the spreadsheet. However, the organizations of the structures of the spreadsheets must be converted into a single tabular structure with proper column headers. Let us, for example, consider the spreadsheet partly shown in Figure 4.

	A	B	C	D	F	G	H	I	J	K
7	Prod.	Contract	Contract	Fixed	Fuel	Settled	Futures	(Gains) and Losses		
8	Date	Number	Name	Price	Dth	Price	Price	Total	Realized	Unrealized
9						El Paso SJ		(4)	(4)	(4)
10	Jun-98		Avista	$ 2.22	7,500	$ 1.82		$ 3,000.00	$ 3,000.00	
11	Jul-98		Avista	$ 2.22	7,750	$ 1.86		$ 2,790.00	$ 2,790.00	
12	Aug-98		Avista	$ 2.22	7,750	$ 1.81		$ 3,177.50	$ 3,177.50	
13	Sep-98		Avista	$ 2.22	7,500	$ 1.55		$ 5,025.00	$ 5,025.00	
14	Oct-98		Avista	$ 2.22	7,750	$ 1.67		$ 4,262.50	$ 4,262.50	
15	Nov-98		Avista	$ 2.22	7,500	$ 1.88		$ 2,550.00	$ 2,550.00	
16	Dec-98		Avista	$ 2.22	7,750	$ 1.96		$ 2,015.00	$ 2,015.00	
17	Jan-99		Avista	$ 2.22	7,750	$ 1.72		$ 3,875.00	$ 3,875.00	
18	Feb-99		Avista	$ 2.22	7,000	$ 1.63		$ 4,130.00	$ 4,130.00	
19	Mar-99		Avista	$ 2.22	7,750	$ 1.51		$ 5,502.50	$ 5,502.50	
20	Apr-99		Avista	$ 2.22	7,500	$ 1.59		$ 4,725.00	$ 4,725.00	
21	May-99		Avista	$ 2.22	7,750	$ 2.03		$ 1,472.50	$ 1,472.50	
22					91,250			$ 42,525.00	$ 42,525.00	$ -
23										

Fig. 4: A portion of a typical Excel spreadsheet that targeted for invariant inference highlighted in orange.

For Daikon to work, the data blocks must be restructured in an acceptable format. Daikon infers a list of invariants that are only true and not redundant, as shown in Figure 5. As shown in Figure 5, the invariants "Settled >= 1.51" represents the formula in column G with the header "Settled Price El Paso SJ". The corresponding test case for this invariant would be =IF(G10>=1.51, "OK", "ERROR"). This test case can be used for all the rest of the corresponding cells and columns.

```
Total == Realized
ContractName one of { "Avista", "Engage" }
Fixed == 2.22
Settled >= 1.51
Fixed >Settled
```

Fig. 5: Example of Inferred Invariants.

As shown in the Figure 5, the invariants "Settled >= 1.51" represents the formula in the column G with the header "Settled Price El Paso SJ". The corresponding test case for this invariant would be =IF(G10>=1.51, "OK", "ERROR"). This test case can be used for all the rest of the corresponding cells and columns.

DISCUSSION AND FUTURE WORKS

This paper presented a review of different approaches that have been carried out to prevent, detect and remove errors from spreadsheets. All the proposed approaches work toward a common goal of helping end-users to create error-free spreadsheet files and improve the quality of their spreadsheet files. However, these approaches have not successfully prevented or detected all types of errors that end-users commonly make

Detecting and preventing errors in spreadsheets is difficult because of the significant input of the formula that needs to be explored. As we have seen, researchers proposed various approaches for end-users to prevent errors from spreadsheets by providing templates. These templates include all the formulas and the structure of the spreadsheet. Various approaches were proposed to assist the end-users test, localizing, and repairing faults in spreadsheets. Researchers implemented Machine learning (ML) techniques to reduce the end-users cognitive load and simplify the testing and debugging process. The main obstacle of the ML-based methods is unable to discover new types of faults, e.g., those that did not exist in the training data. One major limitation that spreadsheet systems like Microsoft Excel have is the lack of abstractions of code reuse. A promising area for future work in this direction is automatically recommending formulas to help end-users build a spreadsheet model more accurately. Recommendations tools can use the spreadsheet metrics to offer similar spreadsheets or formulas. In addition, such tools might overwhelm some end-users with less experience. As a result, novel interaction mechanisms with different ways of visualizing faulty formulas are required to better guide the end-users in the debugging process.

CONCLUSION

Spreadsheets are used as a basis for financial tasks, decision-making, and data analysis. End-users use spreadsheets for their simplicity, and there are many spreadsheets written by end-users who do not have a solid background in programming and testing.

Testing spreadsheet files is an essential activity and is a time-consuming and intensive process. Therefore, researchers have made a significant effort to help end-users create, test, and debug spreadsheets. This paper describes various software testing spreadsheet techniques, their work, and their error detection ability. We discussed in detail the concept of multiple techniques and approaches and illustrated examples of how they work and how they detect errors in spreadsheet files.

Based on our study of the automated testing tools for spreadsheet, we found the following:

- Some spreadsheet testing approaches require end-users to have an understanding of the testing. For some less-experienced end-users might be overwhelmed by some advanced approaches.
- Further improvements are possible for machine learning approaches. For example, recommendation engine techniques for recommended existing formulas or tables is a promising area of research.
- Researchers improved some existing techniques which need more end-user studies to examine the improved techniques value.

It is essential to understand how end-users' create and design spreadsheets to invent, design, and build new approaches to produce error-free spreadsheet files and improve the quality of the spreadsheet files. End-user studies are an excellent way to guide the researchers toward creating automated testing tools that best end-users needs.

Acknowledgment

We thank the anonymous reviewers for the valuable feedback and comments that helped us improve the quality of the manuscript.

References

[1]- Wensheng Dou, Shing-Chi Cheung, and Jun Wei. 2014. Is Spreadsheet Ambiguity Harmful? Detecting and Repairing Spreadsheet Smells due to Ambiguous Computation. In Proceedings of International Conference on Software Engineering (ICSE). 848–858.

[2]- Pak-Lok Poon, Fei-Ching Kuo, Huai Liu, and Tsong Yueh Chen. How can non-technical end users effectively test their spreadsheets? Information Technology & People, 2014.

[3]- Adil Mukhtar, Birgit Hofer, Dietmar Jannach, and Franz Wotawa. Spreadsheet debugging: The perils of tool over-reliance. Journal of Systems and Software, 184:111119, 2022.

[4]- Ray Panko. What we don't know about spreadsheet errors today: The facts, why we don't believe them, and what we need to do. arXiv preprint arXiv:1602.02601, 2016.

[5]- Sohon Roy, Felienne Hermans, and Arie Van Deursen. Spreadsheet testing in practice. In 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER), pages 338– 348. IEEE, 2017.

[6]- F Okezie, I Odun-Ayo, and S Bogle. A critical analysis of software testing tools. In Journal of Physics: Conference Series, volume 1378, page 042030. IOP Publishing, 2019.

[7]- Awais Azam and Khubaib Amjad Alam. Spreadsheet based software engineering. volume 2, pages 15–15, 2019.

[8]- Daniel W Barowy, Emery D Berger, and Benjamin Zorn. Excelint: automatically finding spreadsheet formula errors. Proceedings of the ACM on Programming Languages, 2(OOPSLA):1–26, 2018.

[9]- Dietmar Jannach, Thomas Schmitz, Birgit Hofer, and Franz Wotawa. Avoiding, finding and fixing spreadsheet errors–a survey of automated approaches for spreadsheet qa. Journal of Systems and Software, 94:129–150, 2014.

[10]- Thomas W Price and Tiffany Barnes. Comparing textual and block interfaces in a novice programming environment. In Proceedings of the eleventh annual international conference on international computing education research, pages 91–99, 2015.

[11]- Bas Jansen and Felienne Hermans. Xlblocks: A block-based formula editor for spreadsheet formulas. In 2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pages 55–63. IEEE, 2019.

[12]- Bas Jansen and Felienne Hermans. The effect of a block-based language on formula comprehension in spreadsheets. In 2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC), pages 288–299. IEEE, 2021.

[13]- Advait Sarkar, Andrew D Gordon, Simon Peyton Jones, and Neil Toronto. Calculation view: multiple-representation editing in spreadsheets. In 2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pages 85–93. IEEE, 2018.

[14]- Thomas Schmitz, Birgit Hofer, Dietmar Jannach, and Franz Wotawa. Fragment-based diagnosis of spreadsheets. In Federation of International Conferences on Software Technologies: Applications and Foundations, pages 372–387. Springer, 2016.

[15]- Shing-Chi Cheung, Wanjun Chen, Yepang Liu, and Chang Xu. Custodes: Automatic spreadsheet cell clustering and smell detection using strong and weak features. In 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), pages 464–475, 2016.

[16]- Patrick Koch, Konstantin Schekotihin, Dietmar Jannach, Birgit Hofer, and Franz Wotawa. Metric-based fault prediction for spreadsheets. IEEE Transactions on Software Engineering, 47(10):2195–2207, 2019.

[17]- Yakun Zhang, Xiao Lv, Haoyu Dong, Wensheng Dou, Shi Han, Dongmei Zhang, Jun Wei, and Dan Ye. Semantic table structure identification in spreadsheets. In Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, pages 283–295, 2021.

[18]- Justin Smith, Justin A Middleton, and Nicholas A Kraft. Spreadsheet practices and challenges in a large multinational conglomerate. In 2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pages 155–163. IEEE, 2017.

[19]- Margaret Burnett, Andrei Sheretov, Bing Ren, and Gregg Rothermel. Testing homogeneous spreadsheet grids with the" what you see is what you test" methodology. IEEE Transactions on Software Engineering, 28(6):576–594, 2002.

[20]- Robin Abraham and Martin Erwig. Autotest: A tool for automatic test case generation in spreadsheets. In Visual Languages and Human-Centric Computing (VL/HCC'06), pages 43–50. IEEE, 2006.

[21]- Marc Fisher, Gregg Rothermel, Darren Brown, Mingming Cao, Curtis Cook, and Margaret Burnett. Integrating automated test generation into the wysiwyt spreadsheet testing methodology. ACM Transactions on Software Engineering and Methodology (TOSEM), 15(2):150–194, 2006.

[22]- Sohon Roy, Arie Van Deursen, and Felienne Hermans. On the effectiveness of automatically inferred invariants in detecting regression faults in spreadsheets. In 2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), pages 199–206. IEEE, 2018.