Regression Testing: Test Cases for Graphical Images

Janis Bicevskis[✉*,1], Edgars Diebelis[*,2], Zane Bicevska[*,1], Andrejs Neimanis[*,2]

[1] *University of Latvia, Raina boulevard 19, Riga, LV-1586, Latvia*
[2] *DIVI Grupa Ltd, Avotu street 40a-33, Riga LV-1009, Latvia*

Abstract

The paper proposes regression testing solution for systems with several thousands of test cases to be run. The system generates patterns - many various images consisting of SVG objects – lines, circles, etc. The image diversity makes impossible to carry out testing in a traditional way - by recording test cases and then replaying them. The authors propose (1) For each graphical image (pattern), image-specific parameters shall be identified, allowing for regression testing to automatically identify significantly different images from the benchmark images. The value of the parameter that describes the images is calculated first, such as the overall length of the image lines, the area, and so on. A slight difference between the result and the calculated parameter is acceptable. Otherwise, the test has revealed a non-compliance to be assessed by experts. (2) Support for testing (instrumentation) is incorporated into the system, including code fragments that allow changes in SVG objects' selecting order. This allows to track the process of generating a graphic image and thus identify the causes of the difference from the benchmark images. The proposed approach is demonstrated with the help of a practical example. Other areas of application could be, for example, in cartography, where differences in maps are assessed, or in medicine, where the state of human health is assessed after visual changes in organs.

Keywords

Regression Testing, Instrumentation, Development and Operations.

1. Introduction

The information system that serves as an inspiration for a new regression testing approach is intended for the construction of clothing pattern by individual customer size. Modellers design ever-new clothing models (there are currently more than 150 clothing models in the system) and publish them for purchase in an internet shop. Customers choose one of the models offered and submit their measurements, such as height, arm length, circumference, etc. (up to 47 parameters in total). The system generates a specific pattern of clothing using the customer's measurements (see Figure 1). New clothing models are being constructed, which in turn results in system changes and the need for regression testing. In addition, the system must ensure the stability of generated patterns as the system changes: the regenerated patterns may not differ significantly from the primary after modification of the system.

The main problem for regression testing lies in the number of test cases. Each graphic image (pattern) consists of ten and more pieces and generating them requires considerable computing resources (generating one pattern requires about 100 sec). All models (more than 150 in total) must be tested for

Baltic DB&IS 2022 Doctoral Consortium and Forum, July 03–06, 2022, Riga, LATVIA
✉ Corresponding Author.
* These authors contributed equally.
✉ Janis.Bicevskis@lu.lv (J.Bicevskis); Edgars.Diebelis@di.lv (E.Diebelis); Zane.Bicevska@lu.lv (Z.Bicevska); Andrejs.Neimanis@di.lv (A.Neimanis).
ORCID 0000-0001-5298-9859 (J.Bicevskis), 0000-0002-5950-9915 (E.Diebelis), 0000-0002-5252-7336 (Z.Bicevska), 0000-0002-3320-8209 (A.Neimanis).

all possible measurement sets (currently more than 150). The Figure 2 shows that pieces of the same pattern can substantially vary according to different measurements.

This size of work (more than 22500 test cases) requires huge volume of computational resources (a brief estimation: 22500 test cases * 100sec per test case = ~ 700hours). This was also demonstrated by expert assessment and practical experience in testing the system. The variety of patterns does not allow a significant reduction in the number of test cases, since the pattern images depend on the size of the body and the pattern. In addition, the number of clothing models is gradually rising.

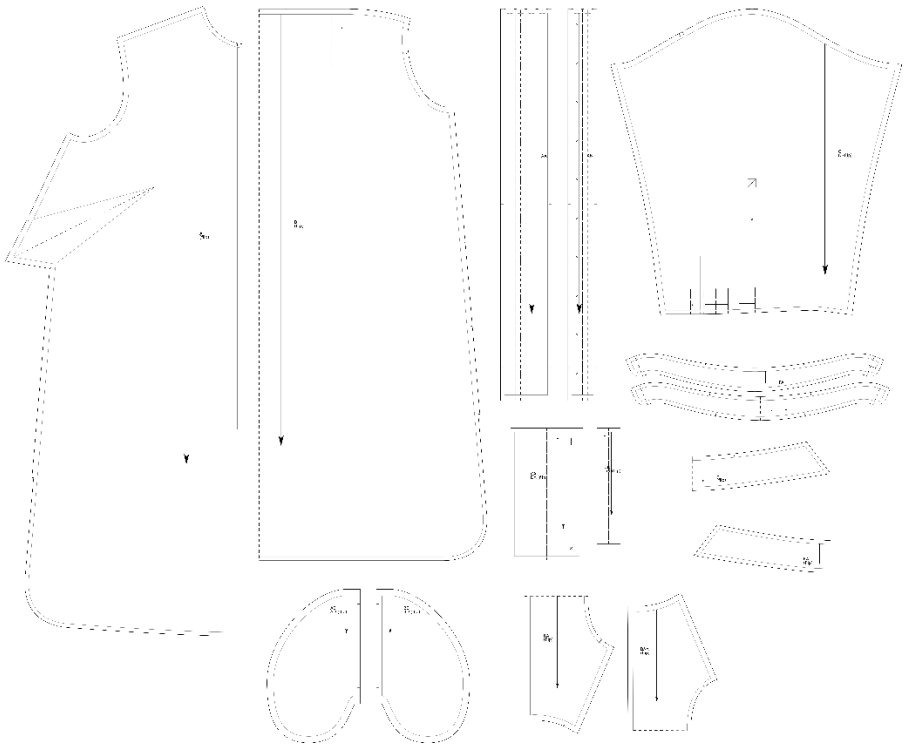

Figure 1: Example of a pattern for a specific model for a specific customer

Evaluating of test results is difficult. Many personal working hours are needed to check the compliance of test results with benchmark values (visual comparison of graphical objects) for 22500 cases. Since regression testing must be carried out regularly and repeatedly, a specific approach to regression testing should be sought.

In the given study, the pattern generation system covers both the domain-specific language for describing the patterns and the software that supports the use of this language. The modeller using the domain specific language creates a pattern script - describes the algorithm for construction of a pattern for the specific model in variable sizes. A system's component *Interpreter* generates SVG objects by executing the pattern scripts and draws patterns as graphic images. Thus, testing applies not only to software, but also to pattern generation scripts stored in a database, and testing should be seen not only as testing of separate software modules but as a system-wide checking.

Nowadays, the main goal of software testing is to provide reliable software that could be used properly in everyday life, but this goal is not succeeded yet. Despite numerous resources spent on testing, errors and bugs in software are still causing system failures [1] and [2]. Traditionally, testing is positioned as a separate phase of software development during which the quality of software is improved, sufficient to enable software to be used effectively. In this case, regression testing must be performed on the entire system, both software and scripts, the number of which is constantly increasing. Therefore, this study is based on the DevOps approach [3].

Development and use (operation) are not divided in separate phases but they are mutually integrated, continuing system development and repeating system testing without disrupting its use. The case differs from the traditional regression testing because of the huge number of test cases and because the result of the system work are graphic images that are difficult to automatically compare. The dif-

ference between the proposed approach and the model-based testing [4] is that the full coverage of scripts does not guarantee the quality of testing and, thus, is not sufficient. The script execution order plays also a crucial role since the same scripts may be used more than once for the same model.

The test case base is created by adding all the measurement sets entered by the customers. It is assumed that customer satisfaction with the patterns received shows their correctness. For each graphic image (pattern), image-specific parameters are arranged to allow the identification of significantly different images from benchmark images in regression testing. When comparing the corresponding images, the value of the parameter that describes the images is calculated, such as the overall length of the image lines, area, etc. If a minor difference is found, it is considered that the system still works correctly. Otherwise, the test has revealed a non-compliance to be assessed by an expert.

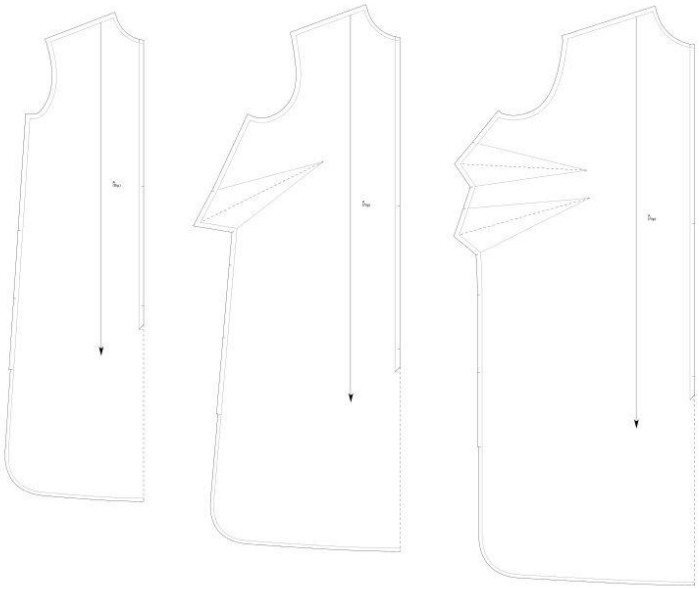

Figure 2: Variability of a single pattern piece, depending on the individual's measures

Moreover, support for testing (instrumentation) is incorporated into the system, including code fragments that allow changes in SVG objects' selecting order. This allows to track the process of generating a graphic image and thus identify the causes of the difference from the benchmark images.

The proposed solution may be topical in testing other graphical imaging systems where a large number of dynamically accumulated tests are required. The approach offers not only the accumulation of a large number of expert-approved test cases but also the automated comparison of graphical images using image parameters - image line lengths, image areas, etc. A similar situation is observed in cartography where map changes are assessed, or in medicine where the state of human health has been assessed after visual changes in the organs.

The paper is structured as follows: Section 2 is devoted to an overview of similar studies by other authors and compares them with the solution proposed in this paper. Section 3 describes the methodology proposed. Section 4 gives analysis of findings of proposed methodology. Section 5 contains conclusions and future study.

2. Related research

2.1. Traditional testing approaches

According to testing literature [5], [6] and standards [7], the testing contains several steps: unit testing, integration testing, system testing, acceptance testing, regression testing, etc. The problem identified in the introduction refers to three specific system features: (1) the system continues to develop on a long-term basis as new clothing models and patterns are added to it, (2) the system must be

stable - if the patterns are regenerated, the results may only be slightly different from the previous ones, (3) the system's testing requires big amount of computing and personnel resources, hence special solutions for regression testing should be sought.

System testing is a black box testing method used to evaluate the completed and integrated system, as a whole, to ensure it meets specified requirements [8]. The functionality of the software is tested from end-to-end and is typically conducted by a separate testing team than the development team before the product is pushed into production.

In this case, the system testing should not be considered as a single step to be executed after integration testing that is followed by acceptance testing. It should be repeated on a regular basis and is caused by new scripts and pattern construction cutting methods used to create new clothing models. It is not enough to have regression testing for new system source code versions, all the system must be repeatedly retested – both the programs and pattern generation scripts.

Regression testing is an important and expensive activity that is undertaken every time a program and scripts are modified to ensure that the changes do not introduce new bugs into previously validated system. Instead of re-running all test cases, different approaches were studied to solve regression testing problems [9]. Data mining techniques are introduced to solve regression testing problems with large-scale systems containing huge sets of test cases, as different data mining techniques were studied to group test cases with similar features in equivalence classes. Dealing with groups of test cases instead of each test case separately helped to solve regression testing scalability issues. Unfortunately, the interesting idea proposed is difficult to apply in the pattern generation system because the equivalence classes will vary depending on the models (the total number of equivalence classes shall be assessed as: 10 * 150 = 1500). The number of test cases is huge because it consists of a multiplication of the number of equivalence classes and the number of test cases in the equivalence classes (1500 * 10 = 15000).

In this paper, we propose another methodology for regression testing of large-scale systems using a specific pattern parameter – the total area of pattern. Minor differences in the overall area of patterns for the same clothing model in the same size are allowed. Although this does not allow to reduce the number of test cases substantially, it offers the possibility of automating regression testing with comparisons of graphical images (SVG objects).

In addition, it should be noted that over the years, the aim of testing has been changed. Testing as an error searching has been replaced by the requirement to make sure the system is ready for use. Detection of errors is one of the methods of verifying the quality of the programmes to make sure that the system is ready for use; unfortunately, if no errors are discovered, it does not guarantee the absence of errors [6]. In this case, a system stability is required: All system use cases that have been accepted should remain correct throughout the life cycle of the system. The stability requirement and repeated regression testing ensure that system testing is complete.

2.2. Testing in DevOps processes

The proposed regression testing approach complies with DevOps principles [10]. DevOps claims removing the barriers between traditionally separated software development phases and operations [3]. Under a DevOps model, development and operations teams work together across the entire software application life cycle, from development and test through deployment to operations.

The nine pillars of DevOps [11] are leadership, collaborative culture, design for DevOps, continuous integration, continuous testing, elastic infrastructure, continuous monitoring, continuous security, and continuous delivery. In this study, the most important are the following five key practices of DevOps:

1. **Continuous Integration.** Unlike traditional systems development practices - waterfall model [8], "V" model [6] - where integration and system testing are carried out once followed by acceptance testing, in DevOps case, the integration testing will continue throughout the entire life cycle of the system. This is due to the development and application of ever-new pattern models and modelling methods to meet customer requirements.
2. **Automated Testing.** Successful use of the system leads to an increase in the number of models and customers, while ensuring the stability of the system; re-requested patterns may only

slightly differ from previous versions of the same pattern. Consequently, the number of different use cases of the system may increase significantly, and the traditional methods of regression testing should be adapted to DevOps for automated testing with large number of test cases.

3. **Continuous Delivery.** The system is constantly being supplemented with new models. At the same time, it is necessary to ensure that the system stability characteristic described above, which requires significant resources.
4. **Continuous Deployment.** Successful applications of the system make it possible to extend its availability. This in turn leads to an increase in the size of the system, additional models, and customers; the amount of work to be carried out in regression testing rises.
5. **Continuous Monitoring.** Each new successful use case, when the customer is satisfied with the patterns received, extends the use of the system. Each case where the customer is not satisfied with the result causes the expert to analyse the causes of the failure and to improve the system, which requires the release of a new versions and regression testing.

DevOps has a promising future with numerous solutions of IT problems [10]. DevOps is an approach that is now adopted by many IT companies to provide reliable and faster solutions to their clients.

3. Methodology

This chapter is devoted to the proposed methodology and includes description of the main components: the architecture of the solution, the usage of use cases as test cases in regression testing, the invariants of graphic images and the application for instrumentation in support of regression testing.

3.1. System architecture

The system architecture is given in the Figure 3. The system consists of two modules: (1) pattern design, used by *Modelers*, and (2) pattern generating, used by *Clients*. *Modelers* describe clothing patterns using domain-specific language statements. The result is an algorithm of pattern generation - a script consisting of statements and logical operations that is stored into the *Models DB*. The scripts define the process of pattern generation according to the individual measurements of a particular client. Each script has a tree structure with branches that describe single-pattern generation process for one set of client measurements.

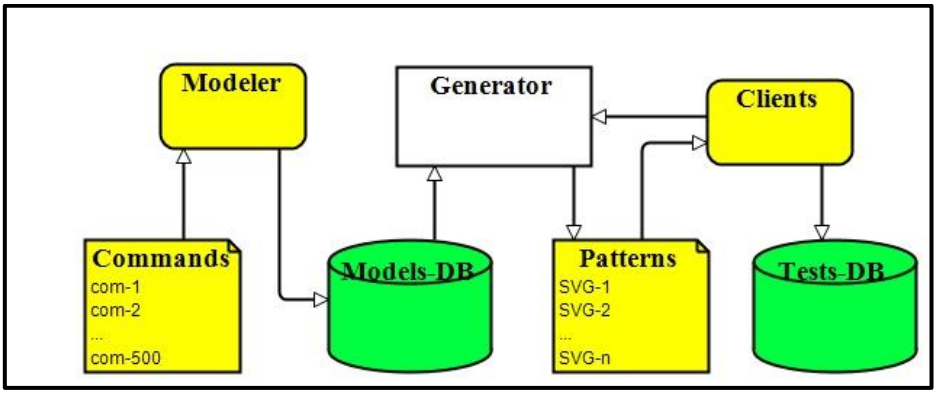

Figure 3: Pattern Generation System Architecture

The *Client* chooses a pattern and enters his/ her measurements into the system. The *Generator*, using a script from *Models DB*, generates an individual pattern for the specific client. The process described will hereinafter be called a *Use Case*. A generated pattern together with the respective measurements may become a test case if the conditions described in the next chapter are valid, the tests are stored into the *Tests DB*. Regression testing using the accumulated tests will provide a system stability

test against the effects caused by the changes as the benchmarks and the generated patterns may only differ insignificantly.

3.2. Use cases and test case accumulation

In the proposed approach, tests are not prepared based on requirement specifications or test models (as it is usually the case in a traditional testing process [6], but they are accumulated gradually, evaluating each specific use case. If the generated pattern corresponds to customer's expectations/requirements, the use case may become a test case. Otherwise, there are discrepancies detected in the operation of the system that may result in incorrect operation of both programs and scripts.

Of course, the total number of tests accumulated in this way may increase significantly and therefore repeated manual testing may be technically impossible. Consequently, regression testing should be able to automatically accept the correctness of the system in most predetermined use cases by filtering cases where significant differences from the benchmark have been detected. The proposed approach is in line with the principles of agile development and DevOps [8], which require close communication with end-users to obtain their confirmation of the correct functioning of the system.

It is proposed to create equivalence classes of test cases including in one class the use cases that are processed by the *Generator* using the same branches of scripts. Usage of equivalence classes would limit the number of regression test cases. This approach allows all use cases to be accumulated at the initial stage of the system use, and to subsequently develop equivalence classes of test cases for regression tests in later stages. Only realistically, once completed use cases are included into the test cases set, in contradiction to the traditional testing process with artificially created test cases to cover all branches of scripts.

However, the proposed approach is not universal and is applicable in cases of graphic images where image evaluation is labour-intensive and manual. The proposed approach is, in the authors' view, also applicable to map processing systems, medical imaging, and other cases.

3.3. Invariants of graphical images

The proposed method uses numerical criterion of similarity between the benchmark image and the regenerated image. If the benchmark image and the regenerated image differ significantly, i.e., the image values exceed the critical limit, a new use case has been discovered. Such a situation has been caused by changes in the system (software, scripts) and the correctness of the changes must be verified by the client.

Table 1
Total pattern area depending on body sizes

Name	Height	Benchmark image (pixels)	Regenerated image (pixels)	Diffeence (pixels)	Difference (%)
Elīna	176	1292387	1292319	68	0.0053
Gunta	170	1378639	1378557	82	0.0059
Vineta	172	1438525	1438303	222	0.0154
Vita	172	1572778	1572506	272	0.0173
Anda	173	1161485	1160793	692	0.0596
Elza	174	1344000	1343299	701	0.0522
Patricija	158	994787	1000932	-6145	0.6177
Dina	166	1178915	1171960	6955	0.5935
Ulrika	164	1352851	1306159	46692	3.5748
Katrīna	163	1343905	1288067	55838	4.3350

Unlike data processing systems, where numerical values typically are used as benchmark values, it is very difficult to find a criterion in image processing that describes the similarity of images. In the

case of a pattern generation system, computable parameters, such as the number of pattern pieces, the total area of the pieces, the number of pattern lines or the total length of lines, may serve as such numerical criterion. Let's look at a small example.

The example shows regression testing results for one clothing model (given in Figure 1) for 140 different customers. The table contains only 10 of 140 body measurement sets that are in use. The overall area of the image/pattern was selected as a criterion for image similarity. The justification for this choice requires additional research on data mining methods, the results of which will be the content of another publication. However, the overall area of the image as a criterion for the similarity of graphic images has proven to be one of the simplest in practical use.

Five cases were found according to the criterion where the benchmark image and the generated pattern image differed by more than 40,000 pixels (two cases in the Table 1 – Ulrika un Katrīna); in ten cases the difference was in the interval from 1000 to 40,000 pixels (Patricija un Dina); in sixteen cases the difference was in the interval from 500 to 1000 pixel (Anda un Elza); in all other cases, the difference was less than 500 pixels (Elīna, Gunta, Vineta, Vita). Industry experts selected the 4,000-pixel difference as critical because these images showed significant differences between the benchmark image and the generated pattern image. However, it is too early to conclude from this example that the regression testing work can be reduced by more than 90%. Further studies are needed.

3.4. Instrumentation

One of the problems described above is how to track the progress of the pattern designing if the generated pattern does not address the requirements of the client or if the regression testing reveals significant differences between the re-generated pattern and its benchmark. This task can be addressed by means of source code instrumentation by including support for testing and identifying the image differences within the system itself.

Instrumentation of software source code to create mechanisms for monitoring of software execution paths is known since the end of 60-ties. The so called "program observers" very built into the programs to let the system print out the current memory content (image) and therefore decide about the correctness of the system to be tested.

Today, instrumentation is often used to collect information about program performance and to analyse it [12]. Time measurement calls are inserted in specified point of the source code, and it lets to identify the performance bottlenecks whilst the program is executed with representative input data. However, points to the limits of application of instrumentation in performance assessment as the instrumentation may put a significant additional load to the system to be analysed, thereby preventing the collection of objective performance measurement data.

Instrumentation is often used in the development of real-time systems [13], trying to produce a minimal impact on the system being developed. Instrumentation allows recording of crucial process events with insignificant impact on the operating time of the system, which is a crucial component in real-time systems.

In the specific pattern generation task, the instrumentation is used to facilitate the identification of non-compliance cases when the generated pattern significantly differs from the benchmark pattern. The system instrumentation allows to identify the script execution sequence, and it in turn allows to track the generation of SVG objects. However, the existence of such a mechanism may unnecessarily burden system resources in cases where the attachments comply with customer requirements (in more than 90% of all use cases). It is therefore proposed to use modified, operationally guided instrumentation, the ideas of which have been published in [14].

The operationally guided instrumentation uses the concept of system enforcement priority. A system enforcement priority is passed on to each system execution session, such as 0, 1, 2, … Each instrument also has a fixed priority defined by its programmer, for example, 1, 2, 3, … Only instruments whose priorities are matched or less than the enforcement priority is executed in a specific session. For example, if the enforcement priority is "0", no instrument is executed. If the enforcement priority is "1", all instruments with priority "1" are executed. If the enforcement priority is "2", all instruments with priorities "1" and "2" are executed etc. This approach provides several opportunities – (1) to de-activate the instrumentation during daily use of the system, which allows saving of re-

sources, (2) to activate the instrumentation in cases when the generated patterns differ significantly from the benchmark patterns, (3) to output additional information on generation of SVG objects by changing the enforcement priority and activating the instruments with higher priorities if necessary.

Although the implementation of operationally controlled instrumentation is not complicated, its applications may be encountered very rarely. This is due to the remoteness of the system development and use processes already mentioned. The development of the DevOps concept is expected to lead to a wider deployment of operationally controlled instrumentation.

4. Research findings

4.1. Generalizations options

The proposed regression testing solution for a system designed to construct patterns is consistent with the DevOps principles. The traditional system development model, when the development and operations groups are separated both in time and organisational terms, creates serious obstacles to the use of the DevOps principles.

A similar situation as in the fitting patterns design system can also be observed in the drawing up of geographical maps. The number of different maps is huge, they are constantly changing due to the change in nature and their placement: new buildings and roads are being built, forests are being cut and planted, etc. This changes the objects on the map. To get a new set of maps that match the actual state in nature, you need to compare the previous map versions with the new ones. They can be slightly different, for example, because of differences in map design or improved measuring technologies etc. The importance of differences can be assessed by industry professionals. Yet the challenges to be addressed are like those of generating the patterns: (1) there are many maps, (2) maps as graphical images can slightly vary between versions, (3) manually comparing cards is impracticable and needs to find options for testing automation, (4) industry professionals need a support in the system for evaluation the correctness in case of differences between various versions of maps.

The industry that faces similar problems with graphic imaging is medicine. Each patient is a unique person with individual parameters such as height, the size of organs etc, and human bodies are timely variable. To detect deviations from the norm, a large number of graphic images must be processed comparing the current images to the previous ones or standard images. The proposed method allows each image to be automatically compared to previous images and reveal the case when image differences are significant.

4.2. Limitations

Three factors should be recognised when analysing the weaknesses of the proposed method:

1. The amount of work to be carried out by professionals to maintain the stability of the system is significant: the industry expert should assess all the prepared patterns and accept/ reject them for further usage in regression tests.
2. The testing solution's performance is a key factor as thousands of test cases must be executed and evaluated, which is impossible to carry out without an appropriate testing automation.
3. The system should be supplemented with support features/instrumentation to be able to track the process of generating graphical images and to discover the causes of incorrect cases.

Each of these factors requires additional functionality, i.e., causes additional work and costs. The design of the system already requires additional options to be addressed at a later stage during operations. Such an approach is rare in the contract-led development as developers usually implement the requirements of the specifications without caring about the implementation of the DevOps principles.

5. Conclusions

Authors have offered a special approach to the development of systems according to DevOps principles. Unlike traditional systems regression testing where test cases are constructed according to sys-

tem specifications, in this case it is proposed to accumulate use cases from previous system applications. Each use case accepted by a customer, or an expert may become a regression test case, it ensures stability of the system as all previous use cases must work correctly after changes in the system. This will not only save the resources needed for preparation of test cases. It also ensures that the system is maintained in good quality as all previous use cases are run repeatedly. Customer-accepted test cases differ from the, often unrealistic, test cases created by testers according to system requirements specification.

Regression testing automation is achieved by using imaging evaluation parameters instead of comparison of graphic images. The parameter values are calculated first and the comparison of the images themselves is replaced by a comparison of the values of the parameters, a slight difference of which is allowed. This significantly reduces the number of personnel resources necessary for checking the match of images.

Support for identifying programs and script errors is achieved through instrumentation of programs that allow to track the progress of creating graphical images. The conventional instrumentation is proposed to be improved by an operationally controlled instrumentation with prioritization mechanisms for different usage modes.

The proposed solution is in line with the DevOps principles as the system's performance assessment is carried out immediately after the change and using previously accepted use cases. This allows for the integration of system verification and validation processes by not dividing the testing of the system during the development from the retesting of the system during its use.

6. Acknowledgements

This work has been supported by University of Latvia projects: AAP2016/B032 “Innovative information technologies".

7. References

[1] V. Garousi, M. V. Mäntylä, When and what to automate in software testing? A multi-vocal literature review. Information and Software Technology, 76, (2016) 92-117.

[2] S. Tahvili, M. Saadatmand, M.Rohlin, W. Afzal, S. H. Ameerjan, Towards execution time prediction for manual test cases from test specification. In Proceedings of 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA), IEEE, (2017). 421-425.

[3] CloudBees homepage. DevOps Methodology: Understanding the Approach and Philosophy. https://www.cloudbees.com/blog/devops-methodology-understanding-the-approach-and-philosophy. Last accessed 2022/03/08.

[4] I. Schieferdecker, Model-based testing. IEEE software, 29(1), 14.

[5] Spillner, A., Tilo Linz. (2021). Software Testing Foundations, 5th Edition, Rocky Nook, (2012). SBN: 9781681988559.

[6] ISTQB. ISTQB Foundation Certificate in Software Testing. (2019). https://www.qa.com/course-catalogue/courses/istqb-foundation-certificate-in-software-testing-inc-exam-stf-2/?learningMethod=Virtual&

[7] ISO/IEC/IEEE 29119-2:2021. Software and systems engineering - Software testing - Part 2: Test processes. (2021). https://webstore.ansi.org/Standards/ISO/ISOIECIEEE291192021-2455205

[8] R. Pressman, M. Bruce. Software Engineering: A Practitioner's Approach. 9th Edition. McGraw Hill. (2020).

[9] P. Kandil, S. Moussa, N. Badr, Regression Testing Approach for Large-Scale Systems. In Proceedings of IEEE International Symposium on Software Reliability Engineering Workshops, (2014). 132-133. doi: 10.1109/ISSREW.2014.96.

[10] Muñoz, Mirna, Negrete, Rodríguez, Mario. A guidance to implement or reinforce a DevOps approach in organizations: A case study. Journal of Software: Evolution and Process. 2021. DOI: 10.1002/smr.2342.

[11] M. Hornbeek, Best of 2021 - 9 Pillars of Engineering DevOps With Kubernetes (2021). https://containerjournal.com › editorial-calendar › 9-pillar.
[12] STACKIFY homepage. The Ultimate Guide to Performance Testing and Software Testing: Testing Types, Performance Testing Steps, Best Practices, and More. (2021). https://stackify.com/ultimate-guide-performance-testing-and-software-testing/#wpautbox_about, last accessed 2022/03/08
[13] S. Fischmeister, P. Lam. On Time-Aware Instrumentation of Programs. In: Proceedings of 15th IEEE Real-Time and Embedded Technology and Applications Symposium. (2009) 305-314. doi: 10.1109/RTAS.2009.26.
[14] Ya. Ya. Bichevskii, Ju. V. Borzov, Priorities in debugging of large software systems. Programming and Computer Software. 8(3) (1983). 129-131.