xVizIT: Visualizing Cognitive Units in Spreadsheets

Karin Hodnigg
Software Engineering Research Group
University of Klagenfurt, Austria
Email: karin.hodnigg@gmail.com

Martin Pinzger
Software Engineering Research Group
University of Klagenfurt, Austria
Email: martin.pinzger@aau.at

***Abstract*—Spreadsheets can be large and complex and their maintenance and comprehension difficult to end-users. Large numbers of cells, complex formulae and missing documentation can impede the understanding of a spreadsheet. Comprehension assesses different levels of a spreadsheet according to a specific maintenance task, ranging from single formulae over sets of cells to complex structural patterns. These levels of abstraction are subsumed under the term cognitive unit. xVizIT helps end-users in maintaining and comprehending spreadsheets. It guides them through a spreadsheet model: Roles of cells and sheets, similar patterns and various concepts of modularity can be explored. It uses modularization algorithms to provide conceptional decompositions of a spreadsheet model, such as equivalence classes or data modules. xVizIT's slice visualizations ease the evaluation of corrective modifications by showing the dependant cells. Furthermore, xVizIT provides a number of complexity measures allowing end-users to estimate the effort to comprehend and maintain a spreadsheet.**

I. Introduction

Creating a spreadsheet is in fact programming. This is already widely accepted among the end-user community. But while the requirements on quality and towards longevity of a spreadsheet are similar to common software built by professionals, the training of the end-user is not. End-users are domain experts, specialists in their fields, who are able to express their complex domain models. But, often they have only little knowledge in testing, quality assurance, or the software maintenance process itself - their core competencies are different by nature. Supporting them in their enterprise to understand and maintain often huge spreadsheet models must thus refrain from being "academic" or too technical.

Approaching the task of maintaining and extending existing and often huge spreadsheets, multiple factors have to be considered. Generally, complexity of a software directly affects the effort to comprehend it (a), the paradigm influences how problems can be solved and represented (b) and (c) the expertise of the end-user is important when it comes to estimating the effort put into full comprehension of a given sheet.

Understanding thousands of cells and formulae individually is not feasible - and current tools, such as Excel and Numbers, provide only insufficient means to assess some kind of abstraction in spreadsheets. However, spreadsheet creation works in favour of abstraction: Repetition and copying formulae is an inherent part of spreadsheet generation, resulting in patterns of formulae [1]. Data dependencies allow a different analysis isolating independent or interdependent computations. Using these (and other) premises, we can find structures in a spreadsheet, and by sophisticated aggregation provide different levels of abstractions [2].

Starting from a global level, a *workbook* consists of different *worksheets*. A worksheet is a matrix consisting of *cells*, every cell can contain either a value or a formula. These levels are superimposed by the mental model in the spreadsheet, reused formulae, tabular organisations, layout information - they form another, more accessible hierarchy of abstractions. The term **cognitive unit** is coined to express the level of abstraction the user tries to understand - be it on a cell level, trying to comprehend the formula, or a larger level, trying to understand repetitive patterns or generic modules in a spreadsheet model. Users seamlessly switch between different levels of abstraction in an opportunistic, "as-needed" approach [3]. Understanding the meaning of a formula and then the meaning of its repetitive use in a worksheet are such levels.

xVizIT addresses the issue of missing abstraction and hidden model information and supports the spreadsheet user in providing answers to the following questions:

- **[Q1]** What are label, input or output cells?
- **[Q2]** Are there similar formulae? Are there modules or self-contained computations?
- **[Q3]** Where does the data for this particular cell come from? If I were to change a cell, what would be affected?
- **[Q4]** How complex is the model? How complex are single formulae?

xVizIT uses a spreadsheet-like view to show our visualizations. In the following, we present several usage scenarios in which our tool is used to provide answers to the questions stated before.

II. Usage Scenario

When it comes to spreadsheets and their maintenance, common questions are: *"The quarterly results in* `Summary!D14` *seems odd, can you take a look a that?", "Oh, we need another monthly report showing the results in $ rather than percent in our company report model - can you add that?", "I am afraid, no-one understands Alice's grading sheet, can you prepare a short demo on how to use it for the other teachers?"*.

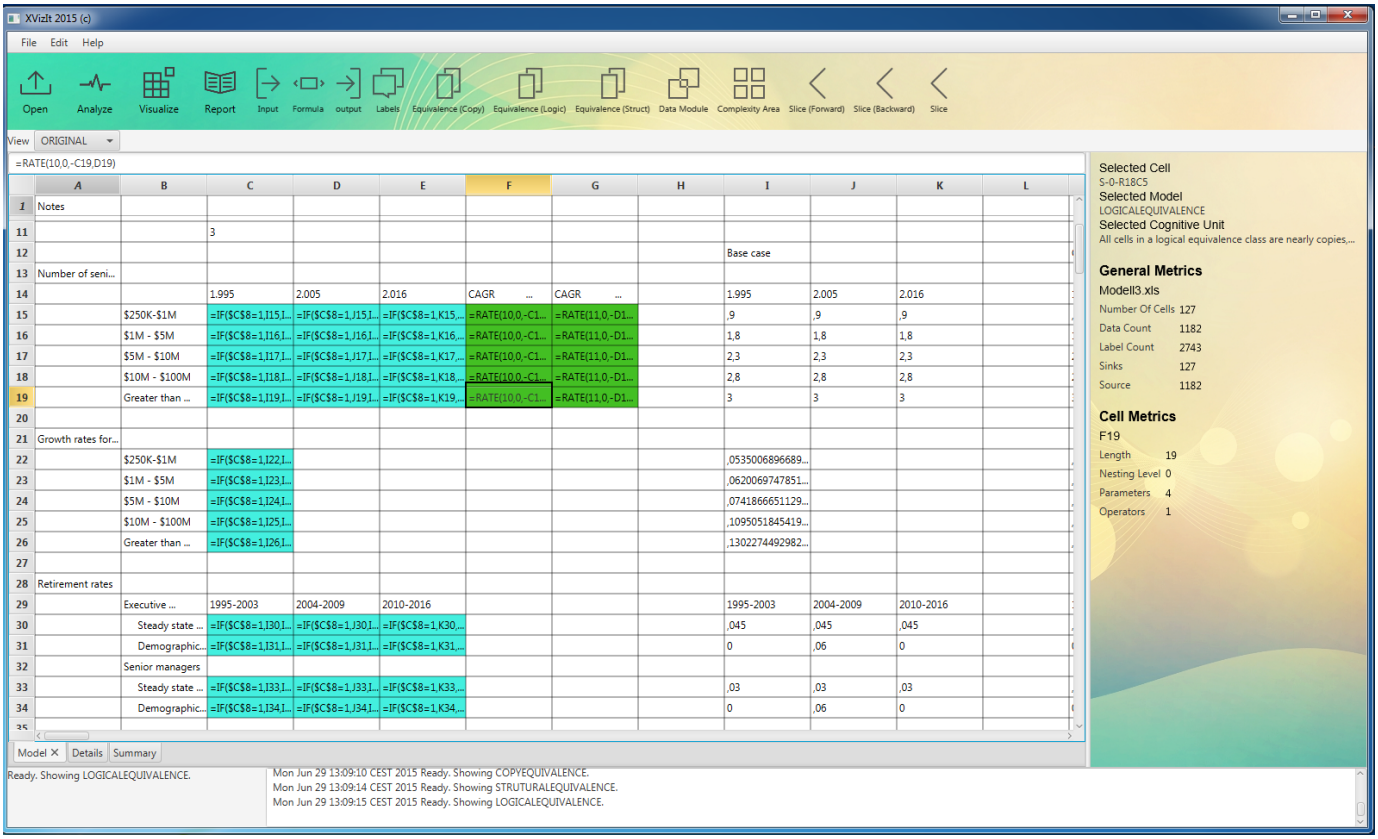

Fig. 1. XVIZIT - highlighting equivalence formulae and showing basic size and complexity metrics for cognitive units.

These questions all target different levels of comprehension and maintenance: the first one implies a data dependency inspection (where does the data come from?), the second requires a deep understanding of worksheet structures and data knowledge, and the third is the most advanced (so that replication and extension does not break the model's design) requiring the provision of documentation and instructions on a legacy spreadsheet.

There is a difference whether a user performs perfective or corrective maintenance tasks. While the latter is likely to be precise and local, the first will need a deeper understanding of the spreadsheet's global structure and already available (intermediate) results in order not to break the already established model. To support both localized and global assessment of spreadsheet models is the goal of XVIZIT.

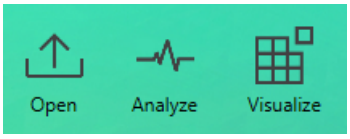

Fig. 2. XVIZIT wizard

XVIZIT offers a guided process in form of a wizard to analyze a spreadsheet. This guidance allows the explanation of every single process step and the interaction on different cognitive levels as seen in Fig. 2 (this is why `Open` and `Analyze` are separated).

With `Open` the spreadsheet in question is opened and a primary analysis (parsing the sheet and mapping it to XVIZIT's internal data structures) is performed. If there are issues opening the spreadsheet, the status window provides detailed feedback. If the spreadsheet can be opened successfully, the spreadsheet's representation is rendered in the main window as depicted in Fig. 1. XVIZIT offers a mini-spreadsheet environment native to the user and more likely to be accepted.[1]

After the spreadsheet is successfully loaded, a fundamental analysis can be performed with `Analyze`. This analysis includes computation of complexity metrics, fundamental roles of cells, and the similarity of formulae. `Visualize` opens a preliminary and limited data dependency graph, that intends to show the complexity and connected components.

A. *[Q1] Fundamental Analysis*

Let us assume, Bob, a novice in spreadsheet modeling, is assigned to comprehend and document Alice's spreadsheet depicted in Fig. 1. He is required to understand what is computed, where and what the model looks like. XVIZIT helps Bob to approach such a fundamental analysis, by highlighting

[1]At the moment, this is our definition of seamless. When the prototype proves to be helpful, a truly seamless integration into a spreadsheet environment will be performed.

all formula cells and value cells separately. Value cells, however, can serve two purposes - either descriptions or real input data.

Currently, XVIZIT distinguishes between the following cell types:

- **Input** All cells that are referenced, but contain no dependencies (might also be self-contained formula cells)
- **Formula** All cells that contain a formula.
- **Output** All cells that contain a formula but are not referenced (formula cells)
- **Label** Cells that are not referenced and do not contain a formula (value cells)

Fig. 1 shows the corresponding buttons (`Input`, `Formula`, `Output`, `Label`) to highlight these cells. By selecting them, XVIZIT highlights the corresponding cells in different colors in the main spreadsheet table. This allows Bob to distinguish label from data from formula cells and answer the first question stated in the introduction.

Moreover, switching between value and formula view is made very easy in XVIZIT. Choosing the corresponding value in the combo box below the wizard toolbar in Fig. 1 leads to an immediate update of the cell contents in the spreadsheet table. Users are thus encouraged to switch between these views in a maintenance process - allowing them to stay focused while navigating the spreadsheet model. Having identified output cells with XVIZIT, Bob now would like to know what the formulas are, if there are many different ones or if there are complex computations.

B. [Q2] Formula Patterns

Spreadsheet creation is inherently and massively based on repetition and copying formulae, resulting in patterns of formula reuse. This often indicates a common computational idea that can be identified; and provides a first abstraction to the potentially huge number of formula cells. Formulae that are equal (or similar) can be seen as one cognitive unit.

According to Mittermeir and Clermont's definition [1], [4] the concept of equivalence can have varying degrees of stringency, ranging from equality over similarity to some kind of relatedness. The one equivalence, that we (and Bob) are mainly interested in, is the equality - formulae resulting from click-and-drag operations during spreadsheet creation. With the `Equivalence` buttons, Bob can highlight similar (or equal) cells in one color. He then can have a look at the formula itself by switching from the value to the formula view.

Interestingly, a visual assessment can highlight issues in the spreadsheet model - disruptions in an otherwise even color pattern indicate disruptions in the spreadsheet model. Otherwise, the pattern highlights cells that convey the same computation in the same color.

C. [Q3] Slices and Dynamic Analysis

The two most important questions that arise during corrective maintenance are, "where does my data come from?" and "what does my change impact?". With that, we leave a more global view of spreadsheet analysis and move into localized visualizations that focus on the vicinity of a cell (or a module) in question.

Hidden models and dependencies are the main obstacle in spreadsheet comprehension. In [5], Ko states that *syntax has long been a significant learning barrier . . . [, largely because of the difficulty of understanding and remembering the hidden and complex rules encoded in language grammars].* Furthermore, the simplicity of the spreadsheet generation (immediate evaluation of formulae, hidden dependencies, copy-and-paste) causes additional cognitive load when trying to revert the process and extract the cognitive model behind the spreadsheet numbers. With Walenstein's observation, that *the dimension of hidden dependencies suggests that if important dependencies are not made explicit, by the system, then they can be missed, creating errors during modification* [6], we learn that precautions must be taken.

Eve needs to analyze a given (for the sake of space clearly arranged) spreadsheet model and see if the invoice value is correct. Using XVIZIT, she selects the corresponding cell in Fig. 3 and selects the slice visualization in the wizard toolbar. Starting from $E8$, all cells that lead to the result are highlighted in red. We use different shades of reds to indicate the different levels in the computed slice.

	A	B	C	D	E
1					
2	Amount	Price	Product	Sum	VAT
3	5	1	Pencils	=A3*B3	=D3*1.2
4	3	7	Notbooks	=A4*B4	=D4*1.2
5	1	2	Marker	=A5*B5	=D5*1.2
6	7	1	Post-It	=A6*B6	=D6*1.2
7	2	5	Paper	=A7*B7	=D7*1.2
8					=SUM(E3:E7)

Fig. 3. XVIZIT slice representation of cell E8

Respectively, impact analysis can be done with the impact slice: if Eve would change a formula or input data, XVIZIT helps her in highlighting dependent cells as represented with the blue cells in Fig. 4.

The visualization of hidden dependencies has been proven to be very useful, with Excel offering handy tracing mechanisms. Yet, they suffer from some limitations, as they discontinue tracing precedents beyond the active worksheet. In that case, only a dialog with the direct precedents on other worksheets is shown while transitive dependencies are not indicated further. Here, XVIZIT's slicing visualization allows the user to switch between sheets and analyze the impact or dependencies uninterruptedly. This is especially important when it comes to comprehending impact in large spreadsheets. Coloring dependant or contributing cells is a powerful visualization that allows the user to specifically focus on relevant cells.

	A	B	C	D	E
1					
2	Amount	Price	Product	Sum	VAT
3	5	1	Pencils	=A3*B3	=D3*1.2
4		7	Notbooks	=A4*B4	=D4*1.2
5	1	2	Marker	=A5*B5	=D5*1.2
6	7	1	Post-It	=A6*B6	=D6*1.2
7	2	5	Paper	=A7*B7	=D7*1.2
8					=SUM(E3:E7)
9					

Fig. 4. XVIZIT impact slice represenation of cell A4

D. [Q4] Complexity Assessment

Another important question when it comes to maintain spreadsheets is "how long will a given maintenance task take?". Although the experience of the spreadsheet user influences the answer, common metrics such as the size of the spreadsheet (number of formulae, number of input and output cells) or its complexity (formula complexity, number of similar formulae and possible modularization) are useful indicators. These metrics are based on previous work discussed in [3]. Although the complexity of formulae is difficult for end-users to assess and depends on their individual experience as Hermans showed in [7], it is crucial in estimating efforts.

XVIZIT offers complexity metrics on different abstraction levels - when a cell, a worksheet or any cognitive unit is selected, the respective complexity assessment is shown in an information panel. Fig. 1 shows examples of general complexity measures (size, labels, sinks) that describe the model's general complexity. Selecting a cell (F19) in the table results in highlighting the internal complexity measures and their presentation in the right information panel.

III. Architecture and Design

An overview of XVIZIT's architecture is depicted in Figure 5[2]. The basic information is generated in a module XSEED that takes care of the internal representation of the spreadsheet model. First, the three dimensional matrix representation is parsed and filled, the first formula complexity metrics are evaluated, then the representation as hypergraph is deduced from an analysis of the given references. The hypergraph representation allows the application of common graph algorithms and a further evaluation of the spreadsheet model complexity. With these representations, cognitive unit abstractions can be determined. Fundamentals include the input, output, formula and label cell highlighting. Different cognitive unit models result in different decompositions of the spreadsheet.

The GUI is based on the MVP-Pattern (model-view-presenter), that allows a generic, yet distinct and maintainable layer of implementation. If the representation needs to be extended or changed, new visualizations should be introduced, the MVP implementation provides a stable framework for that.

Because XVIZIT is an early prototype with a considerable number of planned extensions, a lot of thought has been put into the extensibility of the application. The central spreadsheet model ("Internal Representation") is represented as and can be accessed two-fold, (a) as a matrix and (b) as a hypergraph. This design facilitates the future extension with new metrics, slicing algorithms, and visualizations.

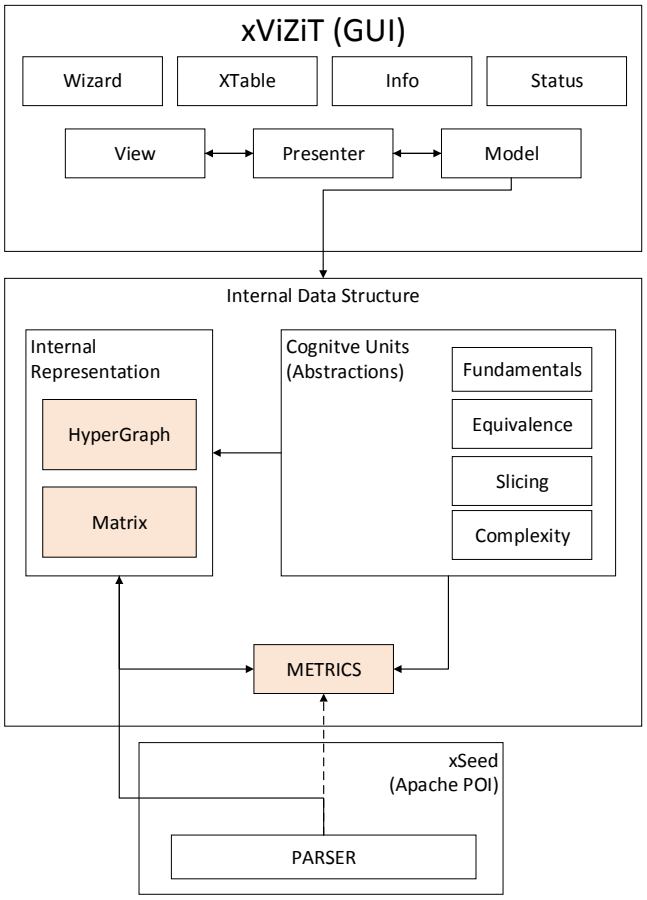

Fig. 5. Modular architecture and basic data flow of XVIZIT

We maintain a blog concerning our progress on spreadsheet visualization[3] where XVIZIT can be downloaded. Feedback and bug reports are very welcome.

[2]XVIZIT has been implemented in JAVA 8, using JAVAFX 8 and CONTROLS FX 8.20.8 to render the GUI [http://fxexperience.com/controlsfx/, 2015-06-20]. Parsing capabilities are included with the APACHE POI 3.10.1 library, that conveniently allows opening spreadsheets [https://poi.apache.org/, 2015-06-20]. As XVIZIT needs a less rich, more fundamental representation to comfortably compute metrics or generate visual representations, the POI framework is solely used to parse the model into internal data structures. A simple graph visualization is implemented using GRAPHSTREAM 1.3 [http://graphstream-project.org/, 2015-06-20] GraphStream is a library offering fundamental graph algorithms and comes with an easy-to-use viewer. However, GraphStream is not capable of handling hypergraph structures as are needed in XVIZIT, so we provided our own generic implementation of a hypergraph.

[3]https://xvizit.wordpress.com/

IV. Key Related Work

We based the tool on various observations and want to start a journey in experimenting and studying structural visualizations of spreadsheet abstractions. Thus, the work of Igaraishi and Tukiainen is fundamental to us. In 1998, Igarashi proposed animated visualizations of spreadsheet dependencies [8]. Tukiainen tested different conceptual levels against error proneness in novice users' models [9]. His conclusion that a low conceptual level correlated with high error rates formed the basis for our abstraction model. These abstractions are based on semantic classes and data modules [4], [1], but are extended by the visualizations in [3]. Hermans discussed the advantages and pitfalls of data flow visualization in [10], where she used different forms of abstractions and dataflow between these modules to represent a high level representation of a spreadsheet model. With an approach of finding data clones, she goes beyond a mere structural approach and inspects the risks of duplication of data in spreadsheets [11].

Although Vemuri found data dependency diagrams to be ineffective in end-user support [12] due to a cognitive gap between the dataflow and the spreadsheet model, a more immediate dependency based slicing is a promising approach. Slicing and comprehension, as discussed by Rilling in [13], is translated into a spreadsheet world. The concept of complexity is introduced to highlight potential pitfalls, or hot-spots, in a model. Here Bregar's complexities [14] have been influential, though were extended in [3]. In [7], Hermans addressed the specific issue of formula complexity in a study.

Further relevant tool research can be found in [5], [15], where Ko et.al. provide an overview of state-of-the-art tool support for end-users.

V. Caveats and Major Limitations

Our spreadsheet visualization approach is - at the moment - purely based on fundamental spreadsheet structures. Procedural extensions, such as VBA, are currently left out. We are aware that this might be an issue due to the extensive use of macros in spreadsheets, however with macros we leave the spreadsheet paradigm and enter a procedural world. We assume that for such spreadsheets containing macros the maintenance is executed by programming experts that are already familiar with other software engineering techniques. Regarding the structural analysis of spreadsheets, Apache POI and consequently xVizIT currently cannot handle matrix formulae that are defined as `"{=SUM...}"`. Furthermore, xVizIT currently is a read-only visualization tool: spreadsheets cannot be edited in the tool environment.

VI. Conclusion and Outlook

We presented a simple, yet powerful static and dynamic visual representations with xVizIT to ease and guide the end user during an extensive comprehension and maintenance process. Although it is difficult to reconstruct the original mental model, it is possible to allow the user a guided tour through the spreadsheet program when maintaining it. xVizIT provides different abstractions (cognitive units) on different levels of comprehension. Opportunistic approaches are supported by providing appropriate levels of abstraction and providing the apt view onto the spreadsheet. Complexity visualization allows the user to understand how much effort is to be expected when maintaining and understanding spreadsheets.

One of the main obstacles to spreadsheet comprehension is the worksheet barrier - visualizations often rely on one worksheet and then require the user to switch to another sheet. To address this issue, we plan to investigate three-dimensional visualizations of the internal model, such as a spreadsheet city [3]. xVizIT is a work in progress. In case you want to follow the implementation process, please feel free to visit the xVizIT blog at https://xvizit.wordpress.com/.

References

[1] M. Clermont, "A scalable approach to spreadsheet visualization," Ph.D. dissertation, University of Klagenfurt, 2003.

[2] K. Hodnigg, M. Clermont, and R. Mittermeir, "Computational models of spreadsheet devlopment: Basis for educational approaches," *Proceedings of the EuSpRIG Annual Conference: "Risk Reduction in End User Computing"*, vol. 5, pp. p.153–168, 2004.

[3] K. Hodnigg and R. T. Mittermeir, "Metrics-based spreadsheet visualization: Support for focused maintenance," *Proceedings of the EuSpRIG Annual Conference: "Enterprise Spreadsheet Management: A necessary Evil?"*, vol. 9, pp. 79 – 94, 2008.

[4] R. Mittermeir and M. Clermont, "Finding high-level structures in spreadsheet programs," *Proceedings of the Ninth Working Conference on Reverse Engineering (WCRE'02)*, pp. 221 – 232, 2002.

[5] A. J. Ko, B. A. Myers, M. L. Coblenz, and J. Stylos, "End-user programming productivity tools," *The Next Step: From End-User Programming to End-User Software Engineering, Proceedings of the WEUSE II Workshop at CHI*, pp. 30 – 32, 2006.

[6] A. Walenstein, "Developing the designer's toolkit with software comprehension models," *In Proceedings of the 13th IEEE International Conference on Automated Software Engineering*, pp. 310–313, 1998.

[7] F. Hermans, M. Pinzger, and A. van Deursen, "Measuring spreadsheet formula understandability," *Proceedings of the 2012 Annual Conference: "The Science of Spreadsheet Risk Management"*, pp. 77 – 96, 2012.

[8] T. Igarashi, J. Mackinlay, B.-W. Chang, and P. Zellweger, "Fluid visualization of spreadsheet structures," *Proceedings of the 1998 IEEE Symposium on Visual Languages*, pp. 118 – 125, 1998.

[9] M. Tukiainen, "Comparing two spreadsheet calculation paradigms: An empirical study with novice users," *Interacting with Computers (Elsevier)*, vol. 13, pp. 427 – 446, 2001.

[10] F. Hermans, M. Pinzger, and A. van Deursen, "Breviz: Visualizing spreadsheets using dataflow diagrams," *Proceedings of the EuSpRIG 2011 Annual Conference: "Spreadsheet Governance - Policy and Practice"*, 2011.

[11] F. Hermans, B. Sedee, M. Pinzger, and A. v. Deursen, "Data clone detection and visualization in spreadsheets," *Proceedings of the 2013 International Conference on Software Engineering*, pp. 292–301, 2013.

[12] S. Vemuri, S. Sengupta, and J. S. Davis, "Data dependency diagrams for spreadsheet applications," *Proceedings of the 30th annual Southeast regional conference*, pp. 467–470, 1992.

[13] J. Rilling and T. Klemola, "Identifying comprehension bottlenecks using program slicing and cognitive complexity metrics," *11th IEEE International Workshop on Program Comprehension*, pp. 115–124, May 2003.

[14] A. Bregar, "Complexity metrics for spreadsheet models," in *Proceedings of the EuSpRIG Annual Conference: "Risk Reduction in End User Computing"*, vol. 5, 2004.

[15] 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, and S. Wiedenbeck, "The state of the art in end-user software engineering," *ACM Comput. Surv.*, vol. 43, no. 3, pp. 21:1–21:44, Apr. 2011.