An AI-based Interactive Tool for Spreadsheet Debugging

Thomas Schmitz
TU Dortmund
44221 Dortmund, Germany
thomas.schmitz@udo.edu

Dietmar Jannach
TU Dortmund
44221 Dortmund, Germany
dietmar.jannach@udo.edu

***Abstract*—Several cases are known where faults in spreadsheets have caused severe losses of money for companies. Besides other factors, the non-existence of advanced testing and debugging mechanisms in environments like MS Excel causes faults in spreadsheets to remain undetected. In this paper we describe the main functionality of the EXQUISITE tool, a software system and add-in to MS Excel that was developed in the context of our research on designing next-generation mechanisms for spreadsheet testing and debugging based on artificial intelligence technology. The tool in particular supports a novel algorithmic debugging approach and was successfully validated through different user studies.**

I. Introduction

Spreadsheets are probably the most successful form of end-user development tools. They are widely used in all areas of industry and often serve as a basis for important decision making processes. However, spreadsheets are prone to error and several cases are known where faults in spreadsheets have caused a severe loss of money. In the academic literature many advanced approaches were proposed over the years to support users when testing and debugging their spreadsheets in order to detect and remove such faults [1]–[4]. Only few ideas however made their way into commercial tools like MS Excel, which still has no built-in support, e.g., to manage test cases. Other academic approaches were mainly evaluated with prototypes that are not well integrated in commercial tools.

In this paper we describe a test and debugging add-in to MS Excel called EXQUISITE, which we have successfully validated through simulation experiments and user studies in the context of different research projects over the last few years [5]–[7]. The purpose of the add-in is to help users locate (and fix) faults in a given spreadsheet. The technical innovation of our tool lies in a novel algorithmic debugging method that is based on the principles of a domain-independent Artificial Intelligence based diagnostic reasoning approach called Model-Based Diagnosis [8]. Based on user-specified test cases, this debugging method can automatically identify which formulas in a given spreadsheet are potentially faulty and cause the observed calculation outcomes to be different from the expected ones. Based on this information, spreadsheet users can then focus their attention during debugging on this subset of the formulas.

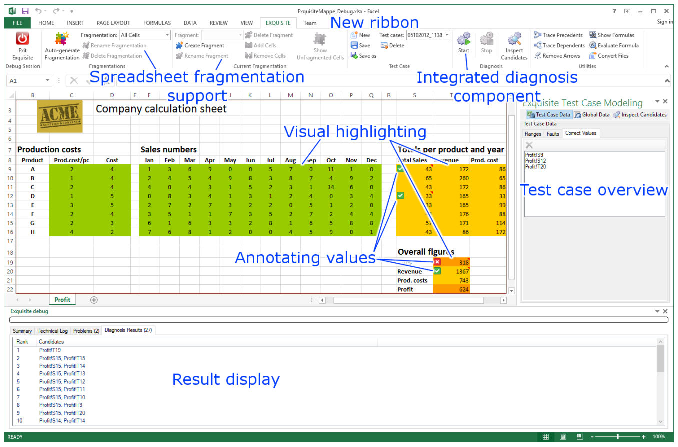

Fig. 1: Components of the EXQUISITE debugging add-in.

II. Debugging Workflow

The user interface of EXQUISITE is embedded within MS Excel. The main components of the add-in are shown in Figure 1. The workflow for the end user is as follows. Once the user detects that some outputs of a spreadsheet are different from the expected ones, he or she can switch to the “Debugging” view as shown in the figure. In the debugging mode, the user stays within the used environment, but is provided with additional functionality, e.g., in the ribbon of the add-in. As a first simple means to support the debugging process we visually highlight the different types of cells (input, intermediate, and output cells), as some types of errors like reference errors can be easily identified by such a visualization.

In the general case, we assume that users incrementally develop spreadsheets with some input values. This means that we are always given at least one test case, and in case of unexpected calculation outcomes, the task for the user is to mark those cells that do not contain the desired outputs and to provide the expected values for them. In addition, users can mark individual calculated values or formulas as being definitely correct. The user can then initiate the automated fault localization (diagnosis) procedure of EXQUISITE which computes sets of formulas that could have caused the unexpected outcomes. The list of (combinations of) possibly faulty cells is displayed at the lower part of the screen and can be ordered by fault probabilities, e.g., the formula complexities.

Three different methods can then be used to find the true reason of the fault. The user can (a) inspect the formulas of the suspicious cells or (b) provide additional test cases (i.e., sets of inputs and annotations about unexpected behavior) to further narrow down the set of potential causes. Another option for the user is to (c) interactively answer a set of questions determined by the system about the correctness of the formulas, see [9]. The questions are determined in a way to find the true reason of the fault with as few interactions as possible regardless of the user's answers. In a recent work we presented a new algorithmic approach to efficiently determine such questions and showed that it was able to quickly determine the questions even for complex systems [10].

To validate that our plug-in is helpful and usable for end users, we conducted different user studies. In [5], for example, we report the results of a between-subjects user study, where the task of the participants was to locate a fault in a given spreadsheet. We compared the efficiency and the effectiveness of the participants when accomplishing this task with our tool and without it. The results showed that users with tool support located the fault faster than those without. Furthermore, a number of participants without tool support did not find the injected fault at all in the given time frame.

III. Supporting the Specification of Test Cases

Generally, the ability of the debugging method to narrow down the set of potentially faulty cells that should be inspected by the user increases when more test cases are provided. Therefore, our add-in allows the user to specify several test cases with different inputs, which we store inside the spreadsheet (.xlsx file) using Excel's built-in extension mechanisms. The functionality of providing multiple test cases is entirely missing in MS Excel. Currently, we use the stored test cases only in the context of the debugging approach. In the future, these test cases could also be used to perform regression tests after maintenance activities.

Providing such test cases can easily become tedious for large spreadsheets, as the user potentially has to manually check the outcomes of longer calculation chains in order to validate the correctness of certain calculations. The EXQUISITE system therefore implements a novel problem decomposition strategy proposed in [6] and [7], which is based on the idea of splitting up a spreadsheet into smaller semantically connected chunks, which we call fragments. Instead of asking users to provide test cases and annotations for the entire spreadsheet, the idea is to let them specify test cases for these smaller fragments that usually are of lower complexity, e.g., because the calculation chains are shorter. These fragment-based test cases are then fed – possibly in combination with test cases for the entire spreadsheet – into our algorithmic debugging method, which is capable of dealing with such partial test cases. Note that the fragmentation of a given spreadsheet can be either specified manually within our tool, or they can be calculated automatically based on heuristics. In [6], we proposed a corresponding algorithm based on an evolutionary algorithm to automatically decompose a given spreadsheet.

First experimental evaluations in [6] showed that partial test cases can help to improve the fault localization process. Furthermore, in a recent user study, we could validate that the debugging process is more efficient when partial test cases based on a fragmentation of the spreadsheet are used.

IV. Technical Architecture

The EXQUISITE tool is developed as a client-server application where the computationally costly process of finding fault candidates in larger spreadsheets can be done either on the local machine or on a remote server. This decoupling in principle also allows us to implement add-ins to other spreadsheet environments and reuse the algorithmic debugging and fragment computation logic. The algorithmic debugging method itself is based on translating the spreadsheet program into a constraint satisfaction problem [5]. The internal inference method is then based on a single-threaded or parallelized version of the "HS-Tree" algorithm as described in [11].

V. Summary

Our tool currently is in the state of a research prototype but has been successfully validated on a number of real-world spreadsheets. While a number of future steps are required to implement the tool's functionality in a commercial tool, our add-in has already shown the general feasibility of integrating better mechanisms for fault localization and avoidance into modern spreadsheet environments. The tool will be presented at the conference as a live demo to stimulate discussions about possible improvements and perspectives for future research.

Acknowledgment

The work was funded by the Austrian Science Fund (FWF, contract I2144) and the German Research Foundation (DFG, contract JA 2095/4-1).

References

[1] D. Jannach, T. Schmitz, B. Hofer, and F. Wotawa, "Avoiding, finding and fixing spreadsheet errors - A survey of automated approaches for spreadsheet QA," *J. of Syst. and Softw.*, vol. 94, pp. 129–150, 2014.

[2] R. Abraham and M. Erwig, "Goal-Directed Debugging of Spreadsheets," in *Proc. VL/HCC '05*, 2005, pp. 37–44.

[3] J. Cunha, J. P. Fernandes, J. Mendes, and J. Saraiva, "MDSheet: A Framework for Model-driven Spreadsheet Engineering," in *Proc. ICSE '12*, 2012, pp. 1395–1398.

[4] F. Hermans, M. Pinzger, and A. van Deursen, "Detecting Code Smells in Spreadsheet Formulas," in *Proc. ICSM '12*, 2012, pp. 409–418.

[5] D. Jannach and T. Schmitz, "Model-based diagnosis of spreadsheet programs: A constraint-based debugging approach," *Autom. Softw. Eng.*, vol. 23, no. 1, pp. 105–144, 2016.

[6] T. Schmitz, B. Hofer, D. Jannach, and F. Wotawa, "Fragment-Based Diagnosis of Spreadsheets," in *Proc. SEMS '16*, 2016, pp. 372–387.

[7] T. Schmitz, D. Jannach, B. Hofer, P. Koch, K. Schekotihin, and F. Wotawa, "A Decomposition-Based Approach to Spreadsheet Testing and Debugging," in *Proc. VL/HCC '17*, 2017.

[8] R. Reiter, "A Theory of Diagnosis from First Principles," *Artif. Intell.*, vol. 32, no. 1, pp. 57–95, 1987.

[9] D. Jannach, T. Schmitz, and K. Shchekotykhin, "Toward Interactive Spreadsheet Debugging," in *Proc. SEMS '14*, 2014.

[10] K. Shchekotykhin, T. Schmitz, and D. Jannach, "Efficient Sequential Model-Based Fault-Localization with Partial Diagnoses," in *Proc. IJCAI '16*, 2016, pp. 1251–1257.

[11] D. Jannach, T. Schmitz, and K. Shchekotykhin, "Parallel Model-Based Diagnosis on Multi-Core Computers," *J. of Artif. Intell. Res.*, vol. 55, pp. 835–887, 2016.