WARDER: Refining Cell Clustering for Effective Spreadsheet Defect Detection via Validity Properties

Da Li[†], Huiyan Wang[‡], Chang Xu[*§], Fengmin Shi[¶], Xiaoxing Ma[§], Jian Lu[§]
State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
Department of Computer Science and Technology, Nanjing University, Nanjing, China
[†]njulida@outlook.com, [‡]cocowhy1013@gmail.com, [¶]shifengmin@qq.com, [§]{changxu, xxm, lj}@nju.edu.cn

***Abstract*—Spreadsheets are widely used, but subject to various defects and severe consequences due to poor maintenance by end users. Existing spreadsheet defect detection techniques fall short of effectiveness, either due to limited scopes or relying on rigid patterns. In this paper, we discuss and improve one state-of-the-art technique, CUSTODES, which uses cell clustering and anomaly detection to extend its scope and make its patterns adaptive to varying spreadsheet styles, but is prone to fragile clustering when involving irrelevant cells, leading to a largely reduced detection precision. We present WARDER to refine CUSTODES's cell clustering based on validity properties, and experimental results show that WARDER improves the precision by 20.7% on average or reach 100% for 79.8% worksheets on cell clustering, which contributes to a precision improvement of 23.1% for defect detection. WARDER also exhibits satisfactory results, against other spreadsheet defect detection techniques, and on another large-scale spreadsheet corpus VEnron2.**

***Index Terms*—Spreadsheet testing, Cell clustering, Defect detection, Validity Property**

I. Introduction

Nowadays, spreadsheets have been widely used in data storage, financial analyses, and quality control [40], and become one of the most popular end-use environments with over 750 million users for representative Microsoft Excel alone [9]. In spite of the popularity, spreadsheets are found to be error-prone [37], and can cause catastrophic consequences, e.g., massive financial loss [1]. Such errors can spread across from data cells to formula cells in spreadsheets, and existing studies [36] have suggested that the latter could typically be the root causes. Within the scope of this paper, we name the errors in formula cells as *defects* in spreadsheets, and focus on effective techniques for their detection.

Detecting spreadsheet defects can be non-trivial. First, spreadsheets are typically maintained by non-programmer end-users, whose behaviors can involve various unprofessional operations, e.g., overwriting a formula with a plain value or replacing it with another plausible formula, simply for ad hoc purposes [36]. This results in a boosting of spreadsheet defects. Second, auditing or tracking services are typically unavailable for common spreadsheet usages [29], and this results in the loss of clues on how spreadsheet defects are introduced and where they are. Third, semantic relationships among spreadsheet cells are typically hidden, and this results in the difficulties in reasoning over spreadsheets for defects. To address these challenges, our research community has proposed various spreadsheet defect detection techniques. They rely on header information to infer type inconsistencies in formula references (e.g., UCheck [4] and Dimension [10]), exploit rectangle patterns to recognize missing or inconsistent formulas (e.g., AmCheck [14] and CACheck [15]), or use adaptive learning to detect anomalies in formula cells (e.g., CUSTODES [11], Melford [42], and ExceLint [8]).

However, these spreadsheet defect detection techniques still have their weaknesses. For the first category (type-based techniques), their inference is vague and focuses on a quite limited scope, resulting in both a low precision and recall rate [48]. For the second category (pattern-based techniques), their relied patterns can be strict and precise for certain features in spreadsheets (thus achieving a high precision, e.g., 71.9% for AmCheck and 86.8% for CACheck [15]), but not adaptable to varying styles in different spreadsheets (leading to a compromised recall rate, e.g., 60.3% for AmCheck and 71.0% for CACheck [15]). For the third category (learning-based techniques), they get increasingly popular in recent years due to their adaptive learning abilities. We take CUSTODES [11] for example, as it is considered to be the "best automated error finder" [8]. CUSTODES clusters cells according to their formula semantics, and at the same time restricts the impact caused by the dissimilarity of faulty formulas and varying styles across spreadsheets by learning features in different spreadsheets. By doing so, it largely increases the recall rate (80% [11]), but at the same time compromises the precision somewhat (65% [11]) when clustering irrelevant cells together.

Regarding these weaknesses, we in this work propose a novel technique WARDER[1], based on the success of CUSTODES's adaptive learning of varying styles (features) across spreadsheet, but improving over its weakness when clustering relevant and irrelevant cells together for defect detection. Our key observation is that the clustering, if accidentally involving irrelevant cells, would largely compromise the effectiveness of defect detection (e.g., reducing the precision). Therefore, our main efforts in WARDER focus on refining the cell clustering to make it more robust by squeezing out irrelevant cells, while preserving CUSTODES's original adaptive learning ability.

[*]Corresponding author.

[1]WARDER is named in a way following CUSTODES's naming tradition.

The refinement is three-folded: (1) *Single-cell validity.* When adding cells into a cluster, WARDER would reject those data cells that can become invalid if cast to formulas for unification. For example, when the data in a cell is replaced by a formula for unification with other formula cells in the same cluster, this formula is found to be invalid for calculation (e.g., citing a wrong place or causing a wrong reference). Then such a data cell should not be added into this cluster. (2) *Multi-cell validity.* WARDER would also reject those data cells from being added into a cluster if these cells, once added, would violate some important properties of existing cells in this cluster. For example, the references of existing cells in a cluster do not overlap before adding a new data cell, but this property would be violated if the cell is added and its contained data is replaced by a formula for unification. Then this new data cell should also not be added into this cluster. (3) *Whole-cluster validity.* Other than cell-level validations, WARDER also examines whole-cluster validity for thus formed clusters. Since each cluster is formed with the aim for identifying few anomalies as defects in spreadsheets, it should contain a formula unifiable with most cells in this cluster. Otherwise, the cluster itself is not qualified since it lacks a common computational semantics [15], and should be canceled to avoid later misbehavior in defect detection.

With these dedicated refinements, our WARDER exhibits clear merits when compared to its predecessor CUSTODES, as well as other popular spreadsheet defect detection techniques, including UCheck, Dimension, AmCheck, and CACheck. For example, regarding CUSTODES's own benchmark of 291 worksheets (basic units in spreadsheets) selected from the EUSES corpus [16], WARDER achieved a significant improvement, as compared to CUSTODES, on cell clustering by boosting 79.8% worksheets, either with a precision improvement of 0.3–94.6% (20.7% on average) or already reaching their upper precision limit (100%), with a small sacrifice of 2.4% on the average recall rate. For defect detection, WARDER's effective cell clustering contributed to its defect detection by achieving 23.1% precision improvement, as compared to CUSTODES. Combining the recall rate, this leads to an increase of F-measure from 0.71 to 0.79. WARDER also outperformed other spreadsheet defect detection techniques by an average precision of 87.8% and recall rate of 71.9%, against 0.5–72.4% and 0.1–68.4% for other techniques, respectively. Moreover, regarding a much larger-scale corpus VEnron2 [45], which contains 1,609 versioned groups refined from original 79,983 worksheets in the Enron corpus [21], WARDER also exhibited its unique superiority over other techniques (41.3% against 16.3–34.3% on the precision).

In summary, this paper makes the following contributions:

- We proposed WARDER, which refines CUSTODES's cell clustering by three validations on cell and cluster validity properties to improve the effectiveness of spreadsheet defect detection.
- We evaluated WARDER comprehensively with both existing benchmark spreadsheets and a large-scale spreadsheet corpus, and compared it with existing popular spreadsheet defect detection techniques.

The remainder of this paper is organized as follows. Section II introduces background knowledge on spreadsheet and its defect detection. Section III proposes our WARDER technique, built on CUSTODES. Section IV evaluates experimentally WARDER with practical spreadsheets and compares it with existing defect detection techniques. Finally, Section V discusses the related work in recent years, and Section VI concludes this paper.

II. Background

In this section, we introduce necessary background knowledge about spreadsheet and its defect detection.

A. Spreadsheet

Spreadsheet. A *spreadsheet* refers to a stand-alone spreadsheet file in a file system. For example, in Microsoft Excel, each opened Excel file is a spreadsheet, which is named like A.xls and B.xlsx.

Worksheet. A *worksheet* refers to a single sheet page inside a spreadsheet. Normally, a spreadsheet can contain multiple worksheets separated for different data storage and calculation purposes.

Cell. A worksheet can contain multiple cells, each of which is referred to by a column number (e.g., A, B, and C) and a row number (e.g., 1, 2, and 3). A *cell* is the minimal information piece in a worksheet, which can contain a (numeric) data[2] (e.g., 200), formula (e.g., A1 + B2), or text string (e.g., "Fruit" or "----") for formatting purposes (e.g., as a table header or delimiter). In the scope of this paper, we are interested in the former two, as they are also the main focus of existing spreadsheet defect detection techniques, and in this case, they are referred to as a *data cell* and *formula cell*, respectively. A data cell contains a numeric data, which can serve as the input to other formula cells, and a formula cell contains a formula, which can automatically update its corresponding value as calculated from other (data or formula) cells whose values serve as the input to this formula.

Reference. References occur to formula cells. When the data contained in a data cell serves as the input to a formula cell, we say that this formula cell (or simply this formula) *references* this data cell and the latter is referred to as a *referenced cell*.

B. Defect Detection

As mentioned earlier, spreadsheets can contain various defects in their formula cells. We introduce two major defect types that are all covered by, and the main focus of, existing spreadsheet defect detection techniques.

Missing formula defect. This defect occurs when a cell contains a data instead of a formula, whose calculation is supposed towards this data value. For example, cell C3 was originally defined by formula A1 + B2. Due to some unknown

[2]Some papers may consider text strings also as data, i.e., their data cells can contain both numbers and text strings. With this clarified, slightly different definitions will not affect our subsequent discussions.

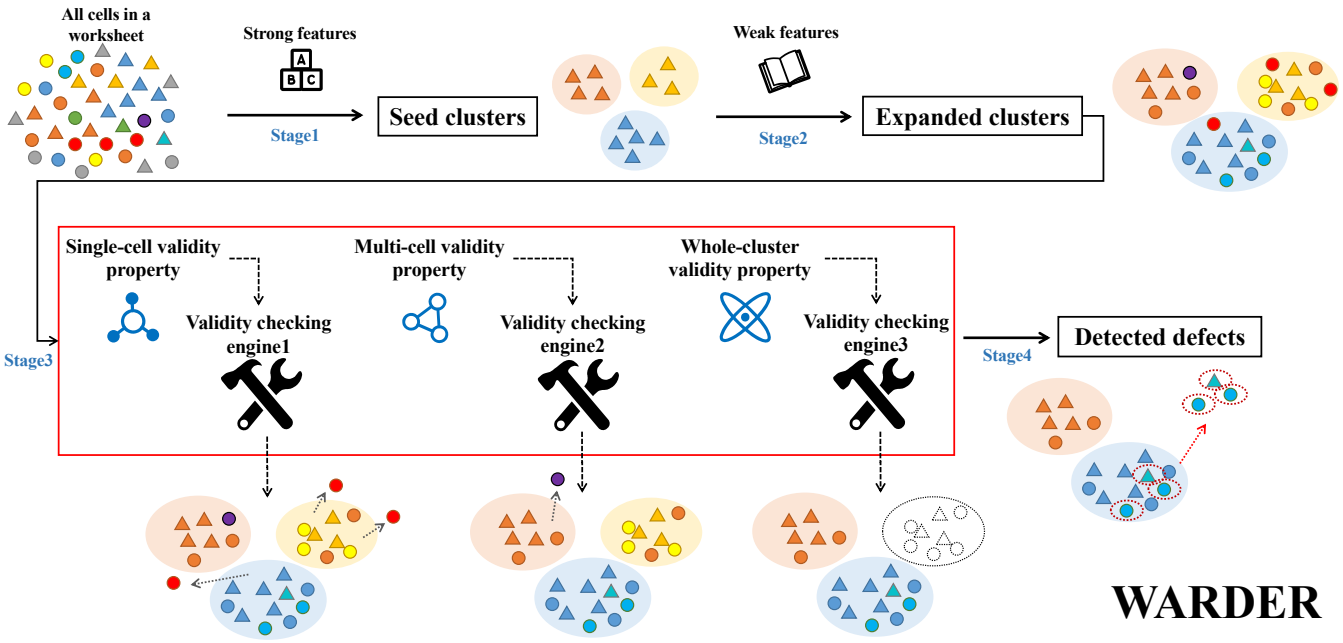


Fig. 1: Workflow of the WARDER technique (Stage3 is its main contributions over its predecessor CUSTODES)

reason, its user overwrote this formula by a plain value 5. It is possible that this formula happens to have this value as its calculation result, and in this case, this defect becomes hidden for now, but would be triggered later when cell A1 or B2 has its value updated.

Inconsistent formula defect. This defect occurs when a cell contains a formula different from those in its surrounding formula cells but in fact it should not. For example, a column (e.g., C) of cells is supposed to calculate the sum of its previous two columns (e.g., C1 = A1 + B1, C2 = A2 + B2, and so on). Suppose that one formula was mistakenly written as C2 = A2 − B2 or C2 = A2, but this defect can be hidden if cell B2 happens to contain a value of zero, although it would be triggered later when cell B2 has its value updated.

A spreadsheet defect may not immediately trigger any visible error in concerned data values, and therefore some pieces of work would consider them as *smells* or *anomalies* [23], depending on their severity levels. Nevertheless, detecting them for timely fixing is important, as spreadsheet users are not programming experts and not sensible to such hidden defects, which can easily grow into catastrophic financial losses in future.

III. WARDER Technique

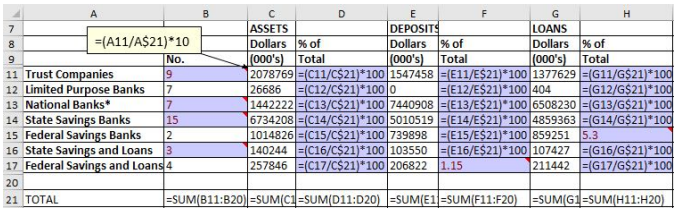
In this section, we propose and elaborate on our WARDER technique for effective spreadsheet defect detection, which is built on the CUSTODES technique. We first introduce WARDER's workflow, as well as its connections to CUSTODES. We then present WARDER's three key refinements for better cell clustering based on validity properties in detail.

A. WARDER's Workflow

As shown in Fig. 1, WARDER, with CUSTODES integrated, follows four stages to cluster relevant cells in a worksheet together and detect defects in each cluster.

First, WARDER uses CUSTODES to form an initial set of seed clusters that each contain cells of similar *strong features*, (e.g., cell formulas and reference relations). This stage is for each initial cluster to own a common computational semantics. Second, WARDER uses CUSTODES to expand each seed cluster with remaining cells that are left from the first stage, as long as these added cells share *weak features* (e.g., spatial relations, fonts, colors, and layouts) with those already in the cluster. This stage is for retrieving back those cells that were previously not put into seed clusters due to the impact of their contained faulty formulas or varying styles across tables. Third, WARDER refines the current clusters by squeezing out those cells from them, which violate the *validity properties* (i.e., single-cell, multi-cell, and whole-cluster ones) associated with these cell clusters. This stage is for improving the cell clustering by identifying irrelevant cells and unqualified clusters. Fourth, WARDER uses CUSTODES to classify anomalies in each qualified cell cluster, and report them as defects to users. This stage is for detecting defects (i.e., faulty cells that contain missing formula or inconsistent formula issues) in clusters of cells with common computational semantics.

CUSTODES is nice in retrieving back many cells into initial clusters in order to improve its recall rate in defect detection [11]. However, the retrieval is *aggressive* in that it could involve quite a few irrelevant cells, which instead impact the precisions of both cell clustering and defect detection for spreadsheets. This is also the target of our WARDER's three



	A	B	C	D	E	F	G	H
7		=(A11/A$21)*10	ASSETS		DEPOSITS		LOANS	
8			Dollars	% of	Dollars	% of	Dollars	% of
9		No.	(000's)	Total	(000's)	Total	(000's)	Total
11	Trust Companies	9	2078769	=(C11/C$21)*100	1547458	=(E11/E$21)*100	1377629	=(G11/G$21)*100
12	Limited Purpose Banks	7	26686	=(C12/C$21)*100	0	=(E12/E$21)*100	404	=(G12/G$21)*100
13	National Banks*	7	1442222	=(C13/C$21)*100	7440908	=(E13/E$21)*100	6508230	=(G13/G$21)*100
14	State Savings Banks	15	6734208	=(C14/C$21)*100	5010519	=(E14/E$21)*100	4859363	=(G14/G$21)*100
15	Federal Savings Banks	2	1014826	=(C15/C$21)*100	739898	=(E15/E$21)*100	859251	5.3
16	State Savings and Loans	3	140244	=(C16/C$21)*100	103550	=(E16/E$21)*100	107427	=(G16/G$21)*100
17	Federal Savings and Loans	4	257846	=(C17/C$21)*100	206822	1.15	211442	=(G17/G$21)*100
20								
21	TOTAL	=SUM(B11:B20)	=SUM(C1	=SUM(D11:D20)	=SUM(E1	=SUM(F11:F20)	=SUM(G1	=SUM(H11:H20)

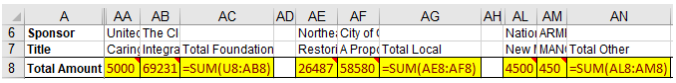
Fig. 2: Worksheet "Summary1201" for illustrating WARDER's single-cell validity refinement. There is one cluster marked in purple by CUSTODES, leading to four wrongly reported defects as annotated with red triangle marks, while these four cells would be squeezed out from the cluster by WARDER's single-cell validity refinement, and no longer be wrongly reported as defects.

refinements, as we elaborate in detail below.

B. Single-cell Validity Refinement

The first refinement concerns, when WARDER (following CUSTODES) expands initial seed clusters with additional data cells, whether such cells to add are valid themselves. Note that it may be difficult to tell whether these cells are valid or not directly, since they contain plain values only, without any visible relationship with other cells. Nevertheless, since these cells to add and original cells in the target cluster are to be merged together, they should share a common computational semantics according to the cell cluster definition. Then, these cells to add should be unifiable with some formula as those original cells. To validate this expectation, WARDER would test all formulas existing in the original cells in the cluster, to see whether any of them can fit in these cells to add. Here, "fit" means that such a formula, once replacing the plain value in a data cell to add, would still be computable. Otherwise, if all formulas are tested to be failed, e.g., citing a wrong place or causing a wrong reference, the data cell to add is problematic, and should be prevented from being added into this cluster. This is known as the *single-cell validity refinement*.

Fig. 2 gives one example by worksheet "Summary1201", in which 25 cells (B11, B13, B14, B16, D11–-17, F11–-17, and H11–-17) are clustered together (in purple) by CUSTODES. CUSTODES detected six defects (annotated with red triangle marks), in which two (F17 and H15) are true positives (missing formula defects) and the other four (B11, B13, B14, and B16) are false positives. The latter four data cells were retrieved into this cluster due to their shared weak features (e.g., similar headers and layouts) with original cells in the cluster by CUSTODES. However, such retrieval is problematic according to WARDER's single-cell validity refinement. In fact, if any of the four data cells is considered into the cluster, one has to unify its contained data with a formula unifiable with other cells in this cluster. For example, considering cell B11, the best candidate for its unifiable formula could be "=(A11/A$21)*100", following the pattern shared by other cells in this cluster. However, this formula is non-computable, as both A11 and A$21 refer to a text string, which cannot participate into any arithmetic calculation. Similar problems occur to cells B13, B14, and B16 as well. Therefore, WARDER would reject such data cells from being added into this cluster.



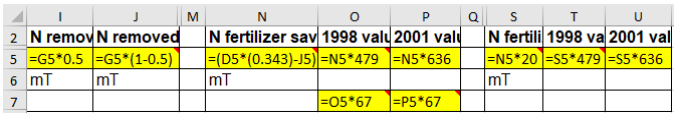
	A	AA	AB	AC	AD	AE	AF	AG	AH	AL	AM	AN
6	Sponsor	Unitec	The Cl			Northe:	City of (			Natio	ARMI	
7	Title	Carin(	Integra	Total Foundation		Restori	A Prop(	Total Local		New I	MAN	Total Other
8	Total Amount	5000	69231	=SUM(U8:AB8)		26487	58580	=SUM(AE8:AF8)		4500	450	=SUM(AL8:AM8)

Fig. 3: Worksheet "Detail for the College of A&S" for illustrating WARDER's multi-cell validity refinement. There is one cluster marked in yellow by CUSTODES, leading to six wrongly reported defects as annotated with red triangle marks, while these six cells would be squeezed out from the cluster by WARDER's multi-cell validity refinement, and no longer be wrongly reported as defects.

C. Multi-cell Validity Refinement

The second refinement concerns, when WARDER expands initial seed clusters with additional data cells, whether such cells to add will not break existing multi-cell properties of original cells in a cluster. We take references of a cell for example, which are an important feature of spreadsheet cells. Suppose that the references of original cells in a cluster never overlap with each other. Then we would expect that a data cell to add should also not violate this property, when it is added into this cluster and its contained data is replaced by a formula for unification with other cells in this cluster. This expectation can also be expressed in a reversed way, i.e., references, if already overlapping with each other for original cells in a cell cluster, should not encounter non-overlapping cases for new data cells to add. That is, the property should keep *consistent* for all cells in this cluster, and can be considered as an editing style across spreadsheet tables. Otherwise, the concerned data cell is considered problematic, and should be prevented from being added into this cluster. This is known as the *multi-cell validity refinement*.

Fig. 3 gives one example by worksheet "Detail for the College of A&S", in which 9 cells (AA8, AB8, AC8, AE8, AF8, AG8, AL8, AM8, and AN8) are clustered together (in yellow) by CUSTODES. CUSTODES then detected six defects (AA8, AB8, AE8, AF8, AL8, and AM8) simply due to their contained plain values (missing formula defects)), but all of them are false positives. On the other hand, WARDER would reject adding the six data cells (AA8, AB8, AE8, AF8, AL8, and AM8) into the cluster, and thus avoid detecting them as defects. In fact, the six data cells do not share a common computational semantics as the other three formula cells (AC8, AG8, and AN8). The former data cells refer to some specific values, which are directly from users, while the latter formula cells calculate the sums of several cells left to them. WARDER distinguishes them by its multi-cell validity refinement: the references of the latter three formula cells do not overlap, but this property would be violated if merging the former six data cells with them together and replacing the data of the former with any formula found in the latter cells. For example, when the data in cell AF8 is replaced by



	I	J	M	N	O	P	Q	S	T	U
2	N remov	N removed		N fertilizer sav	1998 valu	2001 valu		N fertili	1998 va	2001 val
5	=G5*0.5	=G5*(1-0.5)		=(D5*(0.343)-J5)	=N5*479	=N5*636		=N5*20	=S5*479	=S5*636
6	mT	mT		mT				mT		
7					=O5*67	=P5*67				

Fig. 4: Worksheet "World 1996" for illustrating WARDER's whole-cluster validity refinement. There is one cluster marked in yellow by CUSTODES (wrongly clustered), leading all the associated cells to be wrongly reported defects as annotated with red triangle marks, while this cluster would be removed by WARDER's whole-cluster validity refinement, and its associated cells would no longer be wrongly reported as defects.

a formula "SUM(AD8:AE8)" by following the pattern of cell AG8, its references (AD8 and AE8) would overlap with cell AG8's references (AE8 and AF8). Similar problems occur to cells AA8, AB8, AE8, AL8, and AM8 as well. Therefore, WARDER would also reject such data cells from being added into this cluster.

D. Whole-cluster Validity Refinement

The last refinement concerns the validity of finally formed cell clusters, i.e., it focuses on cluster-level rather cell-level validity properties. It is expected that a cell cluster should follow a common computational semantics in terms of a unified formula that can cover most cells in this cluster. WARDER would test all existing formulas available in this cluster, and if none of them can serve for this purpose, it would consider this cluster unqualified and cancelling it from further actions to avoid misbehavior (e.g., considering most cells as defects, which turn out to be mostly false positives) in later anomaly detection. This is known as the *whole-cluster validity refinement*.

Fig. 4 gives one example by worksheet "World 1996", in which ten cells are clustered together (in yellow) by CUSTODES. CUSTODES then detected seven out of them as defects, but all of them are false positives. In fact, the ten cells contain almost totally different formulas (five patterns), which strongly indicate that they essentially follow different computational semantics. By its whole-cluster validity refinement, WARDER would reject the whole cluster. Note that WARDER needs a threshold value to control the judgment of "covering most cells". To play safe, WARDER chooses a conservative value of 50% to protect as many cell clusters as possible (CACheck chose a more aggressive value of 70%).

E. Refinement Process

According to their dependency, WARDER's three validity refinements are applied in turn. They complement each other and work at different levels (from single-cell, to multi-cell, and finally whole-cluster validities). They aim to together improve the cell clustering to make it more robust, eventually contributing to WARDER's spreadsheet defect detection, as our following evaluation shows.

TABLE I: Statistics of our experimental subjects

# spreadsheets	# worksheets	# cells	# formula cells	# cell clusters	# defects (faulty cells)
70	291	189,027	26,716	1,610	1,974

IV. Evaluation

In this section, we evaluate our WARDER technique and compare it to existing spreadsheet defect detection techniques.

We implemented WARDER in Java and used Apache POI [2] to manipulate spreadsheets. It contains a total of 7,300 lines of Java code, with about 2,500 lines added to, or modified from, CUSTODES's original code. Like CUSTODES, WARDER also annotates its analyzed spreadsheet cells by means of comments, indicating what defects are associated with the commented cells (e.g., missing formula or inconsistency formula defects).

A. Research Questions

We aim to answer the following three research questions:

- **RQ1 (Effectiveness):** *How effective is WARDER in (cell clustering and) defect detection, as compared to existing spreadsheet defect detection techniques?*
- **RQ2 (Correlation):** *Does WARDER's improved cell clustering contribute to its defect detection on the precision?*
- **RQ3 (Individual impacts):** *How do WARDER's three validity refinements contribute to its effectiveness on defect detection?*

B. Experimental Design and Setup

Subjects. To facilitate WARDER's comparison with its predecessor CUSTODES, we selected CUSTODES's own benchmark, which was originally sampled from the EUSES corpus [16], as our experimental subjects. The benchmark contains 70 spreadsheets and 291 worksheets, as shown in Table I. The 291 worksheets contain 189,027 cells, among which 26,716 are formula cells. The benchmark also contains ground truths, which annotate 1,610 cell clusters and among them 1,974 defects (faulty cells with missing or inconsistent formula issues), for evaluation purposes.

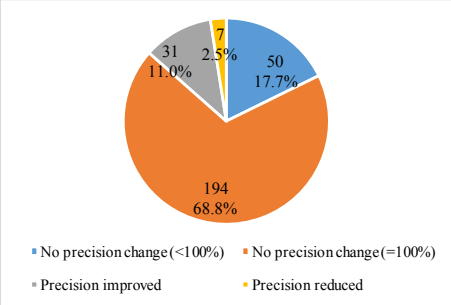
Techniques. We compare our WARDER with five aforementioned spreadsheet defect detection techniques, namely, UCheck, Dimension, AmCheck, CACheck, and CUSTSODES. We obtained their implementations from their respective authors. We compare WARDER with these techniques on their defect detection effectiveness. For CUSTODES, we additionally compare their cell clustering effectiveness.

To study individual impacts (RQ3) of WARDER's three validity refinements, we configured to enable them individually in experiments, which are named WARDER-sc (with single-cell validity), WARDER-mc (with multi-cell validity), and WARDER-wc (with whole-cluster validity). Then the configuration with all the three refinements enabled is named WARDER-full or WARDER directly.

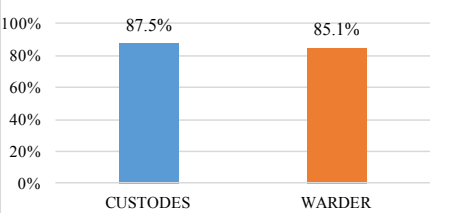
Metrics. Regarding the effectiveness on defect detection, we first measure the number of detects reported by a technique and also in the ground truths (TP), that reported but not in

TABLE II: Defect detection results for the six spreadsheet defect detection techniques

Technique	Detected	TP	FP	$precision_d$	$recall_d$	$F\text{-}measure_d$
UCheck	204	1	203	0.5%	0.1%	0.00
Dimension	1,824	14	1,828	0.8%	0.7%	0.01
AmCheck	2,372	1,316	1,030	56.1%	66.7%	0.61
CACheck	1,866	1,350	516	72.3%	68.4%	0.70
CUSTODES	2,380	1,539	841	64.7%	78.2%	0.71
WARDER	1,612	1,415	197	87.8%	71.9%	0.79



(a) Precision comparison (in terms of affected worksheets)



(b) Recall rate comparison (overall)

Fig. 5: Cell clustering results for CUSTODES and WARDER

the ground truths (FP), and that in the ground truths but not reported (FN). Based on them, we calculate $precision_d$ = TP / (TP + FP), $recall_d$ = TP / (TP + FN), and $F\text{-}measure_d = 2 \times precision_d \times recall_d / (precision_d + recall_d)$ to measure the technique's effectiveness on spreadsheet defect detection.

Regarding the effectiveness on cell clustering (applicable to WARDER and CUSTODES), we follow CUSTODES's pairwise similarity comparison [11] to calculate TP, FP, and FN. We then calculate a technique's effectiveness on cell clustering by $precision_c$, $recall_c$, and $F\text{-}measure_c$ in a similar way.

Environment. All experiments were conducted on a commodity PC with an Intel® Core™ i7-6700 CPU @3.41GHz with 64GB RAM. The machine was installed with Microsoft Windows 10 Professional and Oracle Java 8.

C. Experimental Results and Analyses

In the following, we analyze the experimental results and answer the three research questions in turn.

1) RQ1: Effectiveness. We first evaluate WARDER's effectiveness on cell clustering and defect detection, and compare it to the other five spreadsheet defect detection techniques.

Regarding defect detection, Table II compares the results for all the six techniques, which include precision, recall rate, and F-measure values, as well as the statistics of detected defects and their contained true positives and false positives. From the table, we observe that: (1) UCheck and Dimension obtained only very low scores (less than 1% for precision and recall rate, and 0.1 for F-measure) due to their limited analysis scope, echoing earlier studies [48]; (2) AmCheck and CACheck obtained much higher scores (56.1–72.3% precision, 66.7–68.4% recall rate, and 0.61–0.70 F-measure) due to their effective analysis patterns (e.g., cell arrays); (3) CUSTODES obtained a slightly better score (0.71 F-measure) than CACheck, with a focus on the recall rate (78.2% against 68.4% for CACheck); (4) WARDER, as expected, focuses on improving the precision and obtained a large improvement from 64.7% to 87.8%, leading to a final jump on the F-measure from 0.71 to 0.79 (the highest among all), even with a small sacrifice on the recall rate of about 6%, as against CUSTOEDES. One may concern that CUSTODES detected more true positives (124) than WARDER, but this was accompanied with much more false positives (644), which can be overwhelming for manual inspection.

Regarding cell clustering, Fig. 5 compares the results for CUSTODES and WARDER, which include precision and recall rate. For the precision (Fig. 5a), we partition 282 worksheets containing at least one cluster out of the total of 291 ones into four categories: (1) WARDER improved the precision for 31 worksheets, and reduced for 7 ones; (2) the precision kept unchanged for 244 worksheets, in which 194 ones already reached 100% (i.e., upper limit). In other words, WARDER improved the cell clustering for 225 worksheets (79.8%), either by improving the precision or already reaching their upper limit of 100%. We further look into details for the 38 worksheets with precision changes (Fig. 6). We observe that WARDER improved the cell clustering precision by 0.3% to 94.6% (20.7% on average), and the improvement gains are significant, much more than those lost due to reduced precisions. Besides, we also note that WARDER's effectiveness on the precision improvement came only with a marginal reduction on the recall rate of 2.4% (Fig. 5b).

From Fig. 6, we observe that WARDER improved cell clustering for most worksheets, but one may notice one exception for worksheet "Detail for College ...", where its precision dropped from 100% to zero. We further looked into this case. The ground truths suggest that cells {O11, W11, Z11, AD11, AR11} should be clustered together, as illustrated in green in Fig. 7. CUSTODES "correctly" clustered these cells together, but WARDER did not. However, we found that these five cells actually contain different formulas, and they thus violate WARDER's whole-cluster validity property (there is no common computational semantics shared among most of these cells). This explains why WARDER rejected this cluster (potentially flaw of the ground truths). In fact, this cell cluster indeed does not contain any defect. Therefore, this precision dropping on cell clustering did not affect WARDER's defect detection ability.

In summary, to answer research question RQ1, we conclude that *WARDER is effective in both cell clustering and defect detection. It greatly improves the precision (by 15.5–87.3%),*

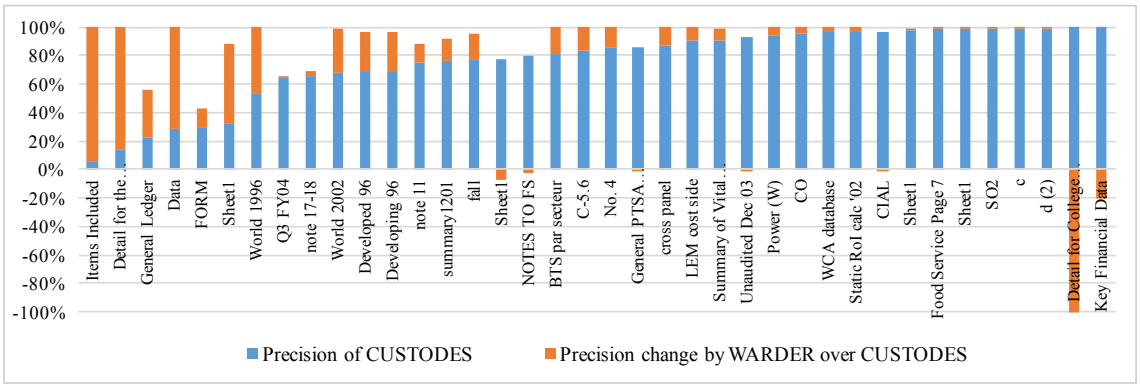


Fig. 6: Cell clustering results for CUSTODES and WARDER (precision changes)

	A	N	O	V	W	Y	Z	AC	AD	AQ	AR
9	Spon	U.S. Dep	Total Federal	Thoma	Total Foundation	Clevelar	Total Lo	Cente	Total Other	Ohio D	Total State
10	Title	Modeling		Citizen		Educatic		Fee a		Readin	
11	Total	423331	=SUM(B11:N11)	9720	=SUM(Q11:V11)	86070	=+Y11	5000	=SUM(AB11:AC11)	80851	=SUM(AF11:AQ11)

Fig. 7: Worksheet "Detail for College of Education" (one cluster marked in green by the ground truths)

and achieves the best F-measure (0.79) value among all studied spreadsheet defect detection techniques.

2) RQ2: Correlation. We then study the correlation between WARDER's precision improvement over CUSTODES on cell clustering and that on detect detection.

We use three symbols ↑, ↓, and → to represent precision improved, precision reduced, and precision unchanged these three cases, respectively. Then, we partition 140 worksheets containing at least one defect in the ground truths out of the total of 291 ones into three categories (in Table III): (1) the "correlation supported" category indicates that when WARDER, as compared to CUSTODES, has its precision improved, reduced, unchanged on cell clustering, that on defect detection behaves the same way; (2) the "correlation unsupported" category indicates that when WARDER has its precision improved on cell clustering, that on defect detection keeps unchanged or is even reduced, or when having its precision reduced on cell clustering, that on defect detection keeps unchanged or is even improved; (3) the "unknown" category lists the remaining combinations, which neither support nor unsupported the correlation. As a whole, we can observe that the first category dominates (90%), and thus suggests that WARDER's focused precision improvement on cell clustering indeed brings about its improvement on defect detection.

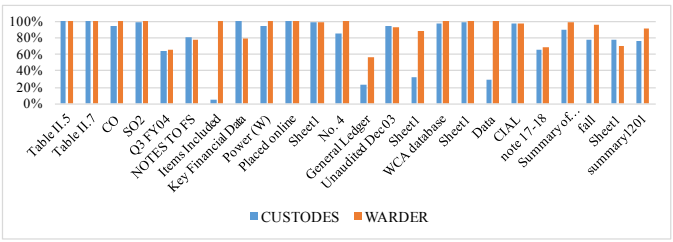
Fig. 8 shows more details about the precision comparison between WARDER and CUSTODES on cell clustering (Fig. 8a) and defect detection (Fig. 8b). To be focused, we removed those worksheets (116) having their precision unchanged for both cell clustering and defect detection, and listed only the remaining 24 ones. One can observe from the figure detailed precision changes, as well as their change correlations between cell clustering and defect detection in most cases. One may notice one exception for worksheet "CO", where WARDER's defect detection precision dropped from 100% to

TABLE III: Correlation study for WARDER against CUTODES on precision changes between cell clustering and defect detection (↑: precision improved, ↓: precision reduced, →: precision unchanged)

Category	Precision change type (cell clustering)	Precision change type (defect detection)	# work-sheets	Sum of each category
Correlation supported	↑	↑	8	126 (90.0%)
	↓	↓	2	
	→	→	116	
Correlation unsupported	↑	→	5	9 (6.4%)
	↑	↓	2	
	↓	→	2	
	↓	↑	0	
Unknown	→	↑	5	5 (3.6%)
	→	↓	0	
Total	-	-	140	140 (100.0%)

zero, although it improved the cell clustering precision. We further looked into this case. The ground truths suggest that cells {B11, E11} (in green) and {C11, F11} (in orange) should form two clusters (in Fig. 9), and cells C11 and F11 are both defects (with red triangle marks). CUSTODES detected the two defects *accidentally* by clustering the four cells together. The is accident because the four cells actually do not share any common computational semantic (the two green ones calculate the largest value, while the two orange ones calculate the second largest value). CUSTODES considered the two orange cells as defects simply because they contain plain values only (missing formula defects). One the other hand, WARDER clustered {B11, E11} only together and thus did not detect any defect in them. It missed the two orange cells because they do not contain any formula and should not form a cluster according to their definition. Without any additional evidence (e.g., more cells together and some contain formulas that can unify values in other cells), WARDER chose not to form such clusters (otherwise, more false positives can result).

In summary, to answer research question RQ2, we conclude that *WARDER's improved cell clustering over CUSTODES contributes to its improved defect detection, and this correlation is supported by 90.0% worksheets.*



(a) Precision on cell clustering between CUSTODES and WARDER

(b) Precision on defect detection between CUSTODES and WARDER

Fig. 8: Precision details in cell clustering and defect detection between WARDER and CUSTODES for worksheets with changed precisions

	A	B	C	D	E	F
4		MAX 1-HR			MAX 8-HR	
5	SITE NAME	1ST	2ND	OBS> 35	1ST	2ND
6	ASHE STREET	5.1	5.1	0	3	3
7	GREENVILLE H	5	4.7	0	3.7	3.4
8	STATE HOSPIT	5.3	4.6	0	3.8	3.3
9						
11	State Wide Max	=MAX(B6:C8)	5.1		=MAX(E6:F8)	3.7

Fig. 9: Worksheet "CO" (two clusters marked in green and orange by the ground truths)

3) RQ3: Individual impacts. Finally, we study the individual impact of WARDER's three validity refinements on its effectiveness in detecting spreadsheet defects. WARDER was configured with each refinement enabled only (named WARDER-sc, WARDER-mc, WARDER-wc, as aforementioned), and compared to the full-fledged WARDER (WARDER-full).

Table IV compares defect detection results for CUSTODES and WARDER's four configurations. We observe that: (1) WARDER's each validity refinement is useful, and individually improved the precision for defect detection by 4.9–7.6% over CUSTODES, with a small sacrifice (1.9–2.3%) on the recall rate, leading to an improvement on F-measure from 0.71 to 0.73–0.74; (2) combining all the three validity refinements together (i.e., WARDER-full) achieved the highest precision (87.8%) and F-measure (0.79), which are also echoed earlier in Table I.

TABLE IV: Defect detection results for CUSTODES and WARDER configured with different validity refinements

Technique	Detected	TP	FP	$precision_d$	$recall_d$	$F\text{-}measure_d$
CUSTODES	2,380	1,539	841	64.7%	78.2%	0.71
WARDER-sc	2,083	1,506	577	72.3%	76.3%	0.74
WARDER-mc	2,164	1,507	657	69.6%	76.3%	0.73
WARDER-wc	2,071	1,498	573	72.3%	75.9%	0.74
WARDER-full	1,612	1,415	197	87.8%	71.9%	0.79

TABLE V: Defect detection results for the four spreadsheet defect detection techniques on VEnron2

Technique	For all 6,258 worksheets			For sampled 300 worksheets		
	# reported worksheets	# reported defects	Time cost (min)	# defects	# TP	Precision
AmCheck	859	20,280	21	3,316	540	16.3%
CACheck	953	12,953	372	1,559	534	34.3%
CUSTODES	1,284	14,102	537	2,334	629	26.9%
WARDER	1,136	9,462	518	1,240	512	41.3%

In summary, to answer research question RQ3, we conclude that *WARDER's three validity refinements can individually contribute to its effectiveness on defect detection, and achieve the best effectiveness when combined them together.*

D. Case Study

Besides the preceding controlled experiments, we also evaluate our WARDER's effectiveness in detecting spreadsheet defects using a large-scale corpus VEnron2 [45]. VEnron2 contains 1,609 versioned groups, refined from original 79,983 real-life worksheets in the Enron corpus [21]. We chose the latest spreadsheet file from each versioned group, i.e., totally 1,609 spreadsheets, which correspond to a total of 7,140 worksheets as the subjects of our case study. We fed these worksheets to different spreadsheet defect detection techniques to compare their effectiveness on practical spreadsheets. Considering that UCheck and Dimension reported too few defects (less than 1%), we in the case study selected the other four techniques, namely, AmCheck, CACheck, CUSTODES, and WARDER. In order to facilitate our comparisons and make them fair, we removed some worksheets for which at least one technique failed to run normally (e.g., causing unexpected crashes or exceptions, or exceeding our controlled time limit of five minutes for handling each individual worksheet so as to avoid being trapped into dead locks or unknown errors). This treatment left us a total of 6,258 worksheets for our case study.

We note that VEnron2 does not contain ground truths for evaluating a spreadsheet defect detection technique's effectiveness (e.g., recall rate and F-measure). Therefore, we focus mainly on the precision for the four techniques. Besides, due to the large number of all worksheets, although we ran each technique on all these worksheets (e.g., for measuring their time costs), we had to use worksheet sampling and manual inspection for measuring the precision, following AmCheck's and CUSTODES's suggested evaluations. Among all the 6,258 worksheets, 1,525 worksheets were reported to contain defects by at least one technique. Based on them, we randomly sampled 20% (rounded to 300) worksheets from them for manual inspection, which decided whether each reported defect is a true one or not. Based on the inspection, Table V compares the four techniques for their defect detection results.

From Table V (300 worksheets part), we observe that: (1) among the four techniques, WARDER achieved the highest defect detection precision (i.e., 41.3%), outperforming the others by 7.0–25.0%, which echoes WARDER's focus on improving the precision against CUSTODES (41.3% vs. 26.9%); (2) although WARDER reported a little less true positives

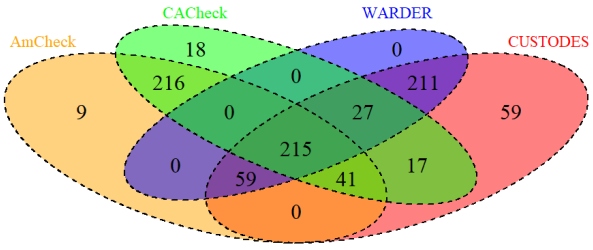


Fig. 10: Venn graph for illustrating the intersections of reported true positives between four spreadsheet defect detection techniques

(512), which was accompanied with much fewer false positives (728), which are 297–2,048 fewer than those of the other three techniques, and this feature can be quite useful since all spreadsheet defects have to be manually verified later in practice.

From Table V (6,258 worksheets part), we observe that, similar to sampled 300 worksheets, WARDER reported fewest defects (9,462), as compared to 20,280 for AmCheck, 12,953 for CACheck, and 14,102 for CUSTODES. Considering that WARDER achieved the highest precision, its report quality is expected to be high (e.g., in 1,240 defects WARDER detected 512 true positives, while in 3,316 (over 2.6 times) defects AmCheck detected only 540 true positives (marginally more)). Regarding the efficiency (time cost), AmCheck took the least time (21 minutes only) for handling all the 6,258 worksheets, while the other three techniques took much more time (351–516 minutes more). This result suggests that AmCheck can be suitable for quickly identifying *potential* defects for spreadsheets, but since its detection quality is low (precision: 16.3%), it is better accompanied with other techniques for *validation* (e.g., for filtering out most false positives). Besides, we note that WARDER was based on CUSTODES and improved its cell clustering only. Therefore, WARDER also worked not so efficiently (518 minutes), as CUSTODES did (537 minutes). Still, WARDER reduced irrelevant cells and unqualified clusters, thus reducing unnecessary workload, and this effort contributed to a reduction to the time cost of 19 minutes.

Besides the overall precision and time cost comparisons, we also study the intersections of reported true positives between the four spreadsheet detection techniques, as illustrated by the Venn graph in Fig. 10. In the figure, the yellow ellipse represents the true positives reported by AmCheck, green by CACheck, pink by CUSTODES, and purple by WARDER. Each sub-area represents a specific intersection of reported true positives by two or more techniques. From the figure, we observe that: (1) The pattern-based group (i.e., AmCheck and CACheck) and learning-based (in particular, clustering-based) group (i.e., CUSTODES and WARDER) clearly complement each other. The former has 243 (9 + 216 + 18) unique defects undetectable by the latter, and the latter has 270 (211 + 59) undetectable by the former. This result suggests that both groups of techniques are useful. (2) In the pattern-based group, CACheck extends over AmCheck. Accordingly, they share a large portion of detected defects (78.4%, or 472 out of 602). As a result, AmCheck has 68 unique defects only, and CACheck has 62 ones. Still, CACheck is welcome, considering that it significantly reduced false positives (from AmCheck's 2776 to its 1025) in Table V. (3) In the learning-based group, WARDER refines over CUSTODES. Accordingly, it reported 512 true positives, only a subset (81.4%) of those reported by CUSTODES (629). Nevertheless, this result is accompanied with WARDER's focus on filtering out irrelevant cells and unqualified clusters, and this effort also led to significantly reduced false positives (from CUSTODES's 1705 to its 728) in Table V. (4) CACheck and WARDER, as the best representative in each group, still complement each other. They each have 292 and 270 unique defects, respectively, undetectable by the other. This result strongly suggests their complementary usage to each other.

As a conclusion, *WARDER is also satisfactory in detecting defects for practical spreadsheets. It achieved the highest precision (41.3%), outperforming other techniques by 7.0–25.0%. Its time cost is a bit high, but comparable to CACheck (at the same magnitude), and less than its predecessor CUSTODES. Regarding detected defects, all studied techniques have their own strengths, and are suggested for complementary usage to each other.*

E. Threat Analyses and Discussions

One threat concerns the calculation of cell clustering metrics (i.e., $precision_c$, $recall_c$, and $F\text{-}measure_c$). They are based on the TP, FP, and FN notions calculated from CUSTODES's pair-wise similarity comparisons. We note that such comparisons count the numbers of cell pairs on whether they belong to the same cluster or belong to two different clusters. Such calculations are different from those measuring detected defects. As a result, studying the correlation between WARDER's cell clustering and its defect detection could be affected to some extent. Nevertheless, we still observed 90% worksheets supporting our studied correlation. This suggests that WARDER's improved cell clustering indeed contributes to its defect defection.

One may also notice that WARDER still has room for improvement, considering that it failed to detect certain spreadsheet defects, as we analyzed earlier. There are two major reasons. First, WARDER has focused on identifying irrelevant cells (for removal) and unqualified clusters (for cancellation). It does not retrieve back those relevant cells that have already been missed by CUSTODES. Therefore, both WARDER and CUSTODES could fail to detect spreadsheet defects related to such missed cells. Second, even if all relevant cells can be correctly clustered, CUSTODES itself can still fail to detect certain defects due to its limited scope in anomaly detection. Since WARDER focuses on improving cell clustering only, it does not touch the anomaly detection part. As a result, both techniques could fail to detect such spreadsheet defects. Nevertheless, we observed in experiments

that WARDER already outperformed CUSTODES largely. This suggests that WARDER has focused on a dominating factor for the improvement. Still, the above analyses point out new ways for further improvement.

We note that we attempted but did not manage to compare our WARDER to the other two learning-based techniques, Melford [42] and ExceLint [8] in our experiments and case study. For the former, we did not find its tool available. For the latter, we did find its tool available but we encountered problems in the evaluation. First, ExceLint's scope is very different from those of the other six studied spreadsheet defect detection techniques, in that it focuses only on detecting part of inconsistent formula defects that have been caused by wrong references. Second, ExceLint considers missing formula defects not so important since they may not trigger errors immediately. However, all the other techniques consider such defects harmful and detect them, since such defects can trigger unexpected errors when concerned spreadsheets undergo future maintenance. In fact, missing formula defects are common in practice spreadsheets (e.g., 64–78% in VEnron2 by different techniques). Therefore, directly comparing WARDER with ExceLint can be unfair and would seriously underestimate ExceLint's effectiveness. Besides, we also encountered problem when running ExceLint as it lacked a specifically-annotated ground truth. Therefore, we leave its comparison to future work.

V. Related Work

In this section, we present and discuss related work in recent years. We organize the related work along four lines, namely, spreadsheet quality issues, spreadsheet defect detection, spreadsheet defect fixing and prevention, and other spreadsheet-related research.

Spreadsheet quality issues. Spreadsheet quality issues are common. They can contain various defects [33], [35], [37], [38], and these defects cause catastrophic losses to human lives [1], [34], [39]. Galletta et.al [17] conducted an empirical study on spreadsheets, and reported that even spreadsheet experts cannot significantly outperform novices in identifying spreadsheet defects. This result suggests that identifying spreadsheet defects can be a non-trivial research problem. Nixon and O'Hara [32] reported a positive assistance by supporting auditing in spreadsheets. Later, Anderson [6] confirmed the usefulness of such assistance, but also raised the concern for numerous missed spreadsheet defects. To better understand spreadsheet relations and maintain the spreadsheet quality, Mittermeir et al. [12], [31] proposed three types of "logical areas" for clustering those formula cells that satisfy three forms of equivalences, namely, copy, logical, and structural equivalences, respectively. Such clustering can help spreadsheet users better understand conceptual models behind spreadsheets, and avoid or inspect defective cells more easily.

Spreadsheet defect detection. There is one line of work particularly focused on detecting spreadsheet defects. UCheck [3], [4] and Dimension [10] are probably representative pioneers on this aspect. Based on unit or dimensional information derived from spreadsheet tables, they verify the correctness of formula calculations by checking whether there exists any illegal combination of incompatible units. Then, Hermans et al. implemented portable tools to detect and visualize several types of spreadsheet defects by focusing on inter-worksheet smells [22], data clones [25], and formula smells [23], [24]. They are the first to adapt the concept of code smell in conventional programs to the spreadsheet domain. After that, Abreu et al. [5] used a generic spectrum-based strategy to localize defects and improved the localization precision and recall for spreadsheet defects. Hofer et al. [26], [27] further studied the impact of different similarity coefficients on the accuracy of spectrum-based spreadsheet defect localization. Meanwhile, AmCheck [14] and its follow-up extension CACheck [15] were proposed to support effective defect detection by focusing on spreadsheet smells caused by ambiguous computations based on the notion of cell array. Similarly, Xu et al. [46] proposed to detect defective empty cells in spreadsheets by analyzing the context of empty cells. Cheung et al. [11] proposed CUSTODES by adaptive learning, based on formula-related cell clustering and anomaly-oriented defect detection. CUSTODES also provided a spreadsheet benchmark to facilitate follow-up research evaluation. Two examples built on this benchmark are Melford [42] and ExceLint [8]. The former used a network-based technique to detect missing formula defects, while the latter used statistical techniques to measure the likelihood of a spreadsheet defect based on the entropy and layout of its associated references for detecting inconsistent formula (or reference) defects. Similar inconsistency issues can also raise concerns for general software in many fields, e.g., context inconsistency detection for adaptive applications [44], inconsistency management for software engineering [43], etc. Our WARDER in this paper is closely related to this line of work, and it makes attempts to improve spreadsheet defect detection by refining CUSTODES's cell clustering based on cell-level and cluster-level validity properties.

Spreadsheet defect fixing and prevention. Spreadsheet defect detection techniques can also come with fixing suggestions. For example, besides detecting spreadsheet smells, CACheck also proposed to automatically repair its detected smells by synthesizing and recovering intended computational semantics in terms of formulas for concerned cells. Some work goes one step further by focusing on preventing spreadsheet defects from occurring in advance. For example, Luckey et al. [30] targeted on supporting correct spreadsheet evolution (i.e., without introducing defects into spreadsheets). Cunha et al. [13] aimed at helping build a more reliable spreadsheet programming environment. Besides, some pieces of work focus on achieving better spreadsheet maintenance practice. For example, Harutyunyan et al. [20] proposed to automatically identifying differences between spreadsheet versions so that maintainance can be more reliably conducted. Badame and Dig [7] proposed to obtain different measures on spreadsheet formulas, so that these formulas can be better refactored for maintenance. Our WARDER work may also consider

enhancing itself with spreadsheet defect fixing or prevention suggestions in future.

Other spreadsheet-related researches. There are also some pieces of work focused on similar topic research. For example, Zhang et al. [47] proposed an automated approach to improving the expression for nested-IF formulas in spreadsheets by removing logic redundancy, so that high-level formula semantics can be more easily identified for better understanding. Samuel et al. [28] modeled common spreadsheet formulas and relations through predicates and expressions, and used a two-stage approach to generating and testing spreadsheets by constraint solving, so that constraints across spreadsheet cells can be discovered. Finally, program synthesis techniques are becoming increasingly popular in the spreadsheet domain, and many pieces of research have been proposed for solving spreadsheet-specific problems such as table transformation [19], string synthesis [18], and number transformation [41] by a programming-by-example approach. We have not considered program synthesis techniques in our WARDER work, but may try along this direction in future (as AmCheck and CACheck, which have tried and received promising results).

VI. Conclusion

In this paper, we study the problem of spreadsheet defect detection. We present WARDER for refining CUSTODES's cell clustering in order to improve its effectiveness in detecting spreadsheet defects. WARDER is based on our key observations that rely on our identified three validity properties to prevent fragile clusters from being formed, which can involve irrelevant cells and unqualified clusters. These properties concern different levels of cluster validities, namely, single-cell, multi-cell, and whole-cluster validities, whose uses can effectively contribute to improved precision of spreadsheet defect detection. Our experimental evaluation with a benchmark and case study with a large-scale spreadsheet corpus have confirmed WARDER's effectiveness in detecting spreadsheet defects, in particular on the detection precision.

WARDER still has room for improvement as we analyzed earlier. For example, its validity framework can be extended with more validity properties. Besides, currently WARDER focuses only on filtering out irrelevant cells, and it can be extended for retrieving back missed relevant cells. Also, it can be extended for improving the anomaly detection part. We are working along these lines.

Acknowledgement

This work was supported in part by National Key R&D Program (Grant #2017YFB1001801) and National Natural Science Foundation (Grant #61690204) of China. The authors would also like to thank the support of the Collaborative Innovation Center of Novel Software Technology and Industrialization, Jiangsu, China.

References

[1] http://www.eusprig.org/index.htm. [Online; accessed 8-March-2019].

[2] https://poi.apache.org/. [Online; accessed 8-March-2019].

[3] Robin Abraham and Martin Erwig. Header and unit inference for spreadsheets through spatial analyses. In *2004 IEEE Symposium on Visual Languages-Human Centric Computing*, pages 165–172. IEEE, 2004.

[4] Robin Abraham and Martin Erwig. UCheck: A spreadsheet type checker for end users. *Journal of Visual Languages & Computing*, 18(1):71–95, 2007.

[5] Rui Abreu, Jácome Cunha, João Paulo Fernandes, Pedro Martins, Alexandre Perez, and João Saraiva. Smelling faults in spreadsheets. In *30th IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, Canada, September 29 - October 3, 2014*, pages 111–120, 2014.

[6] W Anderson. *A comparison of automated and manual spreadsheet error detection*. PhD thesis, Master thesis, Massey University, 2004.

[7] Sandro Badame and Danny Dig. Refactoring meets spreadsheet formulas. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pages 399–409. IEEE, 2012.

[8] Daniel W. Barowy, Emery D. Berger, and Benjamin Zorn. ExceLint: Automatically finding spreadsheet formula errors. *Proc. ACM Program. Lang.*, 2(OOPSLA):148:1–148:26, October 2018.

[9] Patrick Carey and K.N. Berk. *Data Analysis with Microsoft Excel*. Brooks/Cole, 1997.

[10] Chris Chambers and Martin Erwig. Automatic detection of dimension errors in spreadsheets. *Journal of Visual Languages & Computing*, 20(4):269–283, 2009.

[11] Shing-Chi Cheung, Wanjun Chen, Yepang Liu, and Chang Xu. CUSTODES: automatic spreadsheet cell clustering and smell detection using strong and weak features. In *Proceedings of the 38th International Conference on Software Engineering*, pages 464–475. ACM, 2016.

[12] Markus Clermont. Auditing large spreadsheet programs. In *In International Conference on Information Systems Implementation and Modelling*. Citeseer, 2003.

[13] Jácome Cunha, Jorge Mendes, João Saraiva, and Joost Visser. Model-based programming environments for spreadsheets. *Science of Computer Programming*, 96:254–275, 2014.

[14] Wensheng Dou, Shing-Chi Cheung, and Jun Wei. Is spreadsheet ambiguity harmful? detecting and repairing spreadsheet smells due to ambiguous computation. In *Proceedings of the 36th International Conference on Software Engineering*, pages 848–858. ACM, 2014.

[15] Wensheng Dou, Chang Xu, Shing-Chi Cheung, and Jun Wei. CACheck: detecting and repairing cell arrays in spreadsheets. *IEEE Transactions on Software Engineering*, 43(3):226–251, 2017.

[16] Marc Fisher and Gregg Rothermel. The EUSES spreadsheet corpus: a shared resource for supporting experimentation with spreadsheet dependability mechanisms. In *ACM SIGSOFT Software Engineering Notes*, volume 30, pages 1–5. ACM, 2005.

[17] Dennis F Galletta, Dolphy Abraham, Mohamed El Louadi, William Lekse, Yannis A Pollalis, and Jeffrey L Sampler. An empirical study of spreadsheet error-finding performance. *Accounting, Management and Information Technologies*, 3(2):79–95, 1993.

[18] Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. In *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*, pages 317–330, 2011.

[19] William R. Harris and Sumit 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, June 4-8, 2011*, pages 317–328, 2011.

[20] Anna Harutyunyan, Glencora Borradaile, Christopher Chambers, and Christopher Scaffidi. Planted-model evaluation of algorithms for identifying differences between spreadsheets. In *2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 7–14. IEEE, 2012.

[21] Felienne Hermans and Emerson Murphy-Hill. Enron's spreadsheets and related emails: A dataset and analysis. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pages 7–16. IEEE, 2015.

[22] Felienne Hermans, Martin Pinzger, and Arie van Deursen. Detecting and visualizing inter-worksheet smells in spreadsheets. In *Proceedings of the 34th International Conference on Software Engineering*, pages 441–451. IEEE Press, 2012.

[23] Felienne Hermans, Martin Pinzger, and Arie van Deursen. Detecting code smells in spreadsheet formulas. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pages 409–418. IEEE, 2012.

[24] Felienne Hermans, Martin Pinzger, and Arie van Deursen. Detecting and refactoring code smells in spreadsheet formulas. *Empirical Software Engineering*, 20(2):549–575, 2015.

[25] Felienne Hermans, Ben Sedee, Martin Pinzger, and Arie van Deursen. Data clone detection and visualization in spreadsheets. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 292–301. IEEE, 2013.

[26] Birgit Hofer, Alexandre Perez, Rui Abreu, and Franz Wotawa. On the empirical evaluation of similarity coefficients for spreadsheets fault localization. *Autom. Softw. Eng.*, 22(1):47–74, 2015.

[27] Birgit Hofer, André Riboira, Franz Wotawa, Rui Abreu, and Elisabeth Getzner. On the empirical evaluation of fault localization techniques for spreadsheets. In *Fundamental Approaches to Software Engineering - 16th International Conference, FASE 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, pages 68–82, 2013.

[28] Samuel Kolb, Sergey Paramonov, Tias Guns, and Luc De Raedt. Learning constraints in spreadsheets and tabular data. *Machine Learning*, 106(9-10):1441–1468, 2017.

[29] Barry R Lawson, Kenneth R Baker, Stephen G Powell, and Lynn Foster-Johnson. A comparison of spreadsheet users with different levels of experience. *Omega*, 37(3):579–590, 2009.

[30] Markus Luckey, Martin Erwig, and Gregor Engels. Systematic evolution of model-based spreadsheet applications. *Journal of Visual Languages & Computing*, 23(5):267–286, 2012.

[31] Roland Mittermeir and Markus Clermont. Finding high-level structures in spreadsheet programs. In *Ninth Working Conference on Reverse Engineering, 2002. Proceedings.*, pages 221–232. IEEE, 2002.

[32] David Nixon and Mike O'Hara. Spreadsheet auditing software. *arXiv preprint arXiv:1001.4293*, 2010.

[33] Ray Panko. Facing the problem of spreadsheet errors. *Decision Line*, 37(5):8–10, 2006.

[34] 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.

[35] Raymond R Panko. Spreadsheet errors: What we know. what we think we can do. *arXiv preprint arXiv:0802.3457*, 2008.

[36] Raymond R Panko and Salvatore Aurigemma. Revising the panko–halverson taxonomy of spreadsheet errors. *Decision Support Systems*, 49(2):235–244, 2010.

[37] Stephen G Powell, Kenneth R Baker, and Barry Lawson. A critical review of the literature on spreadsheet errors. *Decision Support Systems*, 46(1):128–138, 2008.

[38] Kamalasen Rajalingham, David R Chadwick, and Brian Knight. Classification of spreadsheet errors. *arXiv preprint arXiv:0805.4224*, 2008.

[39] Carmen M Reinhart and Kenneth S Rogoff. Growth in a time of debt. *American Economic Review*, 100(2):573–78, 2010.

[40] Jorma Sajaniemi. Modeling spreadsheet audit: A rigorous approach to automatic visualization. *Journal of Visual Languages & Computing*, 11(1):49–82, 2000.

[41] Rishabh Singh and Sumit Gulwani. Synthesizing number transformations from input-output examples. In *Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*, pages 634–651, 2012.

[42] Rishabh Singh, Benjamin Livshits, and Benjamin Zorn. Melford: Using neural networks to find spreadsheet errors. *Tech. Rep.*, 2017.

[43] George Spanoudakis and Andrea Zisman. Inconsistency management in software engineering: Survey and open research issues. In *Handbook of Software Engineering and Knowledge Engineering: Volume I: Fundamentals*, pages 329–380. World Scientific, 2001.

[44] Huiyan Wang, Chang Xu, Bingying Guo, Xiaoxing Ma, and Jian Lu. Generic adaptive scheduling for efficient context inconsistency detection. *IEEE Transactions on Software Engineering*, 2019.

[45] Liang Xu, Wensheng Dou, Chushu Gao, Jie Wang, Jun Wei, Hua Zhong, and Tao Huang. SpreadCluster: recovering versioned spreadsheets through similarity-based clustering. In *Proceedings of the 14th International Conference on Mining Software Repositories*, pages 158–169. IEEE Press, 2017.

[46] Liang Xu, Shuo Wang, Wensheng Dou, Bo Yang, Chushu Gao, Jun Wei, and Tao Huang. Detecting faulty empty cells in spreadsheets. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 423–433. IEEE, 2018.

[47] Jie Zhang, Shi Han, Dan Hao, Lu Zhang, and Dongmei Zhang. Automated refactoring of nested-if formulae in spreadsheets. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 833–838. ACM, 2018.

[48] Ruiqing Zhang, Chang Xu, Shing-Chi Cheung, Ping Yu, Xiaoxing Ma, and Jian Lu. How effectively can spreadsheet anomalies be detected: An empirical study. *Journal of Systems and Software*, 126:87–100, 2017.