UDC 004.41:681.3.07

Benchmarking process in the spreadsheet editor testing

I. Kravchenko, I. Kukhar, O. Kuznetsov, I. Svatovsky
V. N. Karazin Kharkiv National University, Svobody sq.4, Kharkiv, 61022, Ukraine

This article is dealing with the main peculiarities of different spreadsheet editors and testing challenges related to them. Differences in the features of the table editors are reviewed. Shortcomings of manual testing of table editors are presented. The task was to improve testing effectiveness within the certain limitations, such as time and a budget. “Benchmarking” business process is adapted and introduced as a “benchmarking” testing approach which is applied for testing the table editors’ functionality. At the end of the work, the results and conclusions are presented after such an adaptation.

***Key words**: Benchmarking, Semi-automatic testing, Spreadsheet editors, Table editors, Manual testing, Testing strategy.*

Стаття розглядає основні особливості різноманітних табличних редакторів та пов’язані з цим задачі тестування. Наведені відмінності в функціональних можливостях табличних редакторів та недоліки мануального тестування табличних редакторів. Було поставлено завдання організації тестування для підвищення його ефективності з певними обмеженнями, такими як час і бюджет. “Бечмаркінг” як бізнес-процес адаптований та представлений у якості “бенчмаркінг”-підходу до тестування, який застосований до тестування функціональних можливостей табличного редактору. На завершення роботи приведено результати та висновки після проведення такої адаптації.

***Ключові слова:** бенчмаркінг, напівавтоматичне тестування, редактори електронних таблиць, табличні редактори, мануальне тестування, стратегії тестування.*

В статье рассматриваются основные особенности различных табличных редакторов и связанные с этим задачи тестирования. Приведены отличия в функциональных возможностях редакторов электронных таблиц и недостатки мануального тестирования табличных редакторов. Была поставлена задача организации тестирования для повышения его эффективности с определенными ограничениями, такими как время и бюджет. «Бенчмаркинг» как бизнес-процесс адаптирован и представлен в качестве «бенчмаркинг»-подхода к тестированию, который применен к тестированию функциональных возможностей табличных редакторов. В завершение работы приведены результаты и выводы после проведения такой адаптации.

***Ключевые слова:** бенчмаркинг, полуавтоматическое тестирование, редакторы электронных таблиц, табличные редакторы, мануальное тестирование, стратегии тестирования.*

1 Introduction

Testing of the table editors as a part of software quality assurance is quite a complex and comprehensive task for test engineers [1]. The complexity exists due to a number of reasons and one among them is the existence of different standards for developing spreadsheet editors. For instance, ECMA OOXML standard stands for a coding standard for a well-known and widely used MS Excel table editor. At the same time the family of editors LibreOffice/OpenOfficeCalc uses the different standard - OpenFormula. OpenFormula standard was designed taking into account the fact that it can be used for office applications as well as outside of them. Therefore, different

standards for developing table editors assume that the tables based on each standard will have different features.

In this article we examine the differences between various table editors and take them into account when testing the spreadsheet editor named “Collabio” developed by “XCDS” company.

“Collabio” editor is developed as a multipurpose instrument to handle all common spreadsheet file formats. Thus it is of paramount importance to know the areas that differ before creating the test plan and testing.

Furthermore, we propose the new testing strategy using the “benchmarking” conception in our test process [2]. Benchmarking as a business model has been adapted to our testing strategy and has already brought a number of positive results due to working process has become easier, more predictable and controlled. The high level benchmarking is considered in this article.

2 Main differences of spreadsheet editors and complexity of testing table editors

Table editor testing is a little known area. There is lack of information how the testing of table editors is performed. As a rule, companies hide this information and it is almost impossible to find something in open resources. That is why it is a challenge to design a testing strategy for the table editors without know-how and previous experience in the particular area.

Let us start with a deep investigation of table editors’ differences. Consider some differences in the standards of spreadsheets. There are two main standards for building table editors. Regarding the table editor functions we can note plenty of differences between OpenFormula and ECMA OOXML standards.

The functions can have different number of parameters (arguments). The function arguments are divided to the required parameters and optional ones and their number can differ from one table editor to another. For instance, INDEX function has the following variations:

There are 2 forms of INDEX function in MS Excel: array and reference. The syntax specified for each form is given below:

Array form: **INDEX(array, row_num, [column_num])**

Reference form: **INDEX(reference, row_num, [column_num], [area_num])**

But LibreOfficeCalc table editor allows for the single form of INDEX function:

LibreOfficeCalc syntax: **INDEX(Reference; Row; Column; Range)**

Square brackets indicate that the argument is optional. Thus, for MS Excel editor we have 2 options for the formula syntax where the third and the fourth arguments are optional ones, while LibreOfficeCalc editor supports four arguments for the formula and all of them are required. Obviously it is impossible to support all standards, so particular implementation should be chosen for the “Collabio” editor by business analytics.

Moreover, each argument of the function can be represented by different data types and calculation accuracy can be different. The last but not the least is the difference in the formula syntax. A few examples of syntax differences for records given for the English locale are presented in the table below (syntax for the different locales can be different as well).

Table1 - Syntax difference for MS Excel and LibreOfficeCalc editors:

	MS Excel	LibreOfficeCalc
Array	={1,2,3;3,2,1}	={1,2,3|3,2,1}
Reference	=Sheet1!A1	=Sheet1.A1
Range	=Sheet1!A1:B5	=Sheet1.A1:Sheet1.B5
Function	=SUM(1,2,3)	=SUM(1;2;3)
Range intersection operator	=A1:B5 B5:C6	=A1:B6!B5:C6

Let us take a closer look at those differences. In the last row, in the example for MS Excel, the formula contains a space symbol which is meaningful because it is the binary operator of a range intersection. So, in addition to the differences in the record syntax, there is a problem of tokens used within the given record. The space operator in OOXML is not just a blank space, sometimes it is the operator of a range or cells intersection. This should also be taken into account when designing the test plan. A space in the syntax could lead to the ambiguous situations.

Consider examples where records look almost the same but have an absolutely different meaning.

=SUM(A1,A2,A3) (1)

=SUM (A1,A2,A3) (2)

In the first case, it is a call to SUM function with the list which contains three parameters. In the second case we can see the binary operator of an intersection (space) between 2 parameters – the defined name (named range of cells) and the list which contains three parameters. So both records will be valid in OOXML format editors, but the calculation result will be totally different. In the second case as a calculation result we will receive a cumulative result of the intersection between the named range and the list of parameters.

It is worth mentioning that a record like "=SUM A1" makes sense if SUM is defined range of cells, for instance A1:B2. In this case the calculation result will be the A1 cell value, because it is an intersection value.

Although the syntax of the formulas has plenty of differences, it doesn't prevent developers from creating a single internal representation for both language standards. An internal representation is a syntax parse tree, where differences are eliminated. Certainly, not all records can be converted due to the particular level of incompatibility. Therefore, if the crucial differences are present, there are no other choices but support one record. The MS Excel standard is to be chosen in most cases, as the experience shows.

"Collabio' editor can open both formats of document, but the main format is OOXML. All opened files are converted into the single representation, which doesn't depend on the input format. But it is possible to save this representation into both formats. Developers tried to minimize the differences in order make customers' work easier and more comprehensive.

There is another testing problem, namely, a huge volume of the testing required. A common user usually employs formulas which are associated with the calling function

suggested by the particular spreadsheet editor. All functions [3] are divided into the following groups:

1. Information functions;
2. Financial functions;
3. Logical functions;
4. Lookup and references functions;
5. Math and trigonometry functions;
6. Text functions;
7. Statistical functions;
8. Database functions;
9. Date and time functions;
10. Engineering functions;
11. WEB functions;
12. Cube functions;
13. Compatibility functions.

There are more than 450 functions within the specified groups, taking into account all versions of the table editors. It gives some notion of the volume of testing. Our first time/resource calculation has shown that execution of the test suites, covering the "Functions" table editor functionality will take 3 years by 2 quality assurance engineers. It is obvious that we can not afford such amount of time in a modern community of fast product developing. Therefore, the manual testing is ineffective and does not any suit time and budget restrictions.

Besides, most of the functions use non-trivial calculations, which are hard to verify manually, because an average quality assurance engineer lacks deep knowledge in such domains as mathematics, finance, etc. (and we have the groups of the functions that require such knowledge). Even if a specialist with the required knowledge is hired, the problem of timing is still present. The manual testing takes a lot of time and resources, which is insufficient for business needs. Undoubtedly, the testing expenses should be adequate to the company's resources. So, the decision to use semi-automated testing by developing the macros for creating test cases and writing scripts which will trigger the testing on each new successful product build have been made.

Therefore, the complexity of the table editor testing results not only from existence of different product development standards but from the huge volume of testing as well. Such volume is caused by immense and multiple functionality of various table editors. Another key issue is the complexity of the tested functional. It has been a challenge to design an effective testing system within certain time and budget constraints.

3 The adaptation of the benchmarking business process and its introduction in the spreadsheet editor testing

In spite of the peculiarities and complexities of table editors, it is clear that the product testing is necessary, and testing must answer the questions about the product quality [2, 4].

Main quality criteria are directly related to the risk level including technical risk, product risk as well as project limitations such as time and budget. Testing should provide the satisfying information to stakeholders so that they can make informed

decisions about releases of the system (product) for the next development phase or for transferring the product to customers [5].

It is obvious that the manual testing will not be sufficient enough, the automation testing will require a lot of time and people resources, though will bring some benefits in the end. But what we needed was the immediate testing, because the functionality was growing fast and we needed to establish certain quality control [1]. The unit testing is not sufficient for covering all basic table editor functionality. Besides, the unit testing is a developer's duty, so it was being executed already. Thus we decided to use the semi-automated testing based on the "benchmarking" conception.

Benchmarking is mostly known for its application in the management [2]. It is the process of comparing one's business processes and performance metrics to the industry best practices. In the process of the benchmarking, management identifies the best firms in their industry, or in another industry where similar processes exist, and compares the results and processes of those studied (the "targets") to one's own results and processes. Twelve stages of benchmarking process were determined by Robert Camp (who wrote one of the earliest books on benchmarking in 1989) [6]. We borrowed and updated some of them, so our process contains the following stages:

Stage1 – Identify test data source – the test oracle;

Stage2 – Collect data and create the test suites;

Stage3 – Establish differences;

Stage4 – Analyze differences and adjust the test cases;

Stage5 – Review and recalibrate the test cases.

The test oracle was provided by the business analytics team. In our case it was MS Excel as it is the most popular, the most functional and previously released product, so mostly we relied on it. The next stage was designing the test suites. Based on the aforementioned information, we decided to make a benchmark files in the xlsx format (default test file format - MS Excel, version 2010, xlsx). One single file or a set of files with the test suit for the particular function or functionality was named "Benchmark".

Our test coverage plan included the file formats specified in Table 2.

Table2 – Test coverage: formats and versions of the test files

MS Excel, version 2010, xlsx format	LibreOfficeCalc, latest version (up to date), ods format	Google spreadsheet, xlsx format
MS Excel, version 2013, xlsx format		
MS Excel, version 2016, xlsx format		
MS Excel, version Online, xlsx format		

When the benchmark file was ready we loaded this file through the "FormulaCalculator" – tool created by developers, which helps to check the core functionality of the editor calculations. Then we designed the script on the Python language which automated the process of the file loading and calculating results

through the FormulaCalculator tool on our build system. So each new build the Python script triggered the file loading through the FormulaCalculator tool and as the output we received a report with the testing results in the HTML-format. Therefore, we controlled each code change which could affect the calculations for the formulas and the functions so we could report immediately to developers, to product owners or any other interested party as well. Moreover, creating "Benchmarks" files was also automated after the structure became stable. For this purpose, we used built-in Macros functionality of the VBA language.

Our approach to the building of the test suites is based on the benchmarking method called "Comparison tables", where we compare calculation results of our own application with our oracle – MS Excel spreadsheet editor. Let us consider how the test file looks like in details. On Fig. 1 you can see a typical Benchmark test file.

15	**STANDARD DATA INPUT**									
16	**DATA TYPE: EMPTY,DEFAULT_ARG2**									
17	**NUMBER1 - EMPTY,NUMBER2 - ARG2**									
18	**Test cases**	**Formula/Actual result**	**Syntax**	**ER (App)**	**ER (Excel)**	**STATUS**	**Comment**	**Different ER?**	**New Issue**	**Old Issue**
19	HAND input - CELL	19	BITOR(,19)	19	19	Pass		NO		
20	REFERENCE input - CELL	19	BITOR(M5,19)	19	19	Pass		NO		
21	ARRAY input - CELL	Impossible to input	BITOR({},19)	19	Impossible to input		[SO]Tested manually	YES		
22	REFERENCE input - RANGE	#VALUE!	BITOR(M5:M8,19)	#VALUE!	#VALUE!	Pass		NO		
23	ARRAY input - RANGE	Impossible to input	BITOR({,,,},19)	19	Impossible to input		[SO]Tested manually	YES		
24										
25	**DATA TYPE: BOOLEAN,DEFAULT_ARG2**									
26	**NUMBER1 - TRUE,NUMBER2 - ARG2**									
27	**Test cases**	**Formula/Actual result**	**Syntax**	**ER (App)**	**ER (Excel)**	**STATUS**	**Comment**	**Different ER?**	**New Issue**	**Old Issue**

Figure 1 – Example: standard test cases part of Benchmark file

The meaningful columns within the Fig. 1 table are the "Formula/Actual result" column – where we can see the actual calculation result after loading this test file to the proper table editor ("Collabio" editor in our case), and the "ER(App)" column with the expected result based on our test oracle – MS Excel spreadsheet editor. The "ER (Excel)" column contains the expected result of the MS Excel. Basically, we have two expected results – MS Excel and our own expected result, which is based on the MS Excel but does not necessarily match the MS Excel. So if the warning about the difference between the tested calculation and the MS Excel one is shown "Status" column, we try to determine if it is an error in the tested calculation or a MS Excel issue. In case if this difference results from the MS Excel issue, we change the value in the "ER(App)" column to the correct one and set the comment within the "Comments" column to specify why we have changed the expected result.

The "STATUS" column contains equality verification for the "ER (app)" and the "Formula/Actual result" column performed by the following formula:

=IF((B19)=(D19),"Pass","Fail")

or

=IF(ERROR.TYPE(B22)=ERROR.TYPE(D22),"Pass","Fail")

- when an error is expected as a result of calculation and would like to check if the resulted error type is correct.

The "New Issue" and the "Old issue" columns are added for the defect tracking purposes. Defects are noted during the testing and added to the proper column if the current defect affects relevant test case. Defect tracking is also automated through the integration with the JIRA bug tracking system (which is used in our project) so that if

the defect is closed or fixed we will have information that the defect is no longer affects the current test case and must be removed from the testing files (moved from "New issue" to "Old issue" column).

The standard part of the benchmark file is now created by running macros in the MS Excel file. So the process of creating a test file is easy and fast. Most of the efforts are applied to analyzing the testing results, not to developing the test cases.

However, it is a fact that in testing you should avoid repeating scenarios due to a well-known "Pesticide Paradox" testing principle. To overcome this "Pesticide Paradox", it is really very important to review the test cases regularly. New tests should be developed to check different parts of the software or system which will help to find more defects. Therefore we created so-called "Mixed part" of Benchmark file where each test engineer is welcome to create different testing scenarios relying on his/her own experience in testing. The structure of the "Mixed" part is basically the same, but those test cases are created manually and have unique scenarios for every functionality.

So, let us summarize what benefits "Benchmarking" has brought to our testing process. First of all, we have not checked each calculation manually due to the test oracle. Secondly, it has become clear how different our product is from the market leader (MS Excel editor) and what their main differences are. It gives us the understanding of the advantages and disadvantages of our product, as well as pros and cons of MS Excel editor. Moreover, now we have a set of the benchmarks files for every function functionality and we are ready to answer the question "Are we working better now?" for each new build, so the benchmarking helps to save resources

The creation of a "Benchmark" report is automated process as well. We have built the HTML-report which gathers all the problems and presents results of the test cases output analysis as well as the defects from the JIRA tracking tool and, therefore, represents the current status of the product quality. This report is available for quality assurance engineers as well as for developers and product owners. Developers are able to make a build in order to check how a changed code affects the tests cases. Product owners use this report to gather information and share it with stakeholders.

Below you can find an example of the "Benchmark" report, which consolidates results from all benchmark files (Fig. 2):

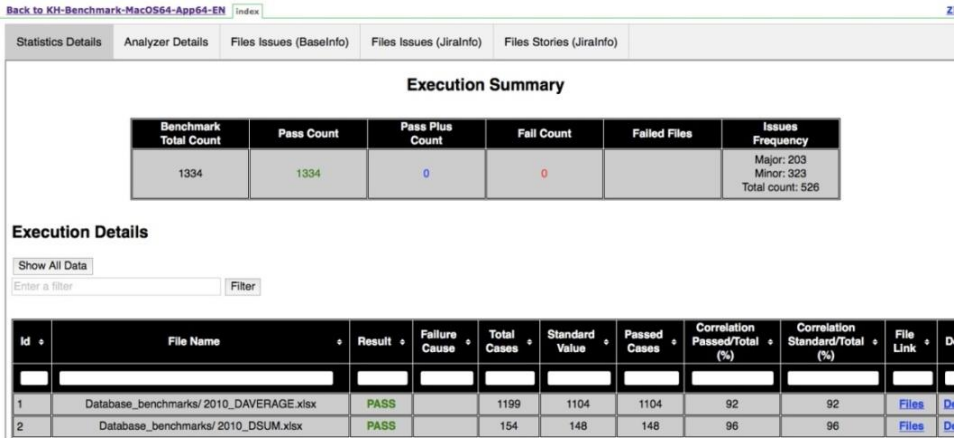

Figure 2. – Example: "Benchmark" report

The report contains several tabs, each with different types of analytics. The first tab represents the general status of each file, the total count of the test cases and a number of passed and failed test cases. The report is generated automatically for each new build of the product. It takes no time for quality assurance engineers to generate a report, so there is no need in additional manual reports.

While analyzing how effective the benchmark testing could be, we have taken into account the time required for covering the basic functionality. On the chart (Fig. 3) we compared time expenses required for covering basic functions (450 functions) for a team of five members.

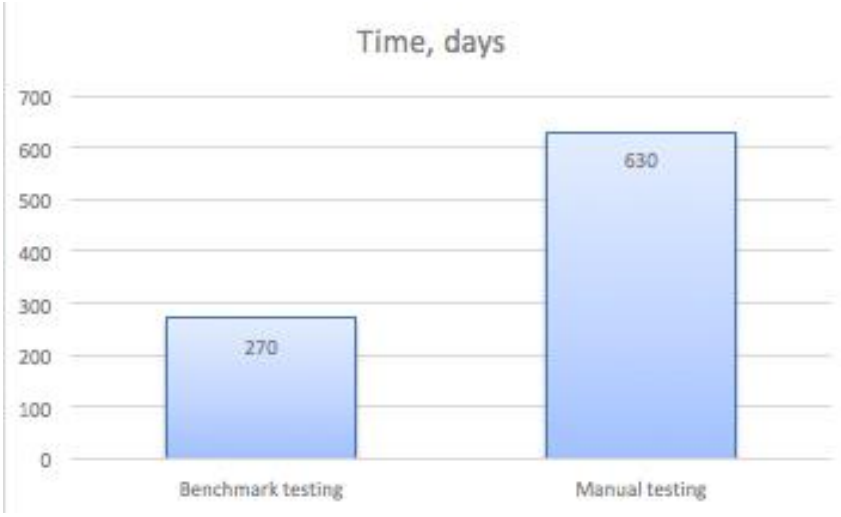

Figure 3. Time chart: benchmark testing via manual testing

The results are approximate but we can see that the estimations are vastly different. The team is able to finish covering the basic functionality within the 270 days using the benchmark approach (less than a year), while the manual testing requires at least 630 days, which is almost two years. The chart is based on the estimation that covering a single function by the benchmark testing (including review process) takes three days, while the manual testing requires at least seven days per function.

4 Conclusions

The table editor testing is a specific testing area and has a range of peculiarities which you should take into account when organizing the testing strategy and process. The main differences between table editors result from the different standards of developing table editors. The accuracy of calculations, syntax differences and different language locales create the additional differences. Another significant problem is volume of testing (because we should check more then 450 functions and great amount of general functionalities). The restrictions, such as time and budget, should also be taken into account.

Benchmarking as a business process has helped us to build the effective testing strategy. It has given us the test oracle. The automation of benchmarking testing

processes has allowed us to achieve the goal – the creation of the fast-answering system that reports about the product quality in any point of time. The benefits of benchmarking is the quickness of creating test cases, which is far faster than creating test cases with verifying testing results manually.

Eventually, the new testing approach is proposed as "benchmark testing", which considers the testing of new functionality based on the existent test oracle. The test oracle is derived from the best or desired product in the industry. But it should not be used as 100% base for final testing results, because it needs the correction after the deep analysis. The corrections should be made in favor of more accurate and logical results. Afterwards the corrected results serve as the expected results for the tested product.

There are the following benefits of the benchmarking approach introduction: the significantly reduced testing time; the automation tools can be used for the benchmarking testing (macros, scripts, integration with bug tracking tools and build systems); the testing complexity is decreased due to the test oracle existence; testing reports are more advanced and easier to work with. Those benefits make the benchmarking approach preferable for testing table editor functionality.

REFERENCES

1. Eric J. Braude, Michael E. Bernstein Software Engineering: Modern Approaches, Second Edition. Wiley. - 2010. – 782 p.
2. David J. Lilja Measuring Computer Performance: A Practitioner's Guide. Cambridge University Press. - 2004. – 261 p.
3. Excel functions (by category), Microsoft support site, [electronic resource] access mode: https://support.office.com/en-us/article/Excel-functions-by-category-5f91f4e9-7b42-46d2-9bd1-63f26a86c0eb?ui=en-US&rs=en-US&ad=US
4. The International Software Testing Qualifications Board® (ISTQB®) [electronic resource], access mode: https://www.istqb.org/downloads/send/2-foundation-level-documents/3-foundation-level-syllabus-2011.html4
5. Software Testing. An ISTQB-BCS Certified Tester Foundation guide Third edition by Brian Hambling. Peter Morgan, Angelina Samaroo, Geoff Thompson and Peter Williams. BCS Learning & Development Ltd. - 2015. – 263 p.
6. Robert C. Camp Business Process Benchmarking**:** Finding and Implementing Best Practices. Milwaukee, Winconsin, ASQC Quality Press. - 1995. - 464 p.

Надійшла у першій редакції 19.12.2017, в останній - 10.01.2018.