Front. Comput. Sci., 2024, 18(2): 182203
https://doi.org/10.1007/s11704-023-2384-6

REVIEW ARTICLE

Spreadsheet quality assurance: a literature review

Pak-Lok POON (✉)[1], Man Fai LAU[2], Yuen Tak YU[3], Sau-Fun TANG[4]

1 School of Engineering and Technology, Central Queensland University, Melbourne 3000, Australia
2 Department of Computing Technologies, Swinburne University of Technology, Hawthorn 3122, Australia
3 Department of Computer Science, City University of Hong Kong, Hong Kong SAR 999077, China
4 The Royal Victorian Eye and Ear Hospital, East Melbourne 3002, Australia

Abstract Spreadsheets are very common for information processing to support decision making by both professional developers and non-technical end users. Moreover, business intelligence and artificial intelligence are increasingly popular in the industry nowadays, where spreadsheets have been used as, or integrated into, intelligent or expert systems in various application domains. However, it has been repeatedly reported that faults often exist in operational spreadsheets, which could severely compromise the quality of conclusions and decisions based on the spreadsheets. With a view to systematically examining this problem via survey of existing work, we have conducted a comprehensive literature review on the quality issues and related techniques of spreadsheets over a 35.5-year period (from January 1987 to June 2022) for target journals and a 10.5-year period (from January 2012 to June 2022) for target conferences. Among other findings, two major ones are: (a) Spreadsheet quality is best addressed throughout the whole spreadsheet life cycle, rather than just focusing on a few specific stages of the life cycle. (b) Relatively more studies focus on spreadsheet testing and debugging (related to fault detection and removal) when compared with spreadsheet specification, modeling, and design (related to development). As prevention is better than cure, more research should be performed on the early stages of the spreadsheet life cycle. Enlightened by our comprehensive review, we have identified the major research gaps as well as highlighted key research directions for future work in the area.

Keywords decision support system, end-user computing, end-user programming, Excel, spreadsheet

1 Introduction

Since their inception, spreadsheets have been widely used in many scientific and mission-critical applications [1] such as linear programming and regression [2], control systems [3], statistical radiobiological evaluation [4], drug-alcohol interaction [5], patient specimen collection [6], neuroscience [7], and nuclear fuel production [8]. Many decision support systems (DSS) in the commercial and social sectors are built on spreadsheets to generate useful information for strategic decision making [9].

Along with the high popularity of spreadsheet applications [10], it was found that about 94% of the spreadsheets in use contained faults [11,12]. A major reason for a high number of faulty spreadsheets is the accelerating trend in end-user computing (or end-user programming) over the last few decades [13−15]. Spreadsheet development, now a prominent example of end-user computing [16,17], has shifted from being often done by well-trained IT professionals to something millions of non-technical departmental end users or *end-user programmers* are now responsible to do. As most end-user programmers are not well trained in software development and testing [18], it is not surprising that many spreadsheets they developed are poorly coded and inadequately tested [19]. Consequently, these spreadsheets are likely to contain faults that are not properly detected and removed before release for daily operational use.

Faulty spreadsheets could result in business risks including: (a) loss in revenue, profit, cash, assets, and tax, (b) mispricing and poor decision-making, and (c) financial failure [20]. If faulty spreadsheets are used in critical areas such as clinical medicine [21] and nuclear operations [8], catastrophic consequences may result. Hence, spreadsheet quality assurance (QA) is a serious issue that cannot be ignored. Our literature review aims to provide a holistic view of how various quality issues of spreadsheets can be addressed. This paper offers the following contributions: (a) an extensive literature review on spreadsheet QA over a 35.5-year period (from January 1987 to June 2022) for target journals and a 10.5-year period (from January 2012 to June 2022) for target conferences; (b) a holistic and comprehensive review of all spreadsheet QA issues over the entire spreadsheet life cycle so that the quality of spreadsheets and the data generated from them can be improved; and (c) identification of existing research gaps in the spreadsheet QA area, shedding light on future research directions.

Received June 23, 2022; accepted March 2, 2023

E-mail: p.poon@cqu.edu.au

2 Existing reviews related to spreadsheet QA

Jannach et al. [22] performed a review mainly on *automated* spreadsheet QA approaches. Also, most of their collected papers were technical-oriented and information-technology (IT) in nature. Our review differentiates from their work in several aspects: (a) our literature search included not only IT journals, but also business-oriented information-systems (IS) journals, dual-focused IT/IS journals, and management science (MS)/operational research (OR) journals, (b) our study covers not only automated QA approaches but also static and manual approaches that are at least equally important, and (c) our study covers all the stages of the spreadsheet life cycle.

Thorne [23] performed a literature review of spreadsheet mistake reduction techniques. Similarly, Powell et al. [24] conducted a critical review of the literature on spreadsheet errors. Since the publication of these two papers in 2008 or 2009, there have been many new research studies published in the past decade related to spreadsheet QA. More importantly, our literature review in this paper clearly differentiates from [23,24] in several significant aspects: (a) our study covers previous work published in a long time period (35.5 years for target journals and 10.5 years for target conferences), (b) our study is based on a well-defined search strategy, (c) the scope of our study is more comprehensive in terms of both the number and type of published works discussed (e.g., journals: technical-oriented IT, business-oriented IS, dual-focused IT/IS, MS/OR; conferences: technical-oriented IT, business-oriented IS), (d) our discussion of spreadsheet QA techniques is set within a spreadsheet life-cycle framework to facilitate in-depth analysis, and (e) our discussion is based on the established IEEE terminology [25−27], e.g., making a clear distinction: (i) among mistakes, faults, and failures, and (ii) among different types of static testing.

More recently, Hofer et al. [28] reported their systematic review on product metrics for spreadsheets. Their paper is fairly restricted in scope — it only focuses on the spreadsheet metrics in several contexts such as fault prediction, refactoring, and risk assessment for audits. Concepts and use of spreadsheet metrics contribute only to a rather small part of the entire spectrum of spreadsheet QA.

3 Methodology and scope

3.1 Target journals and conferences

Our extensive and systematic search of the literature involved a total of 32 target journals plus three reputable professional magazines published by ACM and IEEE, which are known or expected publication outlets for spreadsheet-related studies (see Table 1). To avoid verbosity, we henceforth refer to the three selected magazines [T06,T07,T08] in Table 1 as "journals".

The 35 target journals in Table 1 can be broadly classified as follows. (a) *Technical-oriented IT journals:* They publish papers mainly in computer science, software engineering, and other technical IT areas (16 journals [T01–T16]). (b) *Business-oriented IS journals*: They publish papers mainly in information systems with more emphasis on business applications (14 journals [B17–B30]). (c) *Dual-focused journals targeting on IT or IS:* They publish both technical-oriented IT and business-oriented IS papers (3 journals [D31–D33]). (d) *Management science (MS)/operational research (OR) journals:* They publish papers in MS and OR areas (2 journals [M34–M35]).

Besides journals, our literature search also included the following eight conferences:

- ACM International Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA),
- ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE),
- ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA),
- IEEE/ACM International Conference on Automated Software Engineering (ICASE),
- IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC),
- International Conference on Software Engineering (ICSE),
- Hawaii International Conference on System Sciences (HICSS), and
- International Conference on Information Systems (ICIS).

In the above, the top six are technical-oriented IT conferences and the bottom two are business-oriented IS conferences. Note that: (a) starting from 2017, papers accepted by OOPSLA has been published annually as a special issue in the *Proceedings of the ACM on Programming Languages* (*PACMPL*[1)]), and (b) HICSS and ICIS are commonly considered as top-tier conferences on information systems.

3.2 Search strategy and results

We started our literature search for journal papers from January 1987. Choosing this start date is based on the development history of spreadsheets — it was reported that Microsoft launched the Windows operating system in 1987 and Excel was one of the first application products bundled with it [29]. Our search period was from January 1987 to June 2022 (35.5 years).

Table 2 lists the search terms we used for searching the 35 target journals in Table 1. If the title, abstract, or keywords of a paper contains any search term(s) in Table 2, it would be included in our initial pool of papers. The search was repeated twice: a manual scanning followed by an automatic search (e.g., via the journal's search engines). This reduced the chance of accidentally omitting some relevant papers. After searching, a total of 544 potentially relevant papers were included in the initial pool. Not all these 544 papers are related to spreadsheet QA. For example, some of these papers are

1) *PACMPL* is a journal, not conference proceedings.

Table 1 Target journals

Target journal	Abbreviation	Search started from *
Technical-oriented IT		
T01. ACM Computing Surveys	CSUR	Vol. 19, no. 1, 1987
T02. ACM Transactions on Computer-Human Interaction	TOCHI	Vol. 1, no. 1, 1994
T03. ACM Transactions on Information Systems (previously ACM Transactions on Office Information Systems)	TOIS	Vol. 5, no. 1, 1987
T04. ACM Transactions on Software Engineering & Methodology	TOSEM	Vol. 1, no. 1, 1992
T05. Automated Software Engineering	ASE	Vol. 1, no. 1, 1994
T06. Communications of the ACM	CACM	Vol. 30, no. 1, 1987
T07. Computer	COMP	Vol. 20, no. 1, 1987
T08. IEEE Software	SW	Vol. 4, no. 1, 1987
T09. IEEE Transactions on Reliability	TR	Vol. R-36, no. 1, 1987
T10. IEEE Transactions on Services Computing	TSC	Vol. 1, no. 1, 2008
T11. IEEE Transactions on Software Engineering	TSE	Vol. SE-13, no. 1, 1987
T12. Information & Software Technology	IST	Vol. 29, no. 1, 1987
T13. International Journal of Human-Computer Studies (previously International Journal of Man-Machine Studies)	IJHCS	Vol. 26, no. 1, 1987
T14. Journal of Functional Programming	JFP	Vol. 1, no. 1, 1991
T15. Journal of Computer Languages (previously Journal of Visual Languages & Computing)	JCL	Vol. 1, no. 1, 1990
T16. Journal of Systems & Software	JSS	Vol. 7, no. 1, 1987
Business-oriented IS		
B17. Communications of the Association for Information Systems	CAIS	Vol. 1, 1999
B18. European Journal of Information Systems	EJIS	Vol. 1, no. 1, 1991
B19. Information & Management	IM	Vol. 12, no. 1, 1987
B20. Information & Organization (previously Accounting, Management & Information Technologies)	IO	Vol. 1, no. 1, 1991
B21. Information Systems Journal (previously Journal of Information Systems)	ISJ	Vol. 1, no. 1, 1991
B22. Information Systems Research	ISR	Vol. 1, no. 1, 1990
B23. Information Technology & People (previously Office Technology & People)	ITP	Vol. 3, no. 1, 1987
B24. Journal of the Association for Information Systems	JAIS	Vol. 1, no. 1, 2000
B25. Journal of Information Technology	JIT	Vol. 2, no. 1, 1987
B26. Journal of Management Information Systems	JMIS	Vol. 3, no. 4, 1987
B27. Journal of Organizational & End User Computing (previously Journal of End User Computing and Journal of Microcomputer Systems Management)	JOEUC	Vol. 1, no. 1, 1989
B28. Journal of Strategic Information Systems	JSIS	Vol. 1, no. 1, 1991
B29. MIS Quarterly	MISQ	Vol. 11, no. 1, 1987
B30. MISQ Executive	MISQ-E	Vol. 1, no. 1, 2002
Dual-focused (IT & IS)		
D31. Decision Support Systems	DSS	Vol. 3, no. 1, 1987
D32. Information Processing & Management (previously Information Storage & Retrieval)	IPM	Vol. 23, no. 1, 1987
D33. Interacting with Computers	IWC	Vol. 1, no. 1, 1989
Management science/operational research		
M34. Journal of the Operational Research Society	JORS	Vol. 38, no. 1, 1987
M35. Omega: The International Journal of Management Science (commonly known as Omega)	OMEGA	Vol. 15, no. 1, 1987

* For any target journal that started its publication *after* January 1987, the literature search started from its first publication.

related to spreadsheets in a general context, and some are related to end-user computing outside the spreadsheet domain. We then manually evaluated these 544 papers and selected only those related to spreadsheet QA. To reduce possible omission, this manual evaluation was first done by one of the authors, and was then independently verified by another author. We found 86 such papers: 41 came from technical-oriented IT journals, 30 from business-oriented IS journals, 7 from dual-focused IT/IS journals, and 8 from MS/OR journals (see Table 3).

We took a holistic view in determining whether a paper in the initial pool is related to spreadsheet QA. Instead of only focusing on some specific activities such as testing and debugging, we opine that a wide range of activities throughout the entire spreadsheet life cycle (e.g., problem identification, specification, modelling, design, implementation, testing,

Table 2 Search terms

Search terms *	
Spreadsheet(s); spreadsheet-based	Excel; Lotus (123); Lotus (1-2-3); Lotus (Symphony); VisiCalc; SuperCalc; Quattro
End(-)user computing; user computing; EUC	Personal computing; user(-)led development; user(-)driven development
End(-)user system(s) development; end(-)user software engineering	End(-)user development; end(-)user(s') system(s) development; end(-)user(s') software engineering; end(-)user(s') debugging
End(-)user developed system(s); end(-)user developed software; user(-)developed application(s); user(-)developed system(s); user(-)developed software	End(-)user system(s); end(-)user software; end(-)user application(s)
End(-)user programming; end(-)user programmer(s)	Application(s) development by end(-)user(s); end(-)user developer(s)

* If a search term contains "(s)", "(s')", or "(-)", it means that this search term can either include or exclude the character(s) enclosed in brackets. For example, the search term "End(-)user developed system(s)" could be "End user developed system", "End-user developed system", "End user developed systems", or "End-user developed systems". Also, the search will not differentiate between capital and lowercase letters.

Table 3 Relevant journal papers related to spreadsheet QA

Technical-oriented IT		Business-oriented IS		Dual-focused (IT & IS)		Management science (MS) / operational research (OR)	
Target journals	No. of papers	Target journals	No. of papers	Target journals	No. of papers	Target journals	No. of papers
ASE	4	CAIS	3	DSS	6	JORS	5
CSUR	1	EJIS	0	IPM	0	OMEGA	3
TOCHI	1	IM	2	IWC	1		
TOIS	2	IO	2				
TOSEM	2	ISJ	0				
CACM	2	ISR	0				
COMP	0	ITP	1				
SW	2	JAIS	2				
TR	1	JIT	0				
TSC	0	JMIS	4				
TSE	6	JOEUC	16				
IST	2	JSIS	0				
IJHCS	4	MISQ	0				
JFP	2	MISQ-E	0				
JCL	6						
JSS	6						
Total count:	41	**Total count:**	30	**Total count:**	7	**Total count:**	8

debugging, and maintenance) play a part in spreadsheet QA. For example, if problem identification is not properly performed, the implemented spreadsheet will not satisfy the users' expectation. In this case, the spreadsheet obviously suffers from a "poor" quality issue. In other words, we determine the quality of a spreadsheet by its ability to satisfy users' expectation. Accordingly, if a paper is related to any activity in the spreadsheet life cycle that affects the ability of a spreadsheet in fulfilling its users' expectation, this paper is considered to be related to spreadsheet QA.

We repeated the above search strategy for the eight target conferences listed in Section 3.1 for a 10.5-year period (from January 2012 to June 2022). This search found an additional of 36 papers related to spreadsheet QA. See Table 4 for the breakdown details. All in all, our search found 122 (= 86 + 36) journal and conference papers related to spreadsheet QA.

Table 4 Relevant conference papers related to spreadsheet QA

Target conferences	No. of papers
OOPSLA (or *PACMPL* from 2017 onwards)	2
ESEC/FSE	3
ISSTA	2
ICASE	2
VL/HCC	18
ICSE	9
HICSS	0
ICIS	0
Total count:	36

The 86 relevant journal papers (see Table 3) and the 36 relevant conference papers (see Table 4) formed our "primary" source of literature. We also followed the relevant references (including other journal papers, conference papers, and edited book chapters) mentioned in the papers in the primary source. These relevant references subsequently found formed our "secondary" source of literature.

3.3 Paper analysis

To frame our analysis of the papers, we first conducted a rigorous review of several software-engineering-based approaches which have been advocated in the literature for achieving quality development of spreadsheet software. Based on our review, we decided to adapt and extend an existing spreadsheet life-cycle framework for the purpose of this study. The framework consists of five stages, which are further refined to activities or substages, as detailed in Section 5.

4 Key terminologies and concepts

Inconsistent use of terminologies and concepts is observed in some literature on spreadsheets and even within the software community. Such use differs from the standard meanings attached to the terminologies/concepts in the context of software engineering/development. This inconsistency problem may cause confusion to software practitioners/resear-

chers in the field, and make our subsequent discussion difficult to comprehend. To alleviate this problem, we first discuss the standard meanings of some key terminologies/concepts used in our literature review.

4.1 Failure, fault, mistake, and error

According to the IEEE terminologies [25−27], a *failure* is an "externally visible" deviation from the system's specification, which is caused by a *fault* in the program, which in turn is created due to a *human mistake* (or simply *mistake*). Suppose, for example, we test a spreadsheet by entering some input data and then examining the output, and a discrepancy between the expected and actual results is found. This "observed" discrepancy represents a failure, indicating the existence of one or more faults (e.g., incorrect formulae) in the spreadsheet. These faults were introduced into the spreadsheet by its developer because of human mistakes such as typos or wrong logical deduction. Note that mistakes cannot be determined by solely examining the software, as opposed to observing failures and identifying faults.

Consider, for example, a spreadsheet with a formula that calculates a salesperson's sales commission given his/her total sales s and the applicable commission rate r. According to the company policy, if $s \geqslant \$1{,}500$, $r = 5\%$. Otherwise, $r = 3\%$. Suppose the value of s is stored in the cell F6. The correct formula should be "= F6 * (IF(F6 >= 1500, 5%, 3%))". When $s = \$1{,}500$, the correct output should be \$75 (= \$1,500 × 5%). Suppose the actual formula coded in the spreadsheet is "= F6 * (IF(F6 > 1500, 5%, 3%))" and, hence, the actual output is \$45 (= \$1,500 × 3%). In this case, the miscoding of the arithmetic operator ">" (instead of ">=") in the formula represents a fault, and the discrepancy observed between the expected result (= \$75) and the actual result (= \$45) represents a failure. Unless we check with the spreadsheet developer, it is impossible to determine the underlying cause (e.g., whether a typing mistake or misinterpretation of the company policy in the developer's mind) for making this mistake.

Senders and Moray [30] define an *error* as "an action that is not intended by the actor [a mistake]; not desired by a set of rules [a fault caused by a mistake] or an external observer [a failure]; or that led the task or system outside its accepted limits [a fault leading to failure(s)]". Here, an error is defined in a "coarse" manner that collectively refers to a failure, a fault, and a mistake. Sheridan [31] defines (*human*) *error* as "an *action* that fails to meet some implicit or explicit criterion". If the term "action" is interpreted as "human action", then Sheridan's definition of "error" is similar to the meaning of "mistake" in the IEEE terminology. In this way, the definition of Sheridan [31] is relatively narrow in scope when compared with the definition of Senders and Moray [30]. To further add to the inconsistency, Panko and Sprague [32] state in their paper that "programmers refer to what we call errors as faults".

In this paper, as far as possible, we use the terms "failure", "fault", and "mistake" in accordance with the IEEE terminology. When there is no ambiguity, we also use the term "error" (by following the definition in [30]) to refer to failure, fault, and mistake in a collective manner.

4.2 Different types of testing

Testing is a QA technique in software development, and is often classified into either dynamic or static [33,34]. *Dynamic testing* involves executing the software system with test data, and then verifying the output and operational behavior of the software [34]. *Static testing*, however, refers to the examination and analysis of a work product (e.g., a software system or document). If the work product is a spreadsheet, then static testing does not involve directly executing the formulae or macros of the spreadsheet to compute outputs based on some given test data. Technical reviews, inspections, audits, and desk checking are examples of static testing [35]. Static testing is sometimes called *human testing* when done manually [34], but nowadays automated static analysis tools are also available for spreadsheets. For example, MS Excel provides a built-in automatic static analysis tool that enables a spreadsheet developer to trace precedents and dependents for each cell value. Neither dynamic testing nor static testing is considered sufficient on its own. It is found that dynamic and static testing are largely complementary to each other in terms of the types of faults detected [36].

Table 5 outlines some common static testing techniques based on the IEEE terminology. In this paper, when discussing the relevant spreadsheet QA studies, we follow the definitions in Table 5 as far as possible. For example, *inspection* refers to a QA exercise that takes a formal, team-based approach, while a "one-person inspection" coined by some researchers will be referred to as *desk checking* which is often, but not necessarily, done by the *author* of the work product being reviewed. An *author* refers to the person who generates/develops the work product.

Obviously, the target of dynamic testing is the software program (which is a spreadsheet in the context of this paper). For static testing, however, the scope is broader. Static testing can be applied to a software program or any non-software product (e.g., a software specification document). Since the focus of this paper is on spreadsheets, when we discuss static testing, we assume that it is applied to spreadsheets only. This implies that static testing (same as dynamic testing) is applied after a spreadsheet has been implemented. Because of this, we will discuss static testing and dynamic testing in Section 6.4 (after discussing the "implementation" stage in Section 6.3).

Recall in Section 4.1 the difference in meaning between "failure", "fault", and "mistake". In general, a static testing technique requires *visual* examination of a spreadsheet, including its embedded formulae, for correctness. Thus, static testing is typically focused on *fault* detection. In dynamic testing, the tester executes a spreadsheet, followed by comparing the expected output and the actual output to look for *externally observable* discrepancy between the two outputs. Hence, dynamic testing often targets at *failure* detection. Upon observing a failure, the tester will perform *fault localization* (a debugging task) to identify and locate the faults in the spreadsheet which cause the failure. This will then be followed by fault removal or repair (another debugging task).

Besides, testing can also be broadly classified into functional and performance. *Functional* testing aims at

Table 5 Different types of static testing

Type	Definition
Review	It is a process or meeting during which a work product (e.g., a system or document) is presented to project personnel, managers, users, customers, or other interested parties for comment or approval [26]. Examples of reviews are management reviews, technical reviews, walkthroughs, inspections, and audits [27]. *Remark:* The above definition implies that a QA exercise performed solely by the author is not a review (and, hence, also not an inspection or audit), because it involves no other people.
Inspection	It is a type of review that involves a *visual* and *static* examination of a work product to identify anomalies or defects [27]. The examination is on a *peer* basis, led by an impartial facilitator who is trained in inspection techniques. The climax is a "meeting of the minds" at a participant meeting [34]. In other words, inspection is a *formal, team-based* task. Since people other than the author are involved, inspections are highly effective in finding defects [34].
Desk checking	It *visually* examines code listings, test results, or other documentation, usually (but not necessarily) by the author, to identify problems such as software faults and violations of development standards [26]. Desk checking is argued to be relatively ineffective in detecting software faults because it is an undisciplined process and it lacks the synergistic effect of the inspection team [34]. Another drawback of desk checking is that, if done by the author alone instead of another independent person, the effectiveness of fault detection will be lower because people are generally ineffective in checking their own programs [32,34].
Audit	It is a *systematic*, *independent*, and *documented* process for obtaining evidence and evaluating it objectively for determining the conformance of work products (e.g., source code) and processes to applicable standards, guidelines, plans, specifications, and procedures [26][27]. An audit often involves a lead auditor, a recorder, an initiator, one or more auditors, and the audit organization. *Remark:* By definition, an audit is an "independent" QA exercise. Thus, the author cannot be an auditor. Also, the IEEE terminology does not stipulate that an audit can only occur *after* the system has been released to its users for their use. Some studies (e.g., [37,38]) use the term "field audit" to refer to a QA exercise performed by the authors or other end users for spreadsheets that *are already being used* in organizations. When discussing other relevant studies in this paper, if needed, we will replace the term "field audit" by another more appropriate term that conforms to the IEEE terminology. For example, if the term "field audit" is used to refer to a QA exercise performed by a *single person* (either the author or another person) *after* the spreadsheet has been released for production use to process real-life data, we will call it a "post-release desk checking".

evaluating the system's compliance with its specified "functional" requirements, whereas *performance* testing covers "non-functional" aspects of a system such as efficiency, scalability, reliability, and usability. This paper mainly focuses on functional testing of spreadsheets. In the rest of the paper, we simply refer to functional testing as "testing". By a similar reason, the term "quality assurance (QA)" used in this paper mainly focuses on the "functional" aspect of spreadsheets.

5 Software engineering-based spreadsheet life cycle

Following the growing importance of spreadsheet applications, some researchers (e.g., [18]) proposed to use a software engineering-based approach to spreadsheet development. Some even coined the term "spreadsheet engineering" to emphasize the importance of this development approach [39,40]. Along this direction, the research community generally argues for a life-cycle approach to spreadsheet QA, because mistakes and faults could be introduced at any stage of the development life cycle [41].

We found five studies [41−45] related to the spreadsheet life cycle. With respect to the main focus, these studies can be classified into two categories. The first category includes three studies [41−43]; they used a spreadsheet life-cycle framework to facilitate the discussion on the *existing practices* and *problems* associated with each life-cycle stage. The second category includes the other two studies [44,45]; they used a spreadsheet life cycle as a vehicle for proposing (new) "best practices" that should be used in individual life-cycle stages.

We now look at the first category of studies. Leon et al. [41] proposed their spreadsheet life-cycle framework with the following five stages: (a) planning, (b) design and development, (c) usage, (d) modification, and (e) disposition. The last three stages (c)−(e) are collectively known as "operation". Leon et al. [41] then used their framework to investigate the specific controls that firms implemented to mitigate potential spreadsheet risks throughout a spreadsheet life cycle. They found that, in most firms, neither formal rules nor informal guidelines for QA have been implemented [41]. Panko and Halverson [42] proposed a life-cycle framework that involves five stages: (a) requirements and design, (b) cell entry, (c) drafting, (d) debugging, and (e) operation. It was observed that: (i) error rates varied over the life cycle, (ii) when entering numbers and formulae in cells, spreadsheet developers made many mistakes that they corrected immediately, (iii) developers often spent little effort in initial analysis and design, and (iv) many did not perform systematic error detection even after the drafting stage, except simple checking such as the reasonableness of numbers [46]. Lawson et al. [43] developed a seven-stage (designing, testing, documenting, using, modifying, sharing, and archiving) framework to study spreadsheet "best practices" by comparing "experienced" spreadsheet users with "inexperienced" ones. They found that some practices were more often performed by the most "experienced" developers and users, such as the use of model evaluation techniques (e.g., extreme case testing and the built-in static analysis toolbar for checking formulae and cell references[2]) and commercial static analysis software (which some people refer to as "auditing software").

Next, we turn to the second category of studies [44,45]. Ronen et al. [44] are among the pioneers who proposed a spreadsheet life-cycle framework. Their framework consists of nine stages: (a) problem identification, (b) definition of model outcome/decision variables, (c) model construction, (d) testing, (e) documentation, (f) QA (called "auditing" in [44]) of spreadsheet models and structure, (g) user manual preparation, (h) training, and (i) installation. Stages (g) and (h) are optional if the spreadsheet is designed to be used only by the developer, whereas the other seven stages are mandatory.

[2] In MS Excel, this built-in static analysis toolbar is called "formula auditing" and it provides several checking functions such as "trace precedents" and "trace dependents". Note that the use of the word "auditing" here is not in line with the definition of the term "[software] audit" in the IEEE terminology (see Table 5).

Their framework was mainly introduced for the situation where spreadsheet development is solely responsible by one developer without much involvement from other stakeholders. In practice, some spreadsheet development tasks involve people other than the developer. In this paper, we shall use the term "other stakeholders" to refer to all people, *except* the developer, engaged in spreadsheet development. Read and Batson [45] introduced their six-stage framework (scope, specify, design, build, test, and use), which is applicable to both situations involving and without involving other stakeholders. This framework classifies spreadsheet models into four types: simple, complex, time-critical, and ill-defined. Each of these spreadsheet models is associated with a different set of "best practices" [45].

From Section 6 onwards, we discuss and analyze in detail the major studies associated with each stage of a spreadsheet life cycle. To facilitate our discussion, a "specific" life-cycle framework needs to be adopted. When deciding what framework to adopt, we observed the following.

- Except the framework in [45], other frameworks [41−44] do not explicitly address the situation where other stakeholders are involved in spreadsheet development.
- Despite the above merit, the framework in [45] does not address an early task of spreadsheet development — problem identification. This task is important because if the problems are ill defined, the resulting spreadsheet will be unable to fulfil all the user requirements and expectation.
- After a spreadsheet has been released for use, it will often be maintained when errors are later detected or new user requirements arise. The task of maintenance is either explicitly or implicitly addressed in the relevant stage of four of the above frameworks [41−43,45]. Indeed, in the framework introduced in [44], maintenance is implicit in the "use" stage.

In view of the above observations, we revise and extend the original six-stage framework in [45] by: (a) incorporating "problem identification" into the first stage and renaming it as "problem and scope identification", (b) renaming the "use" stage as "usage and maintenance" to explicitly include the maintenance task, (c) combining the "specify" and "design" stages into one single stage for ease of discussion, followed by incorporating the stage "modeling"[3], and (d) revising the original stage "test" to "testing and debugging"[4]. The resultant framework after these changes will hereafter be referred to as the "extended spreadsheet life-cycle framework", or simply the "extended framework". It consists of five stages: (i) problem and scope identification, (ii) specification, modeling, and design, (iii) implementation (or building), (iv) testing and debugging, and (v) usage and maintenance. Readers, however, are cautioned that in practice many spreadsheet development projects are not actually performed in a highly structured manner by following every stage of the extended framework. For example, many spreadsheet developers quickly go to implementation (stage (iii)) without first spending sufficient effort in problem and scope identification (stage (i)) as well as specification, modelling, and design (stage (ii)) [45].

6 QA issues in each stage of the spreadsheet life cycle

6.1 Problem and scope identification

6.1.1 Problem identification

In this substage, the developer defines the nature of the problem to be solved by the spreadsheet, and also investigates other issues such as: (a) the way the problem is currently solved (if at all), (b) the performance bottlenecks, and (c) the sources of information [44]. This substage is similar to the task "analyzing the existing system" of the waterfall model, and also includes a "make or buy" analysis in which the developer should determine if an existing template can be purchased for the application to be developed. Numerous such templates are available for purchase, particularly in the areas of income tax calculation, rental analysis, and real estate investments.

6.1.2 Scope identification

The developer defines the spreadsheet's objectives and boundaries, and also considers: (a) the level of complexity that is appropriate to meet the spreadsheet's objectives, (b) the input assumptions (i.e., the estimates or forecasts used in the model) and logical assumptions (for determining the model structure), (c) the data requirements in broad terms (which data are required and how to obtain them), and (d) the time and other resources required for model development. In addition, workshops should be organized for: (i) collecting opinions from the stakeholders involved in model development[5], (ii) soliciting consent on the model objectives, (iii) discussing the trade-off between the scope of the model and its effectiveness, (iv) resolving differences in the model requirements, and (v) agreeing on the data required for the model and responsibilities for producing them [45].

6.2 Specification, modeling, and design

6.2.1 Specification

For large software development, a specification is often prepared which describes in detail the function of a software system prior to programming. Most spreadsheets, however, do not have "formal" specifications, and it is unlikely that spreadsheet developers will be avid specification preparers [40]. Thus, the specification substage is often omitted. It is a

---

[3] *Modeling* is defined as "determining the inputs and outputs, and detailing how outputs shall be computed from inputs" [40]. *Modeling* is often integrated with spreadsheet design and programming.

[4] According to the IEEE terminology, testing is a process to check if the system is working the same as it was supposed to do, but testing does not cover the debugging process to fix the detected faults in the system.

[5] Recall that the development life cycle proposed in [45] is also applicable to *team-based* spreadsheet development, as opposed to the framework proposed in [44], which primarily focuses on spreadsheet development by a single developer.

challenge for spreadsheet developers to determine under what situations a specification is essential, cost-effective, or otherwise appropriate. If the specification task is considered necessary, bubble diagrams, calculation tables, and prototypes can be used for specifying spreadsheet models [45].

6.2.2 Modeling

Mental models Developers have mental models of the spreadsheets with which they develop or interact [47−50]. Thus, understanding their mental models will lead to better knowledge of why spreadsheet development is error-prone and enable the development of new tools and techniques that better correspond to spreadsheet developers' cognitive abilities. Three mental models in the developers' minds were proposed: real-world model, domain model, and spreadsheet model [47−50]. When explaining a spreadsheet, the real-world and domain models are prominent, while the spreadsheet model is suppressed. However, when locating and fixing a fault, one must constantly switch back and forth between the domain model and the spreadsheet model. This suggests that we should strive to improve the mapping between these mental models by improving the correspondence between spreadsheet-specific concepts and application/problem domain concepts.

Model building and training In spreadsheet development, model building is seldom performed, possibly due to the lack of time of the developers [51]. One study [52] found that spreadsheet models were often built in an informal, iterative manner, by employees at all ranks. Another study [53] was performed to identify what procedures could reduce mistakes in spreadsheet development. It was found that a six-week training has significantly improved the logical reasoning of trainees, which in turn enhanced their ability to develop competent spreadsheet models. Here we emphasize that effective spreadsheet training is more than just "learning to hit the keys" without conceptual learning [53].

Model-driven or model-related techniques Several model-driven techniques [39,54−62] have been proposed to define a spreadsheet business *model* which will be consistent with its corresponding customized spreadsheet *application*. Also, refactoring techniques were proposed to improve the overall quality characteristics of ClassSheet models [63,64], which will then be embedded in spreadsheets [57,65,66]. This embedding practice aims to close the gap between creating and using a domain-specific language for spreadsheet models and a totally different framework for actually editing spreadsheet data. By combining and adapting the above model-driven techniques and ClassSheet models, as well as situational method engineering, an integrated approach was proposed where structure and computational behavior of a spreadsheet can be specified by a model with a process-like notation based on pre-defined functional spreadsheet services with typed interfaces [67]. An Example-Driven Modeling technique [68] was also proposed to develop spreadsheet models as an alternative to spreadsheet programming. The main idea is to use an example data set to be processed by a machine learning algorithm in order to infer the relationships between input and output, from which a generalized decision support model is generated. Furthermore, the SpreadSheet-based Modeling (SSM) approach was proposed in [69], which provides a simplified representation to edit the spreadsheet models while the underlying metamodels are unchanged.

6.2.3 Design

The design substage involves producing the most effective structure for the spreadsheet model. A good design makes the model easy to use and understand, reduces the likelihood for a developer to make mistakes when using the model, and facilitates detecting faults due to these mistakes [70].

Design rules and formal design practices Six "golden" rules of spreadsheet design were introduced in [45] to make models easier to comprehend and modify, and reduce the risk of errors. These rules are: (a) separating inputs, calculations, and results, (b) using one formula per row or column, (c) referring to the left and above, (d) using multiple worksheets, (e) using each column for the same purpose throughout the model, and (f) including a documentation sheet. In addition to *rules*, a *block structure* for spreadsheet models was proposed in [44]. This structure has five blocks: (a) identification, (b) macros/menus, (c) map of model, (d) parameters/assumptions, and (e) model itself. Two purposes of this structure are to separate parts of a spreadsheet into blocks to reduce the potential for making mistakes, and to clarify the assumptions of the model to users.

Although the above rules [45] and block structure [44] are useful, they are not enough for designing large and complex spreadsheets with good quality. In view of this, a top-down design approach is highly desirable. Based on the notion of data flow diagrams (DFDs) commonly used in traditional systems development, Ronen et al. [44] introduced their spreadsheet flow diagrams (SFDs) to reduce complexity and to encourage a structured, top-down design. They argued that SFDs are more preferred than DFDs because SFDs focus more on modeling relationships (instead of data flow) and provide an effective means to show the algorithm or the underlying formulae of the model [44]. Besides SFDs, state transition diagrams were proposed for defining macros and menus such that each keystroke selection from a menu moves the user to a state represented by a circle in a state transition diagram.

Similar to the work in [44], a structured design approach for reducing spreadsheet development risks was proposed [71]. We noted a major difference between [44] and [71]. In [44], SFDs were proposed to replace DFDs in spreadsheet development because SFDs are relatively more effective. However, in [71], DFDs were proposed to explicitly model and document the links among data cells in different modules during the design substage. It was also reported in [71] that this DFD-based design approach significantly reduced the number of *linking errors* (which are mistakes in coding links between cells in one part of a spreadsheet to another). In addition, a framework was proposed in [72] which divides the variables into different parts of the workbook, with no duplication and linked through formulae. This framework makes the spreadsheet models easy to use, debug, and modify.

Because spreadsheets are inherently free-form, those developers accustomed to solving problems using very

structured and dedicated MS/OR software packages are often confronted with the challenge of using spreadsheets for solving their optimization problems. Some respond to the challenge by devising rules for implementing models that impose artificial structure on spreadsheets [44,71,72]. However, in MS/OR, this artificial-structure approach might occasionally result in spreadsheet models that are difficult to construct/comprehend and less reliable [73]. To address this problem, some guidelines and suggestions for creating more effective spreadsheet models for optimization problems were provided [73]. These guidelines and suggestions include, for instance, avoiding embedding numeric constraints in formulae, and designing formulae that can be copied. To some extent, adopting these guidelines and suggestions are similar to the approach of using the "golden" rules of spreadsheet design as proposed in [45].

A large class of errors can be attributed to the inability to clearly visualize the underlying computational structure and the poor support for abstraction. To alleviate this problem, a multiple-representation spreadsheet was developed [74]. This spreadsheet contains additional representations that allow abstract operations without changing the conventional grid representation or its formula syntax.

Test-driven development (TDD) is reported to be useful for developing better-quality spreadsheets [11,75,76]. TDD is a development methodology which "forces" the developer to consider a system element's design before coding and take small steps when writing software. Built upon TDD, test-driven spreadsheet development (TDSD) was proposed [11,75,76], whose effectiveness has been verified by several case studies. TDSD was found to be easily understood and used, even for those users without prior knowledge of TDD.

Based on the logical/physical theory, there exist four principal components that characterize any spreadsheet model: schema, data, editorial, and binding [77]. A factoring algorithm was developed to identify and extract these components from conventional spreadsheets with minimal user intervention, followed by a synthesis algorithm which assists users in the construction of executable spreadsheets from reusable model components [77].

Support for capturing and sharing the problem-solving knowledge associated with end-user developed software (including spreadsheets) is often lacking [78]. To improve "literate" programming in the spreadsheet domain, the following design techniques were proposed: (a) *chunking* (breaking the code into logical pieces), (b) a dependency-based computational structure (for eliminating the need to order the declarations for execution), and (c) an automatic generation of the table of contents which provides active hyperlinks to each chuck [78].

Design tools Todays, tools are widely used in software development to improve design productivity and effectiveness. Thus, tools for supporting spreadsheet design are highly desirable. One such tool was proposed in [79] which ensures that the developed spreadsheets have a compatible structure and follow concrete standards. This tool requires the developer to firstly identify and state unambiguously the spreadsheet's overall objective, followed by refining the design into three parts: (a) input data and assumptions, (b) applicable model, formulae or technique for data analysis/processing, and (c) outcome and results. Each part is then decomposed into a lower level of details. The refinement strategy associated with the tool facilitates structuring a spreadsheet into different sections: introduction, data and assumption, model, analysis, macro, and possibly some other sections such as data-entry screen and database lookup. Another tool was also developed which incorporates a block-based formula editor (known as XLBlocks) to replace the default text-based formula editor [80].

6.3 Implementation

6.3.1 Implementation planning and practices

Implementation involves actual coding of the model. Many spreadsheet development projects started their life cycles with this stage, and only little effort was spent on the previous stages (i.e., the Problem and Scope Identification stage and the Specification, Modeling, and Design stage) [45]. Also, many spreadsheet developers rarely did much planning before they started filling the cells in a spreadsheet [37,46,52]. Such a practice likely generates several problems: (a) an unnecessarily complex model which takes longer time to build, (b) assumptions made that were not part of the original intention, and (c) a lack of common understanding of what the model is doing.

A visual description language (CogMap) was developed for spreadsheets to address problem (a) [81]. In essence, CogMap is a simple yet cognitively effective tool for the visual expression of simple assertions about the structure, function, or any other propositions associated with a spreadsheet. A technique for the bidirectionalization of spreadsheet formulae was also developed to allow users to trace backwards for a chain of calculations, e.g., in "what-if" scenarios where the user changes the output and inspect how it possibly affects the input cells [82]. Yet another technique for understanding and inferring units (e.g., dollars, metres, and kilograms) in spreadsheets was developed in view of the fact that many spreadsheet cells do not carry unit information [83]. This may cause a class of spreadsheet faults where calculations (without proper unit transformation) are performed on cells with different units.

Implementation can be done by individuals or in teams. In this regard, the performance between individual and team (of 3 persons) development was also investigated. Compared with individual development, the quality of team-developed spreadsheets was empirically more superior as the percentage of faulty spreadsheets fell from 86% to 27%, and the percentage of cells containing faults fell from 4.6% to 1.0% [84].

6.3.2 Coding mistakes, knowledge, and techniques

It was found empirically that most mistakes end users made in coding spreadsheets were due to poor strategies and attention problems (e.g., overload of working memory or paying attention to the incorrect part of the spreadsheet) [85]. Also, many lookup functions used in spreadsheets were problematic [86]. Furthermore, end users who are more knowledgeable on spreadsheets developed better quality spreadsheet applications

[17]. Thus, there is a need to develop metrics of the developer's expertise on spreadsheets [87].

When coding spreadsheets, decreasing the degree of complexity in formulae will improve the accuracy of a spreadsheet [70]. Specific data and logic validation as well as coding controls through the use of @NA, @IF, and @ERR functions, crossfooting totals, and mechanisms for interactive error feedback should be used where applicable [88]. Also, structured programming techniques, consistent and meaningful names for data ranges (instead of hardcoded cell addresses) and files, macro-controlled templates to avoid user tampering of the process logic, and menu-based macros for printing reports are recommended. In [89], an approach was proposed to generate *elastic sheet-defined functions* (*SDFs*) that work over inputs of arbitrary size. The purpose of SDFs is to facilitate reuse by allowing spreadsheet developers to write code once, rather than repeatedly.

6.4 Testing and debugging

6.4.1 Role of testing and debugging

In the past, few corporations had maintained effective controls to deal with spreadsheet faults. After the financial reporting scandals at Enron, the U.S. Congress passed the Sarbanes-Oxley Act in 2002. Many countries/regions have followed with similar legislation. Consequently, some large corporations have started to look at how they were using spreadsheets in several critical business functions and to put more attention to spreadsheet testing [14,36]. Leon et al. [90] even argued that planning and implementing spreadsheet controls in corporations should go beyond regular compliance.

Kruck [70] contended that increasing the extent of testing and debugging will improve spreadsheet accuracy. The importance of spreadsheet testing was also emphasized in [91,92]. Although most spreadsheet practitioners and researchers generally agreed on the importance of testing (e.g., desk checking, inspection, static analysis tools, and comprehensive test cases), the findings by Caulkins et al. [9] seemed to contradict this mainstream opinion. Their interviews found that a significant minority of respondents believed *informal* quality control procedures were generally sufficient because the "human in the loop" could detect any gross errors. Also, an online survey [41] found that only 24% of the respondent companies had independent QA groups, and only 7% of respondents had used third-party static analysis software. The first part of this finding indicates that QA exercises (particularly desk checking), if any, were most likely done by the spreadsheet developers themselves.

6.4.2 Spreadsheet reuse

One way to lower spreadsheet development costs is reuse. A challenge of reusing spreadsheets is the inadequate support for comparing spreadsheets which are descended from a common ancestor or are "siblings" (generated from templates and later repeatedly instantiated). Hence, it is often difficult to choose which spreadsheets to reuse. To deal with this issue, an algorithm was proposed in [93] to identify the differences between two spreadsheets. Similarly, a method (based on heuristics) and an associated tool were proposed in [94] to analyze the differences between related spreadsheets.

6.4.3 Contingency factors and test planning

Some researchers (e.g., [95,96]) argued that the traditional *validation* methodology was primarily developed for use by expert system builders in the context of large and complex software, which is not the case in spreadsheet development. Thus, this traditional methodology needs to be augmented so that appropriate validation can be performed by non-technical end users. To this end, a validity framework for DSS (including spreadsheets) was developed, and this framework involves fives types of validity: general, logical, interface, data, and system builder [96]. Also, several contingency factors were found that influence the process of validating DSS (including spreadsheets). These factors include: importance/complexity of the decision, life span of DSS (or spreadsheets), budget, abilities of owner/system builder, and organizational factors [96]. Another framework was proposed in [97,98] for initially estimating the required validation effort to ensure that a developed spreadsheet is valid. This framework considers several contingency factors such as risk, complexity, significance, system builder competence, contentiousness, and deadline.

6.4.4 Static testing

Testing effectiveness Compared with inspection, desk checking was reported to be relatively low in fault-detection effectiveness — only about 10% [32,99]. People were generally overconfident on the fault detection effectiveness of desk checking, despite the finding that some types of faults are especially difficult to detect by desk checking when compared with inspection [12]. To improve the effectiveness of desk checking, a general-purpose protocol (with the use of two commercial static analysis tools: XL Analyst and Spreadsheet Professional) was developed [10,100].

In [38], a conceptual model of the factors which affect the performance of fault detection was developed. These factors can be classified into four categories: individual (e.g., domain experience and skill of the reviewer), presentation (e.g., whether the spreadsheet is reviewed on paper or on screen), nature of fault (e.g., type or complexity), and external (e.g., time pressure and desired accuracy). An experiment was also performed to investigate whether the presentation factor was indeed significant with respect to the number of faults detected. It was found that: (a) overall the reviewers detected only about half of the errors, (b) the experimental subjects who checked on paper detected more faults, but took a longer time to finish, than those who checked on screen, and (c) checking spreadsheets which had the formulae displayed or printed (even when printed alongside the corresponding cells than separately as a linear list) did not detect significantly more faults than those with no formulae available. When no formulae were available for checking, this kind of checking effectively targeted at detecting failures.

Static testing techniques and analysis tools MS Excel provides a built-in static analysis toolbar called "formula auditing" (note here the deviation of the term "audit" from the IEEE terminology) for the developer to trace precedents and dependents for each cell value. Using such a tool, however, would be tedious and time consuming for spreadsheets with

many worksheets and many linked cells [101]. To alleviate this problem, many *automatic* approaches for detecting spreadsheet faults have been proposed [18,102−112]. Often supported by their associated tools, these approaches target at detecting specific types of faults by various heuristic decision algorithms. For example, UCheck [113−115] and Dimension [116,117] target at assigned "data types" of input cells and formulae. These data types are derived from the text values of header cells that are positioned in the same row or column as the related input cells. Some other tools focus on cell arrays (e.g., AmCheck [118], CACheck [119], and EmptyCheck [120]) or use adaptive learning to detect anomalies in formula cells (e.g., CUSTODES [121], ExceLint [122], and WARDER [123]). Rather than focusing on spreadsheet faults, an algorithm was proposed in [124] to detect a particular type of mistake (data cloning) — formulae whose values are copied as plain text in a different location of a spreadsheet. Furthermore, an Excel add-in tool, CheckCell [125], was developed to automatically find potential data errors (this tool locates data that has a disproportionate impact on the spreadsheet computation).

However, some researchers (e.g., [126,127]) argued that the above automatic detection approaches are not effective as they only focus on specific fault/mistake types. To address this problem, supervised machine learning algorithms were developed to obtain fault predictors that utilize all data provided by multiple spreadsheet metrics from their "extensive" catalog [126]. Experiments showed that these fault predictors were effective for a wide range of faults [126]. To complement the work in [126,127], we noted six more studies [128−133] which have used the concept of *smelly formulae* (i.e., spreadsheet formulae with potential faults) and/or a list of metrics, as well as detection/refactoring techniques to resolve these smelly formulae. In addition to smelly formulae, other studies [101,134,135] proposed a visualization approach to detecting spreadsheet faults. For example, a desk-checking approach (supported by its associated tool LinkMaster) was proposed which provides useful, visual cues on individual worksheets for finding *linking faults* (incorrect references to spreadsheet cell values on separate work areas) [101].

Two studies [136,137] were conducted to: (a) investigate the different kinds of spreadsheet mistakes, (b) determine the effectiveness of two static analysis tools (the Excel's built-in "Error Check" function and the add-in "Spreadsheet Professional" tools) for detecting mistakes, and (c) compare these tools with manual desk checking. One finding in [136] was that the performance of both static analysis tools was very poor for each category of spreadsheet mistakes. This finding contradicted with another study [100], which reported that the performance of "Spreadsheet Professional" was more superior to manual desk checking. Another finding in [136] showed that static analysis tools for spreadsheets have different fault detection effectiveness. To allow a systematic and objective comparison among these tools, the Formulae, Formats, Relations (FFR) model was developed in [138], serving as a basis for describing and comparing different static analysis tools.

6.4.5 Dynamic testing

Dynamic testing techniques Besides static testing, dynamic testing is also a common spreadsheet QA approach. Numerous dynamic testing techniques were developed to improve the effectiveness of dynamic testing for spreadsheets. Below we outline some of these techniques.

In [139], metamorphic testing (MT) was proposed for detecting spreadsheet failures, and experiments showed that MT is highly effective for such detection [140]. As follow-up work, two failure detection techniques (error trapping (ET) [141] and MT) were compared [20]. Experiments found that neither technique is sufficient in spreadsheet testing [20]. Rather, ET and MT are complementary and should be used together in spreadsheet testing whenever possible.

The "What You See Is What You Test (WYSIWYT)" methodology [13,142−144] is a well researched dynamic testing technique. Based on a definition-use test adequacy criterion [145], this methodology helps users locate faults and highlight potential discrepancies [104]. Another work [146] was conducted to integrate automated test generation into the WYSIWYT methodology to support incremental testing and provide immediate visual feedback. Furthermore, a technique for generating test cases was developed, together with its associated tool, based on backward propagation and solution of constraints on cell values [147].

While most dynamic testing techniques focus on checking *numerical* data, another study [148] took a different approach. In [148], a user-extensible model for *string-like* data (e.g., names) was developed to distinguish at runtime between invalid data, valid data, and questionable data that could be valid or invalid.

Testing effectiveness The WYSIWYT methodology described above uses data-flow concepts. Several empirical studies (e.g., [149]) compared the performance of data-flow and mutation testing. These studies found that mutation-adequate tests detected more faults. Inspired by this finding, a suite of mutation operators was developed to support applying mutation testing for spreadsheets [150]. These mutation operators can also be used for evaluating the effectiveness of dynamic testing tools and debugging tools for spreadsheets.

Test case construction To perform dynamic testing, spreadsheets are executed with test data so that the output results can be checked against the expected results. The comprehensiveness of test data will have a profound effect on the effectiveness of spreadsheet testing. Thus, a "good" set of test data should be constructed to validate the logic of spreadsheets before release to users [88]. In this regard, the MT technique [20,139] discussed above provides an effective and systematic means to generate a "good" set of test cases for dynamically testing a spreadsheet. In addition, an algorithm for splitting spreadsheets into smaller logically connected parts (called *fragments*) was developed [151]. The split fragments can then be individually tested. Studies [151] found that this approach significantly reduces the effort to generate test cases for testing a spreadsheet.

6.4.6 Spreadsheet error classifications and taxonomies

Many studies have developed classifications or taxonomies of

spreadsheet errors. The study in [44] is one of these pioneer works. In [44], although the classification scheme includes eight error types, it is not fine-grained enough because: (a) some of these error types are mistakes (e.g., incorrectly copied formulae) and some of them are faults (e.g., incorrect ranges in formulae and incorrect cell references), and (b) one "error" type called confused range names is in fact undesirable practice potentially contributing to confusion rather than faults that directly cause failures. Another work [152] distinguished between two classes of errors: *domain error* (e.g., a "mistake" in logic due to misunderstanding of the depreciation concept in accounting) and *device error* (e.g., a "fault" involving a wrong reference in the depreciation function). Similar to the work in [44], the error classification in [152] does not differentiate between mistakes and faults.

With respect to spreadsheet faults, the work in [153] distinguished between location fault and formula fault. *Location fault* refers to a fault in formulae that are conceptually correct but one or more of their cell references are wrong, while *formula fault* refers to a misuse of operators or wrong number of operators. On the other hand, two studies [42,137] developed a taxonomy specifically focused on spreadsheet mistakes (Fig. 1). They defined *quantitative mistakes* as those which cause "immediate" incorrect output from spreadsheets, whereas *qualitative mistakes* are those which do not result in "immediate" wrong output but, rather, they often represent poor design and coding practices (e.g., putting a constant instead of a cell reference into a formula). Such a definition of qualitative "mistakes" is debatable, because they do not necessarily produce failures. Instead, they are poor spreadsheet modeling practices and, hence, are strictly speaking not mistakes. Note that, in [154], *latent errors* are defined as those that "do not directly cause the error and occur upstream of the event". Thus, latent errors defined in [154] are obviously of the same in nature as that of qualitative "mistakes" defined in [42,137].

Along with this taxonomy, the study in [155] included two more "mistakes": *jamming* (values of more than one variable are placed in a single cell) and *duplication* (information of a variable is duplicated in the spreadsheet, possibly resulting in data inconsistency). Here, similar to qualitative "mistakes" defined in [42,137], classifying jamming and duplication (which are unrecommended practices) as mistakes is controversial. Similarly, the study in [156] developed a taxonomy to classify the qualitative "mistakes" in Fig. 1 into four subtypes: formula integrity, semantics, extendibility, and logic transparency (see Fig. 2). With respect to the qualitative "mistakes" and quantitative mistakes in Fig. 1, an experiment involving desk checking was performed [157]. This experiment found that: (a) quantitative mistakes were more easily detected than qualitative "mistakes", and (b) the detection rate depended on the type and prominence of mistakes (e.g., whether the mistakes were conspicuous) as well as prior incremental practice with spreadsheet error detection [157].

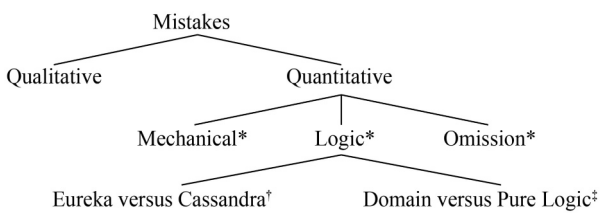


Fig. 1 Taxonomy of spreadsheet mistakes (adapted from [137])

* *Mechanical mistakes* are typing mistakes, pointing mistakes, and other simple slips. *Logic mistakes* are incorrect formulae due to choosing the incorrect algorithm or creating the incorrect formulae to implement the algorithm. *Omission mistakes* are things left out of the model that should be there, and they are often caused by a misinterpretation of the situation.

† *Eureka mistakes* are easy-to-prove logic mistakes. *Cassandra mistakes* are logic mistakes that are difficult to prove even if detected.

‡ *Domain logic mistakes* are caused by the lack of domain knowledge of the developer. *Pure logic mistakes* are caused by a lapse in logic.

6.4.7 Impact of spreadsheet calculation paradigms

It was argued in [158] that a cause of spreadsheet errors is the low conceptual level of spreadsheets, e.g., lack of abstraction and modularity mechanisms. This study [158] compared two spreadsheet calculation paradigms: *traditional* (e.g., Excel) versus *structural*. Structural spreadsheet calculation paradigm is at a higher conceptual level that utilizes goals, plans, and spreadsheet data structures in computation [159]. The study [158] found that the two paradigms produced different error behaviors.

6.4.8 Impact of spreadsheet faults

In [160], the financial impacts of faults in 25 operational spreadsheets were investigated. The study identified 117 genuine faults, in which 70 (60%) of them had financial impact and the remaining 47 (40%) had not. The largest financial impact of a fault was over $100 million, followed by several faults whose impact exceeded $10 million each. A similar study [161] was performed to investigate the impact of spreadsheet errors in an Irish healthcare setting, and found that more than 90% of the spreadsheets studied were faulty and the cell-error rate was 13%.

6.4.9 Debugging

Debugging effort and performance Some researchers (e.g., [162]) argued that the human-computer interface (HCI) literature tended to emphasize the strengths of spreadsheets (e.g., quick gratification of immediate users' needs), but often neglected their weaknesses, one of which being the difficulty in debugging. Some other researchers (e.g., [46]) reported that users did not spend much time in a separate debugging stage,

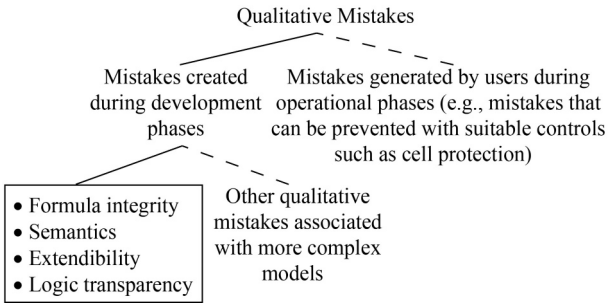


Fig. 2 Taxonomy of qualitative mistakes created during spreadsheet development (adapted from [156])

thereby reducing the effectiveness of this task.

With respect to the performance and behavior of expert and novice end users in spreadsheet debugging, an experiment [163] has the following important results. First, on average, the debugging performance of experts (72%) was 14% better than novices (58%), and this difference was statistically significant, especially for formula faults. Second, regarding the debugging behavior of both types of users in terms of coverage per cell, formula cells took precedence over data cells, with a greater percentage of expert users looked at each cell. Also, summation formula cells and bottom-line value cells attracted more attention than other formula cells. Furthermore, formula cells generating *text* values were not checked as much as formula cells generating *numeric* values. Third, using a debugging tool that gives feedback on cell coverage resulted in slightly higher debugging performance (62% for the experimental group versus 59% for the control group). While the difference in debugging performance was not statistically significant, the cell coverage feedback provided by a debugging tool has greatly increased cell-checking rates.

Debugging strategies, models, and techniques For effective debugging, the task should be well supported by associated strategies, models, and techniques. In view of this, a sensemaking model was derived to analyze how end users went about debugging their spreadsheets [164,165]. The study in [164] found the dominance of information foraging[6], which occupied half to two-thirds of participants' time. This study also identified *selective* (striving for efficiency by following contextually salient cues) and *comprehensive* (seeking a more complete understanding of the problem) styles as two successful strategies for traversing the sensemaking model. The above findings thus revealed important implications for the design of spreadsheet tools to support end-user developers' sensemaking during debugging.

Model-based software debugging (MBSD) is a well-known technique for fault localization in software written in common programming languages such as C++ and Java. One study [166] empirically analyzed and compared the use of three MBSD models in locating spreadsheet faults: the value-based model, the dependency-based model, and the novel dependency-based model [167]. Since this study [166] found that each of the three models has its merits and drawbacks, thus combining these three models to produce a more effective bug diagnosis partitioning technique should be considered. Another study [168] introduced a model-based diagnosis for spreadsheet debugging by translating the relevant parts of a spreadsheet to a constraint satisfaction problem for performing further reasoning (e.g., about input propagations).

Fault localization is an important task for debugging. We noted several studies on fault localization. The first study [169] investigated three fault localization techniques: blocking, nearest consumers, and test count. This study found empirically that the two individual factors, information base and mapping, both had significant impact on the effectiveness of these three techniques. The second study [170] proposed several ways to combine reasoning from UCheck [113−115] and WYSIWYT [13,142−144] for locating faults and mapping the fault information visually to the users. This combined technique was reported to be more effective than either technique alone [170]. The third study [171] investigated the impact of different similarity coefficients on the accuracy of spectrum-based fault localization (SFL, which is a software debugging approach originally developed for traditional procedural or object-oriented programming languages) applied to spreadsheets. This study [171] identified three of the 42 studied similarity coefficients as the best ones to diagnose spreadsheets with SFL. The fourth study [172] proposed a fragment-based spreadsheet debugging approach, based on the rationale that after partitioning a spreadsheet into fragments, users find it easier to assess the correctness or faultiness of each smaller fragments than the entire spreadsheet.

Debugging tools A debugging tool, called GoalDebug, was developed to propagate the expected change from the old, incorrect value to the new, expected value backward over the formula to its arguments and referenced cells to derive formula changes, as well as generates change suggestions for the developer [173]. Similarly, an interactive debugging tool, called EXQUISITE, based on artificial intelligence was developed [174]. In addition, the role of visualization tools in spreadsheet debugging and fault detection was studied in [175]. It was reported that developers using a visualization tool that facilitates chaining to trace formula cell references were significantly faster in identifying and correcting linking faults, compared to developers without using the tool.

Furthermore, two interactive tools for detecting data dependencies in a spreadsheet were developed and evaluated [176]. One of them is an online tool which displays flowchart-like diagrams, while the other tool interactively depicts cell dependencies directly on the spreadsheet. It was found that users generally preferred the second tool though both tools were considered helpful [176]. Recently, a study [177] reported that, although debugging tools are useful, over-reliance on tools might exist and reduce fault identification performance. The reason is that users of debugging tools might focus solely on potential problems pinpointed by the tools, thus paying less attention to other parts of a spreadsheet.

6.5 Usage and maintenance

Generally, users of enterprise (non-spreadsheet) applications do not need to know the tools or programming languages used for developing the applications. By contrast, effective use of a spreadsheet by an end user (including the management) who is not the developer appears to require substantial spreadsheet knowledge [17]. Thus, problems with the quality of spreadsheet applications can be partially offset by the end user's spreadsheet knowledge [17]. In this regard, a training program should be developed to raise the effectiveness of

[6] *Information foraging theory* explains how strategies and technologies for information seeking, gathering, and analysis are adapted to the information collected from the environment. The theory assumes that people, when possible, will modify their strategies or the structure of the environment to maximize their rate of obtaining valuable information [165].

using and accepting end-user applications, including spreadsheets [178]. It was also reported that companies generally offered more user trainings to high-risk spreadsheet applications [19].

Some studies [45,88] recommended to employ specific controls in the use of spreadsheets, such as to: (a) document processing logic in detail to cater for future need of reconstructing the formulae, (b) prepare user manuals that include descriptions of the general context, input, processing, output, backup and recovery procedures, (c) timestamp outputs to safeguard data integrity, (d) formulate explicit error detection and handling procedures, and (e) implement security measures to prevent data loss.

Like most conventional software, spreadsheets are subject to software evolution or maintenance. However, this task is often not supported by version management tools, causing the introduction of new mistakes or faults during evolution. In view of this problem, a spreadsheet corpus (known as VEnron) was developed to provide useful version information on spreadsheet evolution [179]. Also, during the maintenance phase, end users often need to modify a large (legacy) spreadsheet developed by others and whose functionality they do not understand. To address this problem, several techniques were proposed in [180], including: (a) reverse engineering techniques for deriving relational models from existing spreadsheets, and (b) data mining techniques for reasoning about spreadsheet data and inferring functional dependencies among columns (these dependencies being the building blocks to infer the spreadsheet business model as a database schema).

7 Analysis, discussions, implications, and future research

7.1 Analysis and discussions

We call the 122 selected journal and conference papers related to spreadsheet QA the "primary" papers, and the other relevant ones listed in the References Section the "secondary" papers. We counted the number, P, of the primary papers as well as the number, T, of both primary and secondary papers related to the overall spreadsheet life cycle. We also performed similar counts for papers on individual spreadsheet life-cycle stages or substages. A paper may be counted more than once if it addresses multiple issues or areas. Table 6 shows all these P and T values.

Overall, at the level of life-cycle stages, the values for P range from 1 to 81, and the values for T range from 2 to 107. We found that the life-cycle stages "Problem and Scope Identification" and "Usage and Maintenance" have the smallest P and T values. Consider the "Problem and Scope Identification" stage. There are two plausible reasons contributing to its small P and T values. First, Ronen et al. [44] argued that problem identification in spreadsheet development is similar to the task "Studying Existing Systems" in the traditional waterfall model used for implementing non-spreadsheet applications. As problem identification is largely independent of the development platform, existing studies related to problem identification for implementing non-spreadsheet applications are likely to be applicable to spreadsheet development. Consequently, the demand is not high for additional studies that specifically address problem identification for spreadsheets. Second, unlike professional programming where users and developers are different groups of people, there are often the scenarios where users undertake the spreadsheet development work for themselves. Obviously, for end users, problems and requirements are both more easily understood (because the problems and requirements are their own) and more likely to change (because end users may need to negotiate such changes only with themselves) [16,181]. Thus, problem and requirement identification for spreadsheet development are relatively more implicit. The above two reasons may result in fewer studies related to the "Problem and Scope Identification" stage.

Table 6 P and T values for the spreadsheet life cycle and its major stages

	Values of P	Values of T
Spreadsheet life cycle	6	13
Problem and scope identification	1	2
Specification, modeling, and design	26	37
Implementation	13	17
Testing and debugging	81	107
Usage and maintenance	6	7

In Section 6.5 above, we discussed seven studies [17,19,45,88,178−180] related to the "Usage and Maintenance" stage. We identified from them several spreadsheet QA techniques and controls such as input and output controls, documentation, backup and recovery, user training, and change/version management. We noted that not only very few of our collected papers have addressed the relevant techniques and controls of this stage, but even for four of these few papers [17,19,45,88], they just briefly touched on the above techniques and controls without in-depth discussion.

The "Testing and Debugging" stage has the largest values of P (= 81) and T (= 107), even much larger than the P (= 26) and T (= 37) values of the "Specification, Modeling, and Design" stage. Our further investigation of the "Testing and Debugging" stage found that, overall, the three most popular research areas are: static testing (P = 30, T = 44), dynamic testing (P = 10, T = 13), and debugging (P = 17, T = 20). For each of these three areas, we also counted the numbers and the percentages of relevant studies in two categories: technical-oriented IT research and business-oriented IS research. The results are shown in Table 7. We observed that, across all the three areas, the numbers and the percentages of relevant studies in the technical-oriented IT category are much larger than those in the business-oriented IS category. A major reason is that there have been many technical-oriented studies on the development of methodologies, techniques, and tools for these three areas from the software engineering community. Whereas among the relevant studies in the business-oriented IS category, most of them primarily focused on the performance evaluation (via experiments and case studies) and application aspects of testing and debugging.

We also noted an interesting observation regarding the area "Spreadsheet Error Classifications and Taxonomies" (see Section 6.4.6). Among the eight relevant papers (P = 7, T = 8), two of them [44,153] came from technical-oriented IT

Table 7 Distribution of relevant studies among static testing, dynamic testing, and debugging

	Technical-oriented IT research	Business-oriented IS research
Static testing	35 (80%)	9 (20%)
Dynamic testing	9 (69%)	4 (31%)
Debugging	17 (85%)	3 (15%)

journals, four of them [42,152,155,156] from business-oriented IS journals or conferences, one of them [137] from a dual-focused IT/IS journal (*Decision Support Systems* (*DSS*)), and the remaining one [157] from an MS/OR journal (*Omega*). Furthermore, the two papers, one from *DSS* and the other from *Omega*, were written by IS researchers. Thus, overall, the majority of papers related to spreadsheet error classifications and taxonomies are in the business-oriented IS category. We assert that most IS researchers pay relatively less attention to the IEEE terminology when compared with software-engineering researchers. This could be a reason why many IS researchers have not differentiated among mistakes, faults, and failures in their error classifications and taxonomies. Although their practices may be appropriate in their own study contexts, it would be desirable for future studies on this area to follow the well-defined IEEE terminology in order to avoid potential confusion and to generate a greater impact in the spreadsheet community.

7.2 Implications and suggestions for future research

In Section 7.1, it is mentioned that the values of P and T are relatively small for the life-cycle stage "Problem and Scope Identification". It is also mentioned that one plausible reason is that problem identification is not much different between spreadsheet and non-spreadsheet developments [44] and, hence, implying that new studies on this life-cycle stage is not in big demand. This reason is subject to argument. For example, in problem identification of spreadsheet development, a "make or buy (existing template)" analysis may be involved [44] and this analysis may not exist in other non-spreadsheet development projects. Hence, it is advisable that more studies on problem identification specifically related to spreadsheets should be performed.

Also, our analysis in Section 7.1 found that only very few studies are related to the "Usage and Maintenance" stage. Thus, we recommend that more future studies on areas (e.g., input and output controls, documentation, backup and recovery, user training, and change management) related to this stage should be conducted. For example, with respect to user training on spreadsheets, several issues or questions are worth exploring: What modes (e.g., in-class, online, on-the-job, and help-desk) should be used for cost-effective spreadsheet training? With time constraint (e.g., a half-day training), what should be the training approaches and focuses (e.g., technical versus application-domain knowledge, model building, static and dynamic testing, testing analysis tools, debugging, and which kinds of cells, blocks, or structures in the spreadsheet should be spotlighted) to achieve the best outcome in effective usage and maintenance?

Our observation that the P and T values of the "Specification, Modeling, and Design" stage are much smaller than their corresponding P and T values of the "Testing and Debugging" stage also have an important implication. The former stage involves developing spreadsheets with higher quality and fewer faults (i.e., fault prevention), whereas the latter stage mainly focuses on fault detection and removal. As prevention is better than cure, more future research work should be performed to develop new and promising strategies, approaches, and techniques for spreadsheet specification, modeling, and design, as well as how to apply them effectively in the industry.

Section 6.2.3 has discussed the application of TDD to spreadsheet development [11,75,76]. TDD has been advocated by many software researchers and practitioners in *non-spreadsheet* domains for developing quality software. TDD is an iterative development approach with a "test early and continuously" attitude, so that developers need to design applications with testing in mind [182−184]. TDD offers many benefits. For example, it can help developers catch errors well before testing commences and also help clarify design. Despite these merits, we have not noted any studies other than [11,75,76] (which were published by the same group of researchers) to apply TDD to spreadsheet development. Thus, a strong need exists for investigating and exploring this issue in future research studies. A key concept in TDD is that design and testing should not be two distinct tasks to be performed separately; rather there is a relationship between these two tasks. Following this logic, it is also worthwhile to further strengthen and extend the few existing studies (e.g., [71]) on which spreadsheet errors are likely to be reduced by each design practice or technique.

We also note an important finding related to dynamic testing of spreadsheets. Although some techniques (e.g., the WYSIWYT methodology [13,142−144], an automated test case generation based on backward propagation and solution of constraints on cell values [147], and metamorphic testing (MT) [20,139]) have been proposed to systematically generate test cases for a more comprehensive spreadsheet testing, studies [91,92] found that many spreadsheet developers often work with a number of example inputs they use during development. A plausible reason is that spreadsheet users are not willing to invest time in the specification of test cases [168]. Thus, more work needs to be done to educate spreadsheet developers and to promote the use of more systematic test case generation techniques for spreadsheets. Also, current spreadsheet environments do not support the "explicit" management of test cases [168]. To deal with this issue, the EXQUISITE framework was developed with a feature of test case management [168]. Through this framework, defining test cases can be done directly in the spreadsheet. In view of this issue, we suggest that more tools should be developed for test case management in the spreadsheet environments.

Nowadays, with the advent and popularity of the Microsoft Office Suite, a growing number of more sophisticated spreadsheets are integrated with Visual Basic for Applications (VBA) [185]. This integration brings in two new sources of affecting the quality of spreadsheet outputs — the VBA module and its interface with the spreadsheet. Similarly, Excel

spreadsheets can now be integrated with the Power BI platform for providing enterprise BI capabilities [186]. As a result, the issue of spreadsheet quality is no longer restricted to the spreadsheet itself, but is also extended to its interface with other modules or platforms. Thus, a more holistic view of spreadsheet quality is necessary.

Although spreadsheets provide data analysis and computation capability, we noted that in some situations, spreadsheets are primarily used as a data repository [187−190]. For example, in [190], spreadsheets were used as a universal data repository to unify the heterogeneous data sources in a distributed enterprise environment. In [191], Excel was compared with Access (another Microsoft software product) in terms of their relative strength for storing large amount of data. It was argued in [191] that if the goal is to maintain data integrity in a format that can be accessed by multiple users, Access is a better choice. On the other hand, although spreadsheets have some data analysis functions, Broman and Woo [192] recommended to restrict spreadsheets to data entry and storage only, while analysis and visualization should be handled by a separate program. This practice helps reduce the chance of contaminating or destroying the raw data in the spreadsheet. In view of the use of spreadsheets for data storage, several data-related issues are to be considered, such as data quality, data integrity, data integration, data interoperability, and a well-defined data schema. More research work along this direction should be conducted.

The above major discussions in Section 7 are summarized in Table 8.

8 Conclusion and limitations

Spreadsheets will continue to be a popular implementation platform for many decision support systems, intelligent systems, and expert systems in various application domains. Although developing and using spreadsheets are handy, its inherently free-form development style may result in many mistakes and, hence, faults to be introduced at various stages of the development life cycle. Thus, to effectively address the

Table 8 Major observations, possible reasons, and recommendations

Observations	Possible reasons	Recommendations / future research directions
Problem & scope identification		
● Small number of relevant studies.	It is generally perceived that: ● Problem identification is independent of the development platform; thus existing studies on identification in the non-spreadsheet domains are applicable to the spreadsheet domain. ● End users often undertake the spreadsheet development work themselves. Thus, problem & requirement identification are more implicit.	● Many problem identification techniques were originally developed for the "traditional" Waterfall development methodology. To some extent, advocating these relatively "formal" techniques to spreadsheet development may be viewed as a step backwards [44]. It is worthwhile to investigate & develop some techniques for improving the effectiveness of problem identification, but without incorporating "unnecessary" controls or burdens that would undermine the benefits & flexibility of spreadsheet development by end users. ● Not every spreadsheet supports only one user. For those spreadsheets with multiple users whose requirements are different, a well-executed problem & requirement identification process is still needed. ● In problem identification of spreadsheet development, a "make or buy (existing template)" analysis may be involved [44] & this analysis may not exist in other non-spreadsheet development projects. Thus, more studies on problem identification specifically related to spreadsheets are still needed.
Specification, modelling, & design		
● The number of relevant studies related to this life-cycle stage is only about one third of the number of studies related to the "Testing & Debugging" stage.	● Few spreadsheet developers are trained in software engineering. Thus, they may not be aware of or realize the importance of this life-cycle stage. This may in turn lead to few research studies related to this stage.	● As a first step, spreadsheet researcher and practitioners, as well as those teaching IT/IS in universities/colleges, should put in more effort to educate people about (and how to perform) the importance of spreadsheet specification, modelling, & design. ● Specification, modelling, & design involves developing spreadsheets with higher quality & fewer faults, whereas testing & debugging mainly focuses on fault detection & removal. As prevention is better than cure, more future research work should be performed for spreadsheet specification, modelling, & design.
● Some studies which advocate the application of test-driven development (TDD) to spreadsheet development have started to emerge.		● The merits of having a "test early and continuously" attitude is well known in non-spreadsheet domains. TDD supports this attitude. Thus, it is intuitive to apply TDD to spreadsheet development. We recommend that more work should be done on this area.
Testing & debugging		
● Technical-oriented IT studies focused on the development of methodologies, techniques, & tools. On the other hand, business-oriented IS studies mainly focused on the performance evaluation (via experiments & case studies) and application aspects of testing & debugging.	● Possibly due to the difference in research training & expertise between technical-oriented IT & business-oriented IS researchers.	● There is an opportunity for research collaboration between IT & IS researchers. Once a methodology, technique, or tool is developed by IT researchers, IS researchers could conduct performance evaluation of that developed methodology/technique/tool in a business setting (possibly via the IS researchers' established business networks).
● The number of studies on static testing is more than double than that on dynamic testing.		● Most studies on spreadsheet testing focused on either static or dynamic testing. As both testing approaches complement each other in terms of the types of faults detected [36], there should be more studies on combining both approaches for spreadsheet testing.
● Although there exist some techniques (e.g., WYSIWYT methodology [13,142−144] & metamorphic testing (MT) [20,139]) which help users generate test cases in a systematic manner, many spreadsheet developers still often work with a number of example inputs they use during development.	● Spreadsheets users are not willing to invest time in the specification of test cases [168].	● More work needs to be done to educate spreadsheet developers & to promote the use of more systematic test case generation techniques for spreadsheets.

Table 8 (Continued)

Observations	Possible reasons	Recommendations / future research directions
• Current spreadsheet environments do not support the "explicit" management of test cases [168]. Only few existing studies (e.g., [168]) have addressed this issue.		• More frameworks & supporting tools should be developed for test case management in the spreadsheet environments.
Usage & maintenance		
• Small number of relevant studies. • Among the already few studies, the majority of them: (a) come from the business-oriented IS publication venues & (b) just briefly touched on the techniques (e.g., documentation, user training, & change/ version management) relevant to this life-cycle stage.	• Possibly due to the nature of some relevant techniques (e.g., documentation & user training), technical-oriented research studies are less applicable.	• Even if technical-oriented research studies are less applicable, spreadsheet researchers (particularly those from the IS discipline) should consider performing case studies to identify what are the existing techniques being used in practice and how are they used in the real world. Such findings may bolster future studies relevant to this life-cycle stage.

spreadsheet QA issues, a holistic life-cycle approach is recommended. To the best of our knowledge, this paper is one of the first kind to integrate the major research studies on spreadsheet QA from the technical-oriented IT community, the business-oriented IS community, and the MS/OR community.

Almost all literature reviews have their limitations. Ours is no exception. Our literature search did not strictly follow a systematic approach. Even though our review is fairly extensive in terms of the survey time period, the number of search terms, target journals and conferences, as well as other supporting sources, and the collection of relevant papers will never be entirely complete in a broader sense. In other words, there might be some other relevant papers inadvertently omitted from our review because they do not satisfy our inclusion criteria (e.g., published in a journal outside our list). Despite the aforesaid limitations, this paper should serve as a handy reference for a comprehensive overview and big picture of the research work done in the important areas of spreadsheet QA.

Funding note Open Access funding enabled and organized by CAUL and its Member Institutions.

References

1. Grossman T A, Mehrotra V, Özlük Ö. Lessons from mission-critical spreadsheets. Communications of the Association for Information Systems, 2007, 20: 60
2. Ragsdale C T, Plane D R. On modeling time series data using spreadsheets. Omega, 2000, 28(2): 215–221
3. Aliane N. Spreadsheet-based control system analysis and design [Focus on Education]. IEEE Control Systems Magazine, 2008, 28(5): 108–113
4. Bianchi C, Botta F, Conte L, Vanoli P, Cerizza L. Biological effective dose evaluation in gynaecological brachytherapy: LDR and HDR treatments, dependence on radiobiological parameters, and treatment optimisation. Radiologia Medica, 2008, 113(7): 1068–1078
5. Zoethout R W M, Van Gerven J M A, Dumont G J H, Paltansing S, Van Burgel N D, Van Der Linden M, Dahan A, Cohen A F, Schoemaker R C. A comparative study of two methods for attaining constant alcohol levels. British Journal of Clinical Pharmacology, 2008, 66(5): 674–681
6. Dzik W S, Beckman N, Selleng K, Heddle N, Szczepiorkowski Z, Wendel S, Murphy M. Errors in patient specimen collection: application of statistical process control. Transfusion, 2008, 48(10): 2143–2151
7. AlTarawneh G, Thorne S. A pilot study exploring spreadsheet risk in scientific research. In: Proceedings of the EuSpRIG 2016 Conference "Spreadsheet Risk Management". 2016, 49–69
8. Thorne S. The misuse of spreadsheets in the nuclear fuel industry: the falsification of safety critical data using spreadsheets at British Nuclear Fuels Limited (BNFL). Journal of Organizational and End User Computing, 2013, 25(3): 20–31
9. Caulkins J P, Morrison E L, Weidemann T. Spreadsheet errors and decision making: evidence from field interviews. Journal of Organizational and End User Computing, 2007, 19(3): 1–23
10. Powell S G, Baker K R, Lawson B. Errors in operational spreadsheets. Journal of Organizational and End User Computing, 2009, 21(3): 24–36
11. McDaid K, Rust A. Test-driven development for spreadsheet risk management. IEEE Software, 2009, 26(5): 31–36
12. Panko R R. Two experiments in reducing overconfidence in spreadsheet development. Journal of Organizational and End User Computing, 2007, 19(1): 1–23
13. Burnett M, Cook C, Rothermel G. End-user software engineering. Communications of the ACM, 2004, 47(9): 53–58
14. Panko R R, Port D N. End user computing: the dark matter (and dark energy) of corporate IT. Journal of Organizational and End User Computing, 2013, 25(3): 1–19
15. Scaffidi C, Shaw M, Myers B. Estimating the numbers of end users and end user programmers. In: Proceedings of 2005 IEEE Symposium on Visual Languages and Human-Centric Computing. 2005, 207–214
16. Ko A J, Abraham R, Beckwith L, Blackwell A, Burnett M, Erwig M, Scaffidi C, Lawrance J, Lieberman H, Myers B, Rosson M B, Rothermel G, Shaw M, Wiedenbeck S. The state of the art in end-user software engineering. ACM Computing Surveys, 2011, 43(3): 21
17. McGill T J, Klobas J E. The role of spreadsheet knowledge in user-developed application success. Decision Support Systems, 2005, 39(3): 355–369
18. Erwig M. Software engineering for spreadsheets. IEEE Software, 2009, 26(5): 25–30
19. Schultheis R, Sumner M. The relationship of application risks to application controls: a study of microcomputer-based spreadsheet applications. Journal of Organizational and End User Computing, 1994, 6(2): 11–18

20. Poon P-L, Liu H, Chen T Y. Error trapping and metamorphic testing for spreadsheet failure detection. Journal of Organizational and End User Computing, 2017, 29(2): 25–42
21. Croll G J, Butler R J. Spreadsheets in clinical medicine. 2007, arXiv preprint arXiv: 0710.0871
22. Jannach D, Schmitz T, Hofer B, Wotawa F. Avoiding, finding and fixing spreadsheet errors – a survey of automated approaches for spreadsheet QA. Journal of Systems and Software, 2014, 94: 129–150
23. Thorne S. A review of spreadsheet error reduction techniques. Communications of the Association for Information Systems, 2009, 25: 34
24. Powell S G, Baker K R, Lawson B. A critical review of the literature on spreadsheet errors. Decision Support Systems, 2008, 46(1): 128–138
25. IEEE. ISO/IEC/IEEE 15026-1:2019 Systems and software engineering – systems and software assurance – Part 1: concepts and vocabulary. IEEE, 2019
26. IEEE. ISO/IEC/IEEE 24765:2017 Systems and software engineering – vocabulary. IEEE, 2017
27. IEEE. IEEE 1028−2008 IEEE standard for software reviews and audits. IEEE, 2018
28. Hofer B, Jannach D, Koch P, Schekotihin K, Wotawa F. Product metrics for spreadsheets — a systematic review. Journal of Systems and Software, 2021, 175: 110910
29. Power D J. A brief history of spreadsheets. See DSSResources website, 2004
30. Senders J W, Moray N P. Human Error: Cause, Prediction, and Reduction. Boca Raton, FL: CRC Press, 2020
31. Sheridan T B. Risk, human error, and system resilience: fundamental ideas. Human Factors, 2008, 50(3): 418–426
32. Panko R R, Sprague R H Jr. Hitting the wall: errors in developing and code inspecting a ‘simple’ spreadsheet model. Decision Support Systems, 1998, 22(4): 337–353
33. Elberzhager F, Münch J, Nha V T N. A systematic mapping study on the combination of static and dynamic quality assurance techniques. Information and Software Technology, 2012, 54(1): 1–15
34. Myers G J, Sandler C, Badgett T. The Art of Software Testing. 3rd ed. Hoboken, NJ: Wiley, 2011
35. Kikuchi N, Kikuno T. Improving the testing process by program static analysis. In: Proceedings of the 8th Asia-Pacific on Software Engineering Conference. 2001, 195–201
36. Panko R R. Spreadsheets and Sarbanes-Oxley: regulations, risks, and control frameworks. Communications of the Association for Information Systems, 2006, 17: 29
37. Panko R R. What we know about spreadsheet errors. Journal of End User Computing, 1998, 10(2): 15–21
38. Galletta D F, Hartzel K S, Johnson S E, Joseph J L, Rustagi S. Spreadsheet presentation and error detection: an experimental study. Journal of Management Information Systems, 1996, 13(3): 45–63
39. Cunha J, Fernandes J P, Mendes J, Saraiva J. MDSheet: a framework for model-driven spreadsheet engineering. In: Proceedings of the 34th International Conference on Software Engineering. 2012, 1395–1398
40. Grossman T A, Özlük Ö. A paradigm for spreadsheet engineering methodologies. 2008, arXiv preprint arXiv: 0802.3919
41. Leon L, Kalbers L, Coster N, Abraham D. A spreadsheet life cycle analysis and the impact of Sarbanes-Oxley. Decision Support Systems, 2012, 54(1): 452–460
42. Panko R R, Halverson R P Jr. Spreadsheets on trial: a survey of research on spreadsheet risks. In: Proceedings of the 29th Hawaii International Conference on System Sciences. 1996, 326–335
43. Lawson B R, Baker K R, Powell S G, Foster-Johnson L. A comparison of spreadsheet users with different levels of experience. Omega, 2009, 37(3): 579–590
44. Ronen B, Palley M A, Lucas H C Jr. Spreadsheet analysis and design. Communications of the ACM, 1989, 32(1): 84–93
45. Read N, Batson J. Spreadsheet Modelling Best Practice. UK: Pricewaterhouse Coopers, 1999
46. Brown P S, Gould J D. An experimental study of people creating spreadsheets. ACM Transactions on Office Information Systems, 1987, 5(3): 258–272
47. Kankuzi B, Sajaniemi J. A mental model perspective for tool development and paradigm shift in spreadsheets. International Journal of Human-Computer Studies, 2016, 86: 149–163
48. Kankuzi B, Sajaniemi J. An empirical study of spreadsheet authors’ mental models in explaining and debugging tasks. In: Proceedings of 2013 IEEE Symposium on Visual Languages and Human-Centric Computing. 2013, 15–18
49. Kankuzi B, Sajaniemi J. Visualizing the problem domain for spreadsheet users: a mental model perspective. In: Proceedings of 2014 IEEE Symposium on Visual Languages and Human-Centric Computing. 2014, 157–160
50. Kankuzi B, Sajaniemi J. A domain terms visualization tool for spreadsheets. In: Proceedings of 2014 IEEE Symposium on Visual Languages and Human-Centric Computing. 2014, 209–210
51. Carlsson S A. A longitudinal study of spreadsheet program use. Journal of Management Information Systems, 1988, 5(1): 82–100
52. Cragg P B, King M. Spreadsheet modelling abuse: an opportunity for OR? Journal of the Operational Research Society, 1993, 44(8): 743−752
53. Kruck S E, Maher J J, Barkhi R. Framework for cognitive skill acquisition and spreadsheet training. Journal of End User Computing, 2003, 15(1): 20–37
54. Abraham R, Erwig M, Kollmansberger S, Seifert E. Visual specifications of correct spreadsheets. In: Proceedings of 2005 IEEE Symposium on Visual Languages and Human-Centric Computing. 2005, 189–196
55. Erwig M, Abraham R, Cooperstein I, Kollmansberger S. Automatic generation and maintenance of correct spreadsheets. In: Proceedings of the 27th International Conference on Software Engineering. 2005, 136–145
56. Erwig M, Abraham R, Kollmansberger S, Cooperstein I. Gencel: a program generator for correct spreadsheets. Journal of Functional Programming, 2006, 16(3): 293–325
57. Engels G, Erwig M. ClassSheets: automatic generation of spreadsheet applications from object-oriented specifications. In: Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering. 2005, 124–133
58. Luckey M, Erwig M, Engels G. Systematic evolution of model-based spreadsheet applications. Journal of Visual Languages and Computing, 2012, 23(5): 267–286
59. Hermans F, Pinzger M, Van Deursen A. Automatically extracting class diagrams from spreadsheets. In: Proceedings of the 24th European Conference on Object-Oriented Programming. 2010, 52–75
60. Miller G, Hermans F. Gradual structuring in the spreadsheet paradigm. In: Proceedings of 2016 IEEE Symposium on Visual Languages and Human-Centric Computing. 2016, 240–241
61. Cunha J, Fernandes J P, Mendes J, Saraiva J. A bidirectional model-driven spreadsheet environment. In: Proceedings of the 34th International Conference on Software Engineering. 2012, 1443–1444
62. Mendes J. Coupled evolution of model-driven spreadsheets. In: Proceedings of the 34th International Conference on Software Engineering. 2012, 1616–1618
63. Cunha J, Fernandes J P, Martins P, Mendes J, Pereira R, Saraiva J. Evaluating refactorings for spreadsheet models. Journal of Systems

and Software, 2016, 118: 234–250

64. Cunha J, Fernandes J P, Martins P, Pereira R, Saraiva J. Refactoring meets model-driven spreadsheet evolution. In: Proceedings of the 9th International Conference on the Quality of Information and Communications Technology. 2014, 196–201
65. Cunha J, Fernandes J P, Mendes J, Saraiva J. Embedding, evolution, and validation of model-driven spreadsheets. IEEE Transactions on Software Engineering, 2015, 41(3): 241–263
66. Cunha J, Fernandes J P, Mendes J, Saraiva J. Extension and implementation of ClassSheet models. In: Proceedings of 2012 IEEE Symposium on Visual Languages and Human-Centric Computing. 2012, 19–22
67. Mendes J, Cunha J, Duarte F, Engels G, Saraiva J, Sauer S. Systematic spreadsheet construction processes. In: Proceedings of 2017 IEEE Symposium on Visual Languages and Human-Centric Computing. 2017, 123–127
68. Thorne S, Ball D, Lawson Z. Reducing error in spreadsheets: example driven modeling versus traditional programming. International Journal of Human-Computer Interaction, 2013, 29(1): 40–53
69. Miyashita H, Tai H, Amano S. Controlled modeling environment using flexibly-formatted spreadsheets. In: Proceedings of the 36th International Conference on Software Engineering. 2014, 978–988
70. Kruck S E. Testing spreadsheet accuracy theory. Information and Software Technology, 2006, 48(3): 204–213
71. Janvrin D, Morrison J. Using a structured design approach to reduce risks in end user spreadsheet development. Information and Management, 2000, 37(1): 1–12
72. Mather D. A framework for building spreadsheet based decision models. Journal of the Operational Research Society, 1999, 50(1): 70–74
73. Conway D G, Ragsdale C T. Modeling optimization problems in the unstructured world of spreadsheets. Omega, 1997, 25(3): 313–322
74. Sarkar A, Gordon A D, Jones S P, Toronto N. Calculation view: multiple-representation editing in spreadsheets. In: Proceedings of 2018 IEEE Symposium on Visual Languages and Human-Centric Computing. 2018, 85–93
75. Rust A, Bishop B, McDaid K. Test-driven development: can it work for spreadsheet engineering? In: Abrahamsson P, Marchesi M, Succi G, eds. Extreme Programming and Agile Processes in Software Engineering. Berlin: Springer, 2006
76. McDaid K, Rust A, Bishop B. Test-driven development: can it work for spreadsheets? In: Proceedings of the 4th International Workshop on End-User Software Engineering. 2008, 25–29
77. Isakowitz T, Schocken S, Lucas H C Jr. Toward a logical/physical theory of spreadsheet modeling. ACM Transactions on Information Systems, 1995, 13(1): 1–37
78. Dinmore M. Design and evaluation of a literate spreadsheet. In: Proceedings of 2012 IEEE Symposium on Visual Languages and Human-Centric Computing. 2012, 15–18
79. Benham H, Delaney M, Luzi A. Structured techniques for successful end user spreadsheets. Journal of End User Computing, 1993, 5(2): 18–25
80. Jansen B, Hermans F. XLBlocks: a block-based formula editor for spreadsheet formulas. In: Proceedings of 2019 IEEE Symposium on Visual Languages and Human-Centric Computing. 2019, 55–63
81. Hendry D G, Green T R G. CogMap: a visual description language for spreadsheets. Journal of Visual Languages and Computing, 1993, 4(1): 35–54
82. Macedo N, Pacheco H, Sousa N R, Cunha A. Bidirectional spreadsheet formulas. In: Proceedings of 2014 IEEE Symposium on Visual Languages and Human-Centric Computing. 2014, 161–168
83. Williams J, Negreanu C, Gordon A D, Sarkar A. Understanding and inferring units in spreadsheets. In: Proceedings of 2020 IEEE Symposium on Visual Languages and Human-Centric Computing. 2020, 1–9
84. Panko R R, Halverson R P Jr. An experiment in collaborative spreadsheet development. Journal of the Association for Information Systems, 2001, 2(1): 4
85. Ko A J, Myers B A. Development and evaluation of a model of programming errors. In: Proceedings of 2003 IEEE Symposium on Human Centric Computing Languages and Environments. 2003, 7–14
86. Hermans F, Aivaloglou E, Jansen B. Detecting problematic lookup functions in spreadsheets. In: Proceedings of 2015 IEEE Symposium on Visual Languages and Human-Centric Computing. 2015, 153–157
87. Klobas J, McGill T. Spreadsheet knowledge: measuring what user developers know. Journal of Information Systems Education, 2004, 15(4): 427–436
88. Lu M-T, Litecky C R, Lu D H. Application controls for spreadsheet development. Journal of Microcomputer Systems Management, 1991, 3(1): 12–22
89. Mccutchen M, Borghouts J, Gordon A D, Jones S P, Sarkar A. Elastic sheet-defined functions: generalising spreadsheet functions to variable-size input arrays. Journal of Functional Programming, 2020, 30: 26
90. Leon L A, Abraham D M, Kalbers L. Beyond regulatory compliance for spreadsheet controls: a tutorial to assist practitioners and a call for research. Communications of the Association for Information Systems, 2010, 27: 28
91. Roy S, Hermans F, Van Deursen A. Spreadsheet testing in practice. In: Proceedings of the 24th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). 2017, 338–348
92. Hermans F. Improving spreadsheet test practices. In: Proceedings of the 2013 Conference of the Center for Advanced Studies on Collaborative Research. 2013, 56–69
93. Harutyunyan A, Borradaile G, Chambers C, Scaffidi C. Planted-model evaluation of algorithms for identifying differences between spreadsheets. In: Proceedings of 2012 IEEE Symposium on Visual Languages and Human-Centric Computing. 2012, 7–14
94. Schmitz T, Jannach D. Finding errors in the Enron spreadsheet corpus. In: Proceedings of 2016 IEEE Symposium on Visual Languages and Human-Centric Computing. 2016, 157–161
95. Champion D, Wilson J M. The impact of contingency factors on validation of problem structuring methods. Journal of the Operational Research Society, 2010, 61(9): 1420–1431
96. Finlay P N, Wilson J M. A survey of contingency factors affecting the validation of end-user spreadsheet-based decision support systems. Journal of the Operational Research Society, 2000, 51(8): 949–958
97. Olphert C W, Wilson J M. Validation of decision-aiding spreadsheets: the influence of contingency factors. Journal of the Operational Research Society, 2004, 55(1): 12–22
98. Anastasakis L, Olphert C W, Wilson J M. Experiences in using a contingency factor-based validation methodology for spreadsheet DSS. Journal of the Operational Research Society, 2008, 59(6): 756–761
99. Panko R R. Applying code inspection to spreadsheet testing. Journal of Management Information Systems, 1999, 16(2): 159–176
100. Powell S G, Baker K R, Lawson B. An auditing protocol for spreadsheet models. Information and Management, 2008, 45(5): 312–320
101. Morrison M, Morrison J, Melrose J, Wilson E V. A visual code inspection approach to reduce spreadsheet linking errors. Journal of End User Computing, 2002, 14(3): 51–63
102. Ahmad Y, Antoniu T, Goldwater S, Krishnamurthi S. A type system for statically detecting spreadsheet errors. In: Proceedings of the 18th IEEE International Conference on Automated Software Engineering. 2003, 174–183

103. Antoniu T, Steckler P A, Krishnamurthi S, Neuwirth E, Felleisen M. Validating the unit correctness of spreadsheet programs. In: Proceedings of the 26th International Conference on Software Engineering. 2004, 439–448
104. Burnett M, Cook C, Pendse O, Rothermel G, Summet J, Wallace C. End-user software engineering with assertions in the spreadsheet paradigm. In: Proceedings of the 25th International Conference on Software Engineering. 2003, 93–103
105. Coblenz M J, Ko A J, Myers B A. Using objects of measurement to detect spreadsheet errors. In: Proceedings of 2005 IEEE Symposium on Visual Languages and Human-Centric Computing. 2005, 314–316
106. Erwig M, Burnett M M. Adding apples and oranges. In: Proceedings of the 4th International Symposium on Practical Aspects of Declarative Languages. 2002, 173–191
107. Singh R, Livshits B, Zorn B. MELFORD: using neural networks to find spreadsheet errors. Microsoft Research. See Microsoft website, 2017
108. Dou W, Cheung S-C, Gao C, Xu C, Xu L, Wei J. Detecting table clones and smells in spreadsheets. In: Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. 2016, 787–798
109. Dou W, Han S, Xu L, Zhang D, Wei J. Expandable group identification in spreadsheets. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. 2018, 498–508
110. Li D, Wang H, Xu C, Zhang R, Cheung S-C, Ma X. SGUARD: a feature-based clustering tool for effective spreadsheet defect detection. In: Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering. 2019, 1142–1145
111. Zhang Y, Dou W, Zhu J, Xu L, Zhou Z, Wei J, Ye D, Yang B. Learning to detect table clones in spreadsheets. In: Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis. 2020, 528–540
112. Zhang Y, Lv X, Dong H, Dou W, Han S, Zhang D, Wei J, Ye D. Semantic table structure identification in spreadsheets. In: Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis. 2021, 283–295
113. Abraham R, Erwig M. Header and unit inference for spreadsheets through spatial analyses. In: Proceedings of 2004 IEEE Symposium on Visual Languages and Human-Centric Computing. 2004, 165–172
114. Abraham R, Erwig M. UCheck: a spreadsheet type checker for end users. Journal of Visual Languages and Computing, 2007, 18(1): 71–95
115. Abraham R, Erwig M, Andrew S. A type system based on end-user vocabulary. In: Proceedings of 2007 IEEE Symposium on Visual Languages and Human-Centric Computing. 2007, 215–222
116. Chambers C, Erwig M. Automatic detection of dimension errors in spreadsheets. Journal of Visual Languages and Computing, 2009, 20(4): 269–283
117. Chambers C, Erwig M. Reasoning about spreadsheets with labels and dimensions. Journal of Visual Languages and Computing, 2010, 21(5): 249–262
118. Dou W, Cheung S-C, Wei J. Is spreadsheet ambiguity harmful? Detecting and repairing spreadsheet smells due to ambiguous computation. In: Proceedings of the 36th International Conference on Software Engineering. 2014, 848–858
119. Dou W, Xu C, Cheung S C, Wei J. CACheck: detecting and repairing cell arrays in spreadsheets. IEEE Transactions on Software Engineering, 2017, 43(3): 226–251
120. Xu L, Wang S, Dou W, Yang B, Gao C, Wei J, Huang T. Detecting faulty empty cells in spreadsheets. In: Proceedings of the 25th IEEE International Conference on Software Analysis, Evolution and Reengineering. 2018, 423–433
121. Cheung S-C, Chen W, Liu Y, Xu C. CUSTODES: automatic spreadsheet cell clustering and smell detection using strong and weak features. In: Proceedings of the 38th IEEE/ACM International Conference on Software Engineering. 2016, 464–475
122. Barowy D W, Berger E D, Zorn B. ExceLint: automatically finding spreadsheet formula errors. Proceedings of the ACM on Programming Languages, 2018, 2(OOPSLA): 148
123. Huang Y, Xu C, Jiang Y, Wang H, Li D. WARDER: towards effective spreadsheet defect detection by validity-based cell cluster refinements. Journal of Systems and Software, 2020, 167: 110615
124. Hermans F, Sedee B, Pinzger M, Van Deursen A. Data clone detection and visualization in spreadsheets. In: Proceedings of the 35th International Conference on Software Engineering. 2013, 292–301
125. Barowy D W, Gochev D, Berger E D. CheckCell: data debugging for spreadsheets. In: Proceedings of 2014 ACM International Conference on Object Oriented Programming Systems Languages and Applications. 2014, 507–523
126. Koch P, Schekotihin K, Jannach D, Hofer B, Wotawa F. Metric-based fault prediction for spreadsheets. IEEE Transactions on Software Engineering, 2021, 47(10): 2195–2207
127. Zhang R, Xu C, Cheung S C, Yu P, Ma X, Lu J. How effectively can spreadsheet anomalies be detected: an empirical study. Journal of Systems and Software, 2017, 126: 87–100
128. Hermans F, Pinzger M, Van Deursen A. Detecting and refactoring code smells in spreadsheet formulas. Empirical Software Engineering, 2015, 20(2): 549–575
129. Koch P, Hofer B, Wotawa F. On the refinement of spreadsheet smells by means of structure information. Journal of Systems and Software, 2019, 147: 64–85
130. Cunha J, Fernandes J P, Martins P, Mendes J, Saraiva J. SmellSheet detective: a tool for detecting bad smells in spreadsheets. In: Proceedings of 2012 IEEE Symposium on Visual Languages and Human-Centric Computing. 2012, 243–244
131. Hermans F, Pinzger M, Van Deursen A. Detecting and visualizing inter-worksheet smells in spreadsheets. In: Proceedings of the 34th International Conference on Software Engineering. 2012, 441–451
132. Hermans F, Dig D. BumbleBee: a refactoring environment for spreadsheet formulas. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. 2014, 747–750
133. Zhang J, Han S, Hao D, Zhang L, Zhang D. Automated refactoring of nested-IF formulae in spreadsheets. In: Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2018, 833–838
134. Chan H C, Ying C, Peh C B. Strategies and visualization tools for enhancing user auditing of spreadsheet models. Information and Software Technology, 2000, 42(15): 1037–1043
135. Koch P, Schekotihin K. Fritz: a tool for spreadsheet quality assurance. In: Proceedings of 2018 IEEE Symposium on Visual Languages and Human-Centric Computing. 2018, 285–286
136. Aurigemma S, Panko R. Evaluating the effectiveness of static analysis programs versus manual inspection in the detection of natural spreadsheet errors. Journal of Organizational and End User Computing, 2014, 26(1): 47–65
137. Panko R R, Aurigemma S. Revising the Panko-Halverson taxonomy of spreadsheet errors. Decision Support Systems, 2010, 49(2): 235–244
138. Sajaniemi J. Modeling spreadsheet audit: a rigorous approach to automatic visualization. Journal of Visual Languages and Computing, 2000, 11(1): 49–82
139. Poon P-L, Kuo F-C, Liu H, Chen T Y. How can non-technical end

users effectively test their spreadsheets? Information Technology and People, 2014, 27(4): 440−462

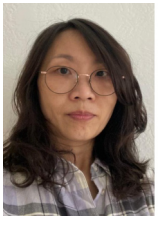
140. Chen T Y, Kuo F-C, Liu H, Poon P-L, Towey D, Tse, T H, Zhou Z Q. Metamorphic testing: a review of challenges and opportunities. ACM Computing Surveys, 2018, 51(1): 4
141. Ringstrom D. Trapping errors within Excel formulas. Accounting Web. See AccountingWEB website, 2012
142. Burnett M. End-user software engineering and why it matters. Journal of Organizational and End User Computing, 2010, 22(1): 1–22
143. Burnett M, Sheretov A, Ren B, Rothermel G. Testing homogeneous spreadsheet grids with the "What You See Is What You Test" methodology. IEEE Transactions on Software Engineering, 2002, 28(6): 576–594
144. Rothermel G, Burnett M, Li L, Dupuis C, Sheretov A. A methodology for testing spreadsheets. ACM Transactions on Software Engineering and Methodology, 2001, 10(1): 110–147
145. Su T, Wu K, Miao W, Pu G, He J, Chen Y, Su Z. A survey on data-flow testing. ACM Computing Surveys, 2017, 50(1): 5
146. Fisher M II, Rothermel G, Brown D, Cao M, Cook C, Burnett M.. Integrating automated test generation into the WYSIWYT spreadsheet testing methodology. ACM Transactions on Software Engineering and Methodology, 2006, 15(2): 150–194
147. Abraham R, Erwig M. AutoTest: a tool for automatic test case generation in spreadsheets. In: Proceedings of 2006 IEEE Symposium on Visual Languages and Human-Centric Computing. 2006, 43–50
148. Scaffidi C, Cypher A, Elbaum S, Koesnandar A, Lin J, Myers B, Shaw M. Using topes to validate and reformat data in end-user programming tools. In: Proceedings of the 4th International Workshop on End-User Software Engineering. 2008, 11–15
149. Kakarla S, Momotaz S, Namin A S. An evaluation of mutation and data-flow testing: a meta-analysis. In: Proceedings of the 4th IEEE International Conference on Software Testing, Verification and Validation Workshop. 2011, 366–375
150. Abraham R, Erwig M. Mutation operators for spreadsheets. IEEE Transactions on Software Engineering, 2009, 35(1): 94–108
151. Schmitz T, Jannach D, Hofer B, Koch P, Schekotihin K, Wotawa F. A decomposition-based approach to spreadsheet testing and debugging. In: Proceedings of 2017 IEEE Symposium on Visual Languages and Human-Centric Computing. 2017, 117–121
152. Galletta D F, Abraham D, El Louadi M, Lekse W, Pollalis Y A, Sampler J L. An empirical study of spreadsheet error-finding performance. Accounting, Management and Information Technologies, 1993, 3(2): 79–95
153. Saariluoma P, Sajaniemi J. Transforming verbal descriptions into mathematical formulas in spreadsheet calculation. International Journal of Human-Computer Studies, 1994, 41(6): 915–948
154. Jhugursing M, Dimmock V, Mulchandani H. Error and root cause analysis. BJA Education, 2017, 17(10): 323–333
155. Teo T S H, Tan M. Spreadsheet development and 'what-if' analysis: quantitative versus qualitative errors. Accounting, Management and Information Technologies, 1999, 9(3): 141–160
156. Leon L, Przasnyski Z H, Seal K C. Introducing a taxonomy for classifying qualitative spreadsheet errors. Journal of Organizational and End User Computing, 2015, 27(1): 33–56
157. Teo T S H, Lee-Partridge J E. Effects of error factors and prior incremental practice on spreadsheet error detection: an experimental study. Omega, 2001, 29(5): 445–456
158. Tukiainen M. Comparing two spreadsheet calculation paradigms: an empirical study with novice users. Interacting with Computers, 2001, 13(4): 427–446
159. Tukiainen M. Uncovering effects of programming paradigms: errors in two spreadsheet systems. In: Proceedings of the 12th Annual Workshop of the Psychology of Programming Interest Group. 2000, 247–266
160. Powell S G, Baker K R, Lawson B. Impact of errors in operational spreadsheets. Decision Support Systems, 2009, 47(2): 126–132
161. Dobell E, Herold S, Buckley J. Spreadsheet error types and their prevalence in a healthcare context. Journal of Organizational and End User Computing, 2018, 30(2): 20–42
162. Hendry D G, Green T R G. Creating, comprehending and explaining spreadsheets: a cognitive interpretation of what discretionary users think of the spreadsheet model. International Journal of Human-Computer Studies, 1994, 40(6): 1033–1065
163. Bishop B, McDaid K. Expert and novice end-user spreadsheet debugging: a comparative study of performance and behaviour. Journal of Organizational and End User Computing, 2011, 23(2): 57–80
164. Grigoreanu V, Burnett M, Wiedenbeck S, Cao J, Rector K, Kwan I. End-user debugging strategies: a sensemaking perspective. ACM Transactions on Computer-Human Interaction, 2012, 19(1): 5
165. Pirolli P, Card S. The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis. In: Proceedings of International Conference on Intelligence Analysis. 2005
166. Hofer B, Höfler A, Wotawa F. Combining models for improved fault localization in spreadsheets. IEEE Transactions on Reliability, 2017, 66(1): 38–53
167. Hofer B, Wotawa F. Why does my spreadsheet compute wrong values? In: Proceedings of the 25th IEEE International Symposium on Software Reliability Engineering. 2014, 112–121
168. Jannach D, Schmitz T. Model-based diagnosis of spreadsheet programs: a constraint-based debugging approach. Automated Software Engineering, 2016, 23(1): 105–144
169. Ruthruff J R, Burnett M, Rothermel G. Interactive fault localization techniques in a spreadsheet environment. IEEE Transactions on Software Engineering, 2006, 42(4): 213–239
170. Lawrance J, Abraham R, Burnett M, Erwig M. Sharing reasoning about faults in spreadsheets: an empirical study. In: Proceedings of 2006 IEEE Symposium on Visual Languages and Human-Centric Computing. 2006, 35–42
171. Hofer B, Perez A, Abreu R, Wotawa F. On the empirical evaluation of similarity coefficients for spreadsheets fault localization. Automated Software Engineering, 2015, 22(1): 47–74
172. Jannach D, Schmitz T, Hofer B, Schekotihin K, Koch P, Wotawa F. Fragment-based spreadsheet debugging. Automated Software Engineering, 2019, 26(1): 203–239
173. Abraham R, Erwig M. GoalDebug: a spreadsheet debugger for end users. In: Proceedings of the 29th International Conference on Software Engineering. 2007, 251–260
174. Schmitz T, Jannach D. An Al-based interactive tool for spreadsheet debugging. In: Proceedings of 2017 IEEE Symposium on Visual Languages and Human-Centric Computing. 2017, 333–334
175. Goswami S, Chan H C, Kim H W. The role of visualization tools in spreadsheet error correction from a cognitive fit perspective. Journal of the Association for Information Systems, 2008, 9(6): 321−343
176. Davis S J. Tools for spreadsheet auditing. International Journal of Human-Computer Studies, 1996, 45(2): 429–442
177. Mukhtar A, Hofer B, Jannach D, Wotawa F. Spreadsheet debugging: the perils of tool over-reliance. Journal of Systems and Software, 2022, 184: 111119
178. Cronan T P, Douglas D E. End-user training and computing effectiveness in public agencies: an empirical study. Journal of Management Information Systems, 1990, 6(4): 21–39
179. Dou W, Xu L, Cheung S-C, Gao C, Wei J, Huang T. VEnron: a

versioned spreadsheet corpus and related evolution analysis. In: Proceedings of the 38th IEEE/ACM International Conference on Software Engineering Companion (ICSE-C). 2016, 162–171
180. Cunha J, Erwig M, Mendes J, Saraiva J. Model inference for spreadsheets. Automated Software Engineering, 2016, 23(3): 361–392
181. Fischer G, Giaccardi E. Meta-design: a framework for the future of end-user development. In: Lieberman H, Paternò F, Wulf V, eds. End User Development: Human-Computer Interaction Series. Dordrecht: Springer, 2006, 427–457
182. Bhadauria V S, Mahapatra R, Nerur S P. Performance outcomes of test-driven development: an experimental investigation. Journal of the Association for Information Systems, 2020, 21(4): 1045–1071
183. Kroll P, Royce W. Key principles for business-driven development. IBM. See fulmanski.pl/zajecia/miasi/materials/kroll/index website, 2015
184. Kumar S, Bansal S. Comparative study of test driven development with traditional techniques. International Journal of Soft Computing and Engineering, 2013, 3(1): 352−360
185. Martin A. An integrated introduction to spreadsheet and programming skills for operational research students. Journal of the Operational Research Society, 2000, 51(12): 1399–1408
186. Kumar B. Create a Power BI report from Excel using Power PI Desktop. SPGuides.com. See SPGuides website, 2019
187. Gordon K J. Spreadsheet or database: which makes more sense? Journal of Computing in Higher Education, 1999, 10(2): 111−116
188. Lakshmanan L V S, Subramanian S N, Goyal N, Krishnamurthy R. On querying spreadsheets. In: Proceedings of the 14th International Conference on Data Engineering. 1998, 134–141
189. Li Y, Zhang C, Wang H, Wu F, Nie Y, Ren Y. A method of data granulation and indicators standardization of spreadsheet. In: Proceedings of the 6th IEEE International Conference on Cloud Computing and Big Data Analytics (ICCCBDA). 2021, 126–130
190. Tang J-F, Zhou B, He Z-J, Uros P. Toward spreadsheet-based data management in distributed enterprise environment. In: Proceedings of the 8th International Conference on Computer Supported Cooperative Work in Design. 2004, 578–581
191. Microsoft. Using Access or Excel to manage your data. Microsoft Support. See Microsoft website, 2022
192. Broman K W, Woo K H. Data organization in spreadsheets. The American Statistician, 2018, 72(1): 2–10

Pak-Lok Poon received the PhD degree in software engineering from The University of Melbourne, Australia. He is an associate professor with the School of Engineering and Technology, Central Queensland University, Australia. His research interests include software testing, requirements engineering and inspection, spreadsheet quality assurance, electronic commerce, and computers in education. He was a guest editor of two special issues of the Journal of Systems and Software (in 2018 and 2021). He was also an organizer for the 3rd, 4th, and 5th International Workshops on Metamorphic Testing in 2018, 2019, and 2020, respectively.

Man Fai Lau received his BSc(Hons) from The University of Hong Kong, China and PhD from The University of Melbourne, Australia. He joined Swinburne University of Technology, Australia in 2000. He received two Australian Research Council (ARC) Discovery Project (DP) research grants in 2005 (a 3-year DP project) and 2007 (a 5-year DP project). His research interests are on software engineering, software testing, data mining, data science, and applications of AI techniques in various areas such as oil drilling and financial analysis.

Yuen Tak Yu received the PhD degree from The University of Melbourne, Australia. He is an associate professor with the Department of Computer Science, City University of Hong Kong, China. His research interests include software testing, ecommerce, and computers in education. His publications have appeared in scholarly journals and leading international conferences, such as ACMTOSEM, IEEETSE, IEEETRel, IEEETSC, JSS, IST, Information Research, Computers and Education, ICSE, FSE, ISSRE, COMPSAC, QRS, ICCE, and others. Dr. Yu is a past chair of the IEEE Hong Kong Section Computer Society Chapter.



Sau-Fun Tang received the Master of Business (Information Technology) degree from the Royal Melbourne Institute of Technology, Australia and the PhD degree in software engineering from Swinburne University of Technology, Australia. She is currently working at the Royal Victorian Eye and Ear Hospital, Melbourne, Australia. She was an instructor in the Department of Finance and Decision Sciences at Hong Kong Baptist University, China and a lecturer in the School of Accounting and Finance at The Hong Kong Polytechnic University, China. Her research interests include software testing, spreadsheet quality assurance, management information systems, electronic commerce, and computers in education.