Advances in Economics, Business and Management Research, volume 138
2nd International Scientific and Practical Conference "Modern Management Trends and the Digital Economy: from Regional Development to Global Economic Growth" (MTDE 2020)

Testing System for Corporation Productivity Improvement Department

Lavrenova E.V.[1], Kalinina A.M.[2], Bocharov M.I.[1,*]

[1]*Institute of Digital Education Moscow City University (MCU) Moscow, Russia*
[2]*Testing department "MVM OOO" (limited liability company) Moscow, Russia*
[*]*Corresponding author. Email: mi1@mail.ru*

ABSTRACT
The article is devoted to the development and implementation of a model for improving the efficiency of the testing department of the MBM LLC Company. The organization and process of software testing play an important role in the digital economy since it is the business continuity that determines the reliability of the data processed by corporate systems and, ultimately, the quality of the sale of goods and services.
The main areas in which the process of increasing the efficiency of the testing department is carried out are the introduction of automated tests for new processes being implemented in the organization. To automate the testing processes, we used the BDD (Behavior-driven development, literally "development through behavior") methodology - this is a software development methodology through continuous communication between developers, testers and analysts.
During the study, the basic principles of organizing an automated testing system in the IT departments of companies were examined, the technical tools that are used in the department were examined, and a testing automation model was proposed. A comparative analysis of the possible means of implementing the proposed model is carried out. As a result of the analysis, the most optimal software tools for a particular project were selected. The process of creating automated tests is described.
After the introduction of automated tests, an assessment on the effectiveness of the constructed system was made to increase the efficiency of the testing department and the use of developed autotests in real practice. The evaluation results showed that for this project, testing automation according to the proposed model became an appropriate solution and brought significant results already at the pilot stage: by freeing up manual testers from the manual execution of regularly repeated tests.
The proposed model allows increasing the efficiency of the testing process and reduce the labor and material costs of the Company for its organization.
Keywords: *software testing, software testing methods, software testing automation, BDD development methodology, agile software development*

1. INTRODUCTION

In the digital age, there is a global trend towards the automation of manual labor in all areas of human life. The automation trend has not ignored the area of software development, which is important in our time [1]. By itself, the use of any software is aimed at the automatic solution for certain problems. In the process of software development, automated solutions can also be used to accelerate subprocesses within the company [2].
What does the software development process consist of? There are different models of software development, but they all consist of the following stages: development of software requirements, design, software implementation, testing and commissioning. Moreover, testing is becoming increasingly important in the life cycle of software development [3]. In this study, a system is being developed to increase the efficiency of the testing department of the MBM LLC Company.
For the testing department of MBM LLC, the problem of the speed of tests by specialists in manual mode is especially relevant [4]: with the release of each new version of the tested software, the number of test scenarios increases significantly, which leads to an excessive load on testing specialists, and, as a result, decrease in the efficiency of their work, increase in the overhead costs of the Company and exceeding the time allotted for testing [5].
The purpose of this study is to increase the efficiency of the department through the automation of technological processes.
The hypothesis of the study was the assumption that the efficiency of the testing department would increase if a

specially selected automated test system for new processes was introduced.
The object of study is the testing department of MBM LLC.
The subject of the study is software testing automation.
The novelty of this study is a unique model for increasing the efficiency of the testing department by automating the testing process.
The practical significance of this study lies in the implementation of the proposed model in practice and is expressed in optimizing the testing process and reducing labour and material costs for organizing the testing process at MBM LLC.
As a result of the study, the following was received:

1. Analysis, the role of the testing department in the business processes of the MBM LLC.

2. Model of a system for increasing the efficiency of the MBM LLC testing department.

3. Calculation of this model effectiveness.

1.2. Information systems and technologies used in the testing department

In 2011, MBM LLC began introducing an integrated automation system NTSWincash, which covers all processes of the retail network. With the NTSWincash system, more than 20 thousand employees of the service department, sales department, warehouse logistics, cash registers, delivery department in stores are working. This system automates all the processes inside the store and the processes of interaction with the client.
The testing department is responsible for the quality of the NTSwincash system. Its main tasks are:

1. Comprehensive quality control system NTSwincash.

2. Preparation of test documentation (test plans, test scripts, etc.).

3. Conducting regression testing (regression testing is a type of testing aimed at checking changes made in the application to confirm the fact that the previously existing functionality works as before) [6] and smoke testing (smoke testing is a short test cycle performed to confirm that after building the code, the installed application starts and performs the main functions) [7].

4. Detection and localization of errors in the operation of the NTSwincash system.

5. Fixing and tracking errors in the operation of the NTSwincash system.

6. Verification of compliance of software product documentation with standards and really implemented functions.

7. Participation in the development and implementation of a quality system.

To improve the processes within the testing department, various auxiliary software solutions are used that simplify the communication processes within the whole unit: Atlassian JIRA, Atlassian Confluence, TestRail.

2. METHODS OF RESEARCH

2.1. Modern software testing methods

Almost all modern companies that are professionally involved in software development use the software life cycle model in their work. The life cycle of software development in different companies may vary, however, there are standards such as GOST R ISO/IEC 12207-2010 that define accepted practices for software development and maintenance. According to this standard, after the stage of a software implementation of the system, the stage of qualified testing of the developed system should follow.
Software testing - checking the correspondence between the real and expected behaviour of the program, carried out on a finite set of tests selected in a certain way. In a broader sense, testing is one of the quality control techniques [8], which includes activities for planning work (Test Management), designing tests (Test Design), performing testing (Test Execution) and analyzing the results (Test Analysis) [9]. Various sets of test cases and testing strategies are aimed at eliminating errors in the code and ensuring accurate and optimal software performance.

2.2. The choice of means for implementing the developed model of the system for improving the testing department efficiency

There are many approaches to automation and many tools for implementing automated tests [10], and the team's task is to select those that are most suitable for a particular project in terms of their technical properties, price and capabilities. As part of a test automation project, it was decided to use the BDD methodology [11].
BDD (Behaviour-driven development) is a software development methodology through continuous communication between developers, testers and analysts. 'Test automation by BDD methodology means that an automated test has a multi-level structure and is written at the highest level in the form of a script describing some behaviour in a human-readable language. This makes automated tests accessible to understanding by all interested parties and allows attracting employees who do not have programming skills (testers, analysts) to the automation process. The BDD methodology allows reducing the time spent on manufactured products testing, improving the quality of testing [12] and making the

process transparent and understandable for all project participants.
The framework on which BDD tests will subsequently be written is the basis of the whole methodology, its main element. That is why the choice of tools for the formation of the framework should be given increased attention.
We'll evaluate some of the most popular tools for testing automation using the selected criteria. The evaluation results are shown in the Table 1.

Table 1 Comparison of Test Automation Tools

	Test complete	**Winium (Selenium)**	**AssertJ + Gherkin**	**FEST**	**Jemmy**	**Squish**
Testing Java Swing UI	+	-	+	+	+	+
Cheapness	-	-	+	+	+	+
Requirements for Specialists	JS, Python, VBS	Java, PHP, Python, C #	Java	Java	Java (Open JDK)	JS, Python, Ruby, Perl
Having a development community	4	3	4	1	2	3
Easy to create and maintain	4	5	5	5	5	5
Code transparency	4	5	5	5	5	4
Speed of tests and getting results	3	5	5	5	5	4
Scalability	2	4	5	5	4	3
Integration in CI / CD	3	5	5	5	5	2

As a result of comparing the above tools, the AssertJ + Gherkin tool was chosen as the most optimal according to the criteria of interest to us. Thus, to create a framework for automating tests using the BDD methodology using these tools, we will need to use the following software implementation tools:

1. JAVA programming language.
2. AssertJ library.
3. Automation tool Cucumber JVM + human-readable language Gherkin.
4. Integrated Development Environment IntelliJ IDEA.
5. Atlassian Bitbucket Git Solution.
6. A tool for creating Yandex Allure reports.

Using the technical equipment already existing in the department will help greatly simplify the process of deploying a test automation project. Thus, the JIRA project management system will help to quickly establish an effective workflow and teamwork, Confluence will provide convenient tools for the joint description of all project documentation for test automation, and using TestRail it will be possible to use and process ready-made test scripts for the needs of the project.

3. RESEARCH RESULTS

3.1. System for improving the efficiency of the software testing department

The support and development of the NTSwincash trading system in the company Mvideo Management LLC are handled by the Information Technology Directorate. Each new version of the system goes through several stages of checks before it is "flooded" with a productive environment (that is, the electronic equipment of real stores).
The process of developing and testing each new version of the NTSwincash system is structured as follows: the planned work front is conditionally divided between five Agile teams, each of which is engaged in the development and testing of improvements within its area of responsibility [13]. For example, the Agile 1 team is engaged in improvements within the Cashier module, the Agile 2 team is engaged in refinements of the Logistics module, etc.
Agile software development (flexible development methodology) is a series of approaches to software development that focus on using iterative development, dynamically forming requirements and ensuring their implementation as a result of constant interaction within self - organizing working groups consisting of specialists of various profiles.
Each Agile team consists of an analyst, developer, and tester. After the analyst formulates the terms of reference,

the developer proceeds to coding. Upon completion of coding, the tester performs testing to assess the quality of the new refinement within the framework of his or her module [14]. To do this, he or she manually goes through pre-prepared sets of functional tests (FT), aimed at detailed verification of a specific refinement within the framework of one module. Critical defects found during functional testing are immediately corrected within the team. All Agile teams work on the new version of the system in parallel, each within its own area of responsibility.

After all Agile teams complete their functional tests, the code is assembled. This means that, for example, all five branches that were developed and tested by Agile teams merge into one common branch. After building the code, the testing department starts full-fledged regression testing. The main purpose of this testing is to make sure that the fresh code of all five Agile teams participating in the project did not "break" the existing functionality and does not conflict with each other. In regression testing, great emphasis is placed on a comprehensive check of integration between all modules of the system [15] in order to identify all errors that could have occurred when merging the five branches developed by the corresponding Agile commands with the new code. Regression testing (RT) is a lengthy process that is carried out by the entire testing department for about a month. All bugs found during the regression testing are logged in the Jira bug tracking system. The most critical bugs, which are vital to to "roll out" a new version of the system to a productive environment, are fixed immediately, in parallel with the regression testing process. Each time the code with the corrected defects is "poured" onto the test environment, the testing department conducts short Smoke-tests in order to make sure that the fresh code did not break the main functionality of the system.

After the Testing Department successfully completes the regression testing, the manager responsible for the release decides on the release of the new version to a productive environment. After putting the new version of the system into operation, defects that the testers did not find in the FT and RT process are reported by the users of the system who encountered problems when using it - sellers, cashiers, service zone managers, storekeepers, online store buyers, etc.

3.2. Selection of evaluation criteria

One of the main goals of the testing department is to minimize the number of defects that users find during operation on a productive environment [16] because in the end, every missed defect at the testing stage costs a lot of money to the company: if the box breaks when trying to pay for the goods, the buyer will not make a purchase if the online store freezes, a potential customer leaves the site without waiting for the page to finally load and goes to another store. Therefore, the price of fixing a defect found before the release is much lower than the price of fixing the same defect, but found after the release of a new version. It is also true that defect detection at the early stages of testing gives significantly more time to fix it. Ultimately, this affects the price of repairing the defect. For example, if some critical defect of the system was detected at the stage of functional testing two months before the planned date of the new system version release to a productive environment, then the company will have enough time to fix this defect in the planned mode. And if the same defect was discovered at the stage of functional testing 3 days before entering the productive environment, then naturally, the cost of fixing will be much higher because it will be necessary to find a developer, remove it from other tasks, because there is no time for planned correction of the defect. All these factors certainly affect the cost of fixing the defect.

Thus, we can conclude that the earlier a defect is detected, the lower is the cost of fixing it. We take this formula as a basis and consider the process of searching for vulnerabilities in the new version of the system as the ratio of the number of detected defects to time.

3.3.Calculation of efficiency

To evaluate the effectiveness of the introduction of automated Smoke tests, it is necessary to collect statistics on the defects found over two time periods:

1. Testing period for a new version of the system until the introduction of automated Smoke tests

2. The period of testing a new version of the system after the introduction of automated Smoke tests

For this, it is necessary to upload for defects from the TestRail system. Due to the wide functionality of this system, it is possible to quickly generate a report on the given parameters and upload them in a convenient format for further analysis. Thus, complete unloading of system defects was performed for two releases for comparison. The data obtained are presented in the Table 2.

Table 2 Evaluation of the automated Smoke tests introduction effectiveness

	Stages											
	Development			**FT (functional testing)**			**RT (regression testing)**			**Productive**		
Week number	1	2	3	4	5	6	7	8	9	10	11	12
The number of defects found before the introduction of automated Smoke tests	0	1	1	4	5	3	13	11	10	8	14	17
Total for the stage:	2			12			34			39		
The number of defects found after the introduction of automated Smoke tests	0	1	1	12	18	15	7	8	6	3	5	6
Total for the stage:	2			45			21			14		

At the stage of functional testing, the number of detected defects in the system is less than at the stage of regression testing. The main flow of defects is detected at the stage of regression testing after manual passage of Smoke-tests. This can be explained by the fact that at the stage of functional testing, checks were carried out in parallel and in isolation for each module of the system. On the one hand, this is a plus since you can simultaneously test the development results of five separate Agile teams, but on the other hand, at this stage, the interaction of the updated system modules with each other is not checked at all.

Thus, after the code is assembled for five Agile commands and the regression testing stage begins, a large jump in the defects found related to the integration between the modules takes place immediately [17]. Key problems are identified with the help of Smoke-tests aimed at checking the operability of the basic functions of the system. Smoke tests are carried out during regression testing every time the code is corrected (for example, a defect was fixed, after which the system code was updated. In order to make sure that this code does not break the most important functions of the system, each time the test team passes Smoke tests). Typically, passing a Smoke test takes four testers on average 3-4 hours.

The process of passing Smoke tests runs parallel to regression testing, so it is necessary to understand that whenever there is a need for Smoke testing, it means that it will be necessary to distract several employees for almost half a day from performing regression testing (which certainly increases the time for passing regression tests).

Due to the fact that most defects are detected at the stage of regression testing, the problem of limited time and labor resources arises: the company does not physically manage to fix and test all the defects found because there is little time to fix it. Therefore, in the first place, the most critical defects are corrected, which is necessary for the new version to be installed in a productive environment. But still, some of the defects remain uncorrected and their correction is transferred to subsequent releases.

Thus, a new version of the software always enters a productive environment with several uncorrected defects. This does not affect the system's performance in the best way, since it partially affects the number of bugs found by the system users themselves during its operation.

The introduction of a system of automated Smoke tests dramatically changed the situation described above. Since automated tests are launched without human intervention, it became possible to conduct Smoke tests much more often (the company began to practice nightly launches of Smoke tests, which allows seeing test results by morning and immediately begin to solve the necessary problems). Also, it became possible to run automated Smoke tests already at the stage of the functional testing. This solution provided for the beginning of the identifying process of different system modules integration key problems at the very early stages of testing.

After the introduction of automated Smoke tests at all stages of testing, the vast majority of defects began to be detected at the first stages of functional testing, which allowed the company to fix significantly more defects in a

planned mode before the release of a new version of the system on a productive environment.

4. DISCUSSING THE RESULTS

To reliably confirm the statistical significance between the number of defects found before and after the introduction of automated Smoke tests, we calculate the paired Student t-test at each stage of testing.

The paired Student t-test is one of the modifications of the Student method used to determine the statistical significance of differences in paired (repeated) measurements.

Paired Student t-test is calculated according to the following formula:

$$t = \frac{M_d}{\sigma_d/\sqrt{n}} \qquad (1)$$

where M_d is the arithmetic mean of the indicators differences measured before and after, σ_d is the average square deviation of the indicators differences, n is the number of subjects.

Since the launch of automated Smoke tests began to be carried out at the stage of functional testing, we calculate the paired Student t-test for this stage:

1. We calculate the difference of each pair of values (d) in Table 5.

Table 3 Calculation of the difference between each pair of values at the FT stage

Week number	The number of defects found before the introduction of automated Smoke tests	The number of defects found after the introduction of automated Smoke tests	Difference (d)
4	4	12	8
5	5	18	13
6	3	15	12

2. Find the arithmetic mean of the differences by the formula:

$$M_d = \frac{\sum d}{n} = \frac{33}{3} = 11 \qquad (2)$$

3. Find the root mean square deviation of the differences from the average by the formula:

$$\sigma_d = \sqrt{\frac{\sum(M_d - d)^2}{\mathrm{n}-1}} =$$

$$= \sqrt{\frac{(11-8)^2 + (11-13)^2 + (11-12)^2}{3-1}} =$$

$$= \sqrt{7} \approx 2{,}65 \qquad (3)$$

4. We calculate the paired Student t-test:

$$t = \frac{M_d}{\sigma_d/\sqrt{n}} = \frac{11}{2{,}65/\sqrt{3}} \approx 7{,}19 \qquad (4)$$

5. Compare the obtained value of paired Student t-test 7.19 with a table value, which with the number of degrees of freedom f = n-1 = 3-1 =2 and the significance level p = 0.05 is 4.303. Since the obtained value is more than critical, we conclude that there are statistically significant differences between the number of defects found at the FT stage before and after the introduction of automated Smoke tests.

Similarly, we calculate the paired Student t-test for the stages of regression testing and operation on a productive environment. We obtain the Student t-test values for RT t = 4.92 and for the productive t = 5.77. Both values are more than critical (equal to 4.303), which indicates the presence of statistically significant differences between the number of defects found before and after the introduction of automated Smoke tests at the RT stage and at the operational stage in a productive environment.

We also calculate the time saved by hand testers through automating the Smoke testing process. As a rule, 4 testers were allocated to pass one Smoke test manually. The average time for which they fully completed all the tests was 3.5 hours. We calculate the total number of man-hours (mh) that was spent on manually passing one Smoke test:

$$MH = 4 \times 3{,}5 = 14 \qquad (5)$$

For one period of regression testing, Smoke testing was performed, as a rule, about 5 times. We calculate the total number of man-hours (mhtotal), which was spent on performing Smoke-tests for the entire period of regression testing:

$$MH_{\text{общ}} = MH \times 5 = 14 \times 5 = 70 \qquad (6)$$

Thus, we can conclude that the introduction of automated Smoke-tests saves 70 man-hours in the testing department for the entire stage of regression testing of the next version of the software product for solving other problems.

5. CONCLUSION

As part of this work, an analysis of the subject area was carried out, during which the main aspects of such an important stage of the software development life cycle as testing were considered. They also considered the basic principles of organizing an automated testing system in the IT departments of companies. It is concluded that in the IT departments of large companies, it is especially important to organize the process of specially organized automated testing.

Thus, from the introduction of a system to increase the efficiency of the software testing department due to a complex of automated Smoke-tests based on the selected testing tools, several positive effects were achieved at MBM LLC:

1. Now there is an opportunity to automatically run Smoke tests at any time of the day without distracting manual testers from performing other tasks.

2. Now there is an opportunity to pass Smoke testing much more often.

3. The time for manual testers to perform other tasks was freed up (70 man-hours for one stage of regression testing).

4. By saving time on passing Smoke tests, the time taken to complete regression testing was reduced.

5. Due to the fact that it became possible to implement runs of automated Smoke tests at the stage of functional testing, it became possible to identify system integration defects at the very early stages of testing the new version.

6. The number of defects found in a productive environment after the release of a new version was significantly reduced (by 25 defects).

REFERENCES

[1] H. Mohanty et al. (eds.) Trends in Software Testing. Springer Science+Business Media Singapore 2017. DOI: https://doi.org/ 10.1007/978-981-10-1415-4_2

[2] Salahirad, H. Almulla G. Gay, Choosing the fitness function for the job: Automated generation of test suites that detect real faults. Softw Test Verif Reliab. 2019; 29:e1701. DOI: https://doi.org/10.1002/stvr.1701

[3] Davis, Economic modeling of board test strategies. J Electron Test 5, 157–169 (1994) DOI: https://doi.org/10.1007/BF00972076

[4] Budnik, G. Fraser, F. Lonetti, et al. Special issue on automation of software testing: improving practical applicability. Software Qual J (2018) 26: 1415. DOI: https://doi.org/10.1007/s11219-018-9410-1

[5] M. Felderer & R. Ramler, Special issue on collaboration in software testing between industry and academia. Software Qual J (2017) 25: 1087. DOI: https://doi.org/10.1007/s11219-017-9395-1

[6] M.H. Vančová, Z. Kovačičová (2019) Challenges in Management of Software Testing. In: Kryvinska N., Greguš M. (eds) Data-Centric Business and Applications. Lecture Notes on Data Engineering and Communications Technologies, vol 20. Springer, Cham DOI: https://doi.org/10.1007/978-3-319-94117-2_7

[7] M. Peuster, M. Marchetti, G. García de Blas, et al. Automated testing of NFV orchestrators against carrier-grade multi-PoP scenarios using emulation-based smoke testing. J Wireless Com Network 2019, 172 (2019) DOI: https://doi.org/10.1186/s13638-019-1493-2

[8] H. Yenigün, C. Yilmaz, & A. Ulrich, J. Int, Advances in test generation for testing software and systems. Softw Tools Technol Transfer (2016) 18: 245. DOI: https://doi.org/10.1007/s10009-015-0404-z

[9] G. Alpaev (2017) Environment. In: Software Testing Automation Tips. Apress, Berkeley, CA. DOI: https:/doi.org/10.1007/978-1-4842-3162-3_3

[10] M. Felderer, & R. Ramler, Software Qual J (2016) 24: 519. DOI: https://doi.org/10.1007/s11219-015-9289-z

[11] Lazăr, S. Motogna, B. Pârv, Behaviour-Driven Development of Foundational UML Components, Electronic Notes in Theoretical Computer Science, Volume 264, Issue 1, 2010, pp. 91-105, ISSN 1571-0661, DOI: https://doi.org/10.1016/j.entcs.2010.07.007

[12] Bagus K. Manuaba, Combination of Test-Driven Development and Behavior-Driven Development for Improving Backend Testing Performance, Procedia Computer Science, Volume 157, 2019, Pages 79-86, ISSN 1877-0509, DOI: https://doi.org/10.1016/j.procs.2019.08.144

[13] K. Curcio, T. Navarro, A. Malucelli, S. Reinehr, Requirements engineering: A systematic mapping study in agile software development, Journal of Systems and Software, Volume 139, 2018, pp. 32-50, ISSN 0164-1212, DOI: https://doi.org/10.1016/j.jss.2018.01.036.

[14] H. Alahyari, R. Berntsson Svensson, T. Gorschek, A study of value in agile software development organizations, Journal of Systems and Software, Volume 125, 2017, pp. 271-288, ISSN 0164-1212, DOI: https://doi.org/10.1016/j.jss.2016.12.007

[15] S. Ali, Y. Hafeez, S. Hussain, et al. Enhanced regression testing technique for agile software development and continuous integration strategies. Software Qual J (2019). DOI: https://doi.org/10.1007/s11219-019-09463-4

[16] F. Sarro, "Predictive Analytics for Software Testing," in 2018 IEEE/ACM 11th International Workshop on Search-Based Software Testing (SBST), Gothenburg, Sweden, 2018 pp. 1-1. DOI: https://doi.ieeecomputersociety.org/

[17] Marijan, A. Gotlieb, , M. Liaaen, A learning algorithm for optimizing continuous integration development and testing practice. Softw: Pract Exper. 2019; 49: 192– 213. DOI: https://doi.org/10.1002/spe.2661