Artur SOSNÓWKA[1]

[1]West Pomeranian University of Technology, Faculty of Computer Science and Information Technology

Visual analysis for Low Level Test Cases in Test Project

Abstract. *Although the majority of software testing in the industry is conducted at the system or acceptance level, most formal research has focused on the unit level. As a result, system and integration level testing techniques are only described informally. This paper presents general visual analysis for low level test selection based on inputs from available Test Management system. A presented analysis criterion includes small subset of test metrics that can be used as a base for further development of the test suites.*

Streszczenie. *Mimo, iż większość testów w przemyśle przy tworzeniu oprogramowania jest przeprowadzana na poziomie systemu lub testów użytkownika, formalne badania koncentrują się dotychczas na poziomie Unit-Test. Wynikiem tego jest stosunkowo słaby opis formalny testów integracyjnych lub systemowych. W artykule tym przeprowadzono ogólną analizę wizualizacyjną opartą na danych pobranych z narzędzi do zarządzania testami. Zaprezentowane kryteria zawierają zbiór metryk testowych, które mogą być używane w dalszym procesie rozwoju bazy testów. (**Wizualna analiza testów niskiego poziomu w projekcie testowym**).*

Keywords: Visualization metaphor, Test City, Test Archaeology, Test metrics, Test case visualization.
Słowa kluczowe: Metafora wizualizacyjna, miasto testów, archeologia testów, metryki testowe, wizualizacja testów.

Introduction

Software development is dealing with growing complexity, shorter delivery times and current progress made in the hardware technology. The biggest, however, not directly seen part of the software lifecycle is the software maintenance. Increasing number of systems used in the corporation and tolerated number of deviations is decreasing when time progressing and users get trusted to the used software. As soon as software is put in the production environment, every big change or even a small adaption of the source code can cause potential danger, in best case; monetary, in worst; image or even loses of human being. The maintenance is provided during the entire period by different groups of technicians or business partners. This makes the task of understanding, programming and maintaining the system source code and its testware more complex and difficult.

A description of the behaviour and possible use cases in the system to be developed is in a state of constant change during the whole project and system software lifecycle. Those changes are based on legal, business, functional, or software architectural needs (e.g. new programming techniques). Required new functionality is gaining focus while the old one is put a side and threated to be not as important as before. Requirements validation and verification process is a part of the whole test management process in which the high (HLTC) and low-level test cases (LLTC) [4], are focusing, in this case, on old but still valid functionality. Afford to handle growing number of an old and new HLTC and LLTC keeps going to be not affordable, or getting be forgotten by purpose. The situation is causing raised maintenance costs to the limit, when new development can produce less costs and even be easier to implement rather than the creation of a new functionality within the old system.

Software quality is according to IEEE definition:

1. The degree to which a system, component or process meets specified requirements.
2. The degree to which a system, component or process meets customer or user needs or expectations [2].

The above given definition is obligating quality assurance teams to perform planned and systematic pattern of actions, and to provide adequate confidence to the product or item that it conforms to established technical requirements [2]. Execution of needed actions to provide at least the same quality during the whole maintenance phase is a big cost factor. Big and complex systems are providing a large number of functions and demanding even larger number of tests. To provide 100% fulfilment, the test team has to ensure that each single functionality is not affected through the code adaptation and its side effects. Necessary actions, and test executions, are provided based on regression test, which gives an overview of the system quality after adaptation. The adaptation of the system causes the demand to adapt an adequate set of tests to fulfill its requirement for the current system. Adaptations are performed based on documented change and stored in the test base, which is maintained with the help of test management software. Even the best managed test base, after few years of usage, is not free of tests; which are too old, obsolete, duplicated or there are no tests for demanded functionality. Those tests cause additional efforts and are not providing expected fulfilment for quality needs. Detection of the problems within a test base can save much effort and reduce necessary maintenance costs.

Each part of the software development process involves resources with different budgeting scope, starting from staff, hardware, software and finalizing on license costs.

According to the software development lifecycle, the last, longest and most costly part of it is the maintenance. The prevailing notion is that software is easy and cheap to change but this is seldom the case. Software maintenance can account for 60 to 80 per cent of the total life cycle cost of a system. Most of the expenditures, as much as three fourths of the total maintenance costs, are for enhancements to the code, rather than correction of defects.

Developers and managers believe that a required change is minor and attempt to accomplish it as a quick fix. Insufficient planning, design, impact analysis and testing may lead to increased costs in the future. Over time successive quick fixes may degrade or obscure the original design, making modifications more difficult [5] therefore finishing in not acceptable, resulting in a low quality system.

Two observations lay the foundation for the enlightened view of testing as an investment. First, like any cost equation in business, we will want to minimize the cost of quality. Second, while it is often cheaper to prevent problems than to repair them, if we must repair problems, internal failures cost less than external failures, especially during the maintenance.

There are however projects which cannot afford to create completely new software and gaining the problems to get satisfying quality within specific budget. The number of available LLTC is much bigger than its available execution time. The projects are coming into the problem to select proper test cases.

Testing in the software development lifecycle

Software development as a process is described in ISO/IEC 12207:2008 for "Systems and software engineering – Software life cycle processes" standard [1]. The standard has the main objective of supplying a common structure so that the buyers, suppliers, developers, maintainers, operators, managers and technicians involved in the software development are using a common language [2].

The primary lifecycle process is divided into five different main processes involved in software product creation and covers a very large area; therefore it is necessary to define a scope in order to differentiate needed details. For detailed definition of the scope please follow ISO/IEC 12207:2008 [1].

Based on the definition for the primary lifecycle we can treat the software/programs as an independent, instances for which testing is a part of each Development, Operation and Maintenance process.

Understanding of Software Testing mostly consists of executing test cases for the system undergoing testing and giving the feedback about working and faulty parts of the system. The test phase, whether it is called Function Test, Component Test, Integration Test or Acceptance Test, consists of 8 generic steps (see Figure 3).

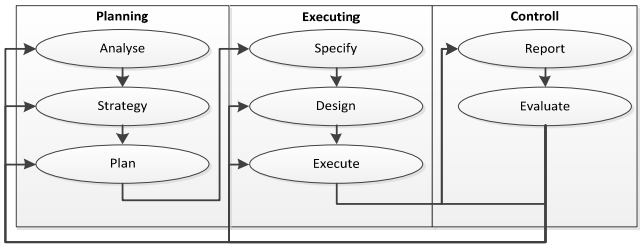

Fig.1. Test process

Those steps are included in a more general definition as:

- Planning and controlling.
- Documentation review and static analysis (known as static testing).
- Design and test execution (known as dynamic testing).
- Result checking.
- Reporting on test process and system under test.
- Evaluation of exit criteria.

All of the above mentioned activities are used to improve both; system being tested, development or testing process itself.

Testing can have several objectives:

- Finding defects (Error detection).
- Acquire trusting about the quality level.
- Support process of decision-making.
- Avoiding defects [3].
- Validate that what has been specified is actually what the user wanted.
- Verify that software behaves as specified.

In other words, validation checks to see if we are building what the customer wants/needs, and verification checks if we are building that system correctly. Both verification and validation are necessary, but require different components testing activity. This is a dynamic process driven by the system adaptation necessity and human needs.

The definition of testing according to the (IEEE, 1059-1993 - IEEE Guide for Software Verification and Validation Plans [5]) standard is that testing is the process of analysing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item.

Test metrics

To be able to perform a quantified decision, defined set of the static and dynamic data of the testware has to be prepared. Based on the available information for LLTC we can extract a set of basic test metrics:

- Amount of LLTC
- Execution status for available LLTC
- Last modification date
- Amount of LLTC per Unit/Functionality
- Costs of test planning and preparation
- Costs of test execution, defect tracking, version and change control.

Dependent on the metrics type, those are to be taken as data export from the test, defect management tool or even statistical data. In our approach we are extracting necessary data from available LLTC testware at User Acceptance and System Integration level. Available metrics can be mapped into the chosen visualization metaphor as:

- Data physical properties (colour, geometry, height mapping, abstract shapes)
- Data granularity (unit cubes, building border or urban block related)
- Effect of Z axis (height) mappings on the image of the city
- Abstraction of data is key issue
- Resulting "data compatible" urban models are much larger than the original VR (Virtual Reality) urban models.

Visualization metaphor

A visualization metaphor is defined as a map establishing the correspondence between concepts and objects of the application under test and a system of some similarities and analogies. This map generates a set of views and a set of methods for communication with visual objects in our case - test cases [22].

An important innovation of computers is that they can transform any media into another. This gives us the possibility to create a new world of data art that the viewer will find as interesting. It does not matter if the detail is important to the author; the translation of raw data into visual form gives a viewer possibility to get info, which is most important just for him. Currently, numerous existing visualization systems are divided into three main classes:

- scientific visualization systems;
- information visualization systems;
- software visualization systems.

Although all visualization systems differ in purposes and implementation details, they do have something in common; they manipulate some visual model of the abstract data and translate this into a concrete graphical representation.

In this paper we are not aiming to present all possible visualization metaphors, as this is not the focus for our research. We would like to show a basic and easy to understand metaphor which is helpful for representation specific test data.

City metaphor

After some of the previous research work, which is however not the focus of this paper, we settled our first attempt to the metaphor which is very widely presented in [1] and is a part of Phd from Richard Wettel [16]. In its research and implementation for software, source code classes are represented as buildings located in city districts, which in turn represent packages, because of the following reasons:

- A city, with its downtown area and its suburbs is a familiar notion with a clear concept of orientation.
- A city, especially a large one, is still an intrinsically, complex construct and can only be incrementally explored,

in the same way that the understanding of a complex system increases step by step. Using an all too simple visual metaphor (such as a large cube or sphere) does not do justice to the complexity of a software system, and leads to incorrect oversimplifications: Software is complex; there is no way around this.

- Classes are the cornerstones of the object-oriented paradigm, and together with the packages they reside in, the primary orientation point for developers [16]

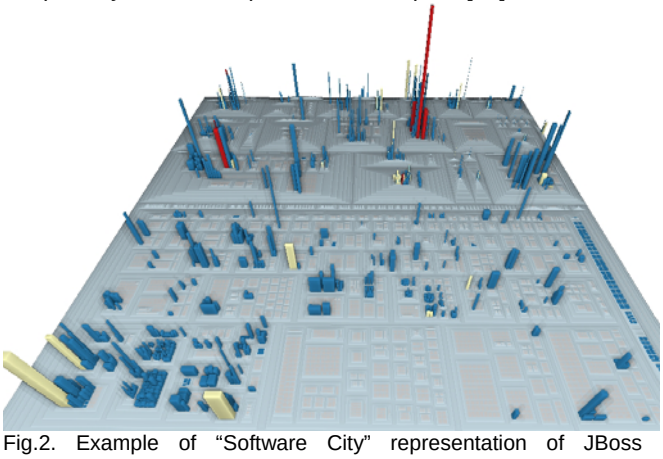

Fig.2. Example of "Software City" representation of JBoss application server

In this paper we discuss an approach to visual analysis of low-level test cases in order to support the selection of appropriate LLTCs [4] for a software product or system.

Test selection and test mining

Test case selection is a very important part of the software test process and development [17] (see Figure 4 - Strategy). The problem for selecting a small set of tests from a large test base such that the most defects are revealed when this subset is executed is occurring at each stage of the test process, whether in Unit, Interface, System or Integration Test. Test case prioritization is a problem of finding an optimal scheduling of the tests in a test suite so that the number of defects found earlier during testing is maximized.

Collect information

The importance of information collection for test design and creation is crucial for further test process steps. At the first stage it is important to identify and collect all information needed about the tested Features/Functions. For both of them the test project is obligated to perform validation and verification to estimate gained quality.

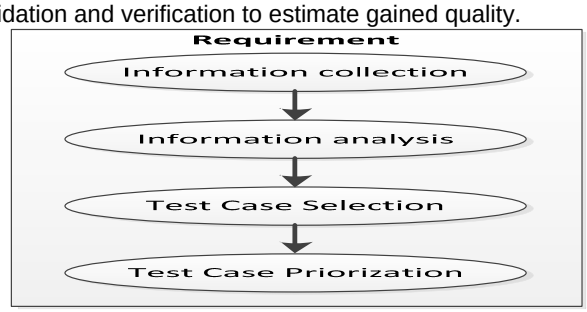

Fig.3. Test selection process based on requirements analysis

In a large number of test projects, a test analyst, tester or test manager performs a test case selection manually. Test selection is based on several decisions criteria's, which are experience based. Selection of the correct test subset is extremely difficult in the case of very large and long projects where thousands of low and high level tests exist.

There are researchers, who are trying to address this problem using profiling, which is looking at the amount of executed code by used test subset. The other ones (e.g. Dickinson – [6]) proposed distribution of the profiles with the profile space by using the cluster analysis on the profiles [7]. Other authors are using solutions based on Proportional Sampling Strategy [18], Optimally Refined Proportional Sampling Strategy [19], Follow-the-Crowd Strategies [20] or Partial Sums Condition [21].

In this paper we present how useful usage of visualization is, based on the "Test City" metaphor. We explain how to perform low-level test case selection, mining and test reorganization based on the very basic set of metrics available in the project.

For experimental work we have established a new system, presented in Figure 5, interacting with several Test Management applications placed on the market. The base idea of the system is an automation extraction and pre-evaluation of several different test metrics. Necessary data is extracted from the Test Management tool via available API connection and evaluated to get required set of metrics. Collected information is stored as a text file, e.g. CSV (Comma Separated Values), and afterwards imported into a visualization framework, wherein necessary analysis has been performed. The analysis result is going to the Test Management tool as an input for Test-Set creation and evaluation.

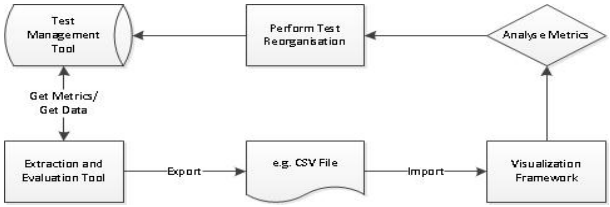

Fig.4. Information exchange in the Test Analyse and Reorganization System

Within our research for three test projects that contains over 4000 LLTC each, we have performed analysis for basic and extended test metrics. The projects have been running independently in three test projects with a large number of common requirements. This allows us to gain lots information that is valuable to prove our concept and create inputs for further work on possible visualization usage in test management domain.

In Figure 6 and 8 we present results of our visualization for the same test project but using different but very basic, test metrics.

1. Test execution status→ mapped to the colour.
2. Test execution age → mapped to the height.
3. Number of executions → mapped to size.

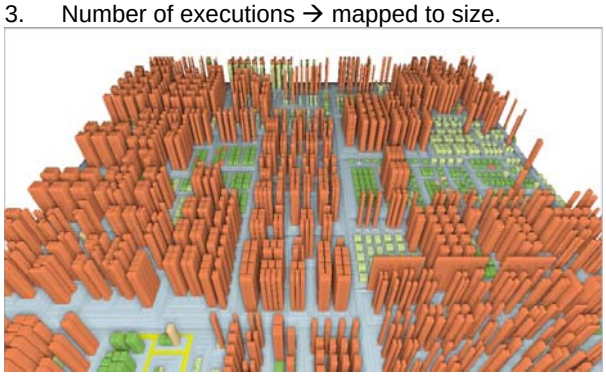

Fig.5. Test City based on LLTC for Test Project

The districts (as a square groups) of the Test City are mapped to the structure available within the Test

Management system (e.g. Test or Test object folders) to provide real reference to the analysed test base.

Looking at the possible analysis for visualization of the test base according to the Figure 6, we can provide the following input for improvements:

1. There are a large number of old LLTC, which has been not executed or there is no execution status available (orange – no status, yellow – open, green - success). Those LLTC shall be adapted, connected to the valid test suite or moved into archive.
2. In the test base is a large number of LLTC, which has been executed later than 365 days ago. Those tests are most likely obsolete and shall be moved to the archive.
3. The number of executions for certain LLTC is very hard to estimate; based just on the size of the buildings, therefore closure is needed. Zooming interesting area (Figure 8 – searched object is marked yellow) we can see that there are as well a large number of LLTC, which has never been executed, but still exists in the test base. Those tests are to be deleted or moved into an archive.

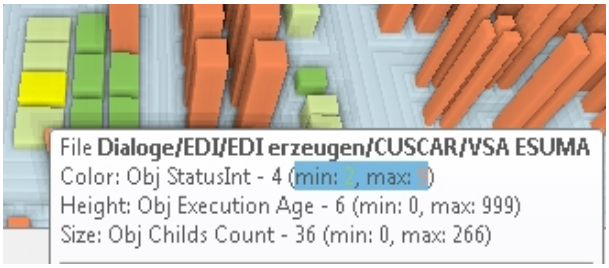

Fig.6. Zooming interesting area of Test City, looking for number of executions (Childs Count).

The other view (see Figure 8) for the same Test City is based on other test metrics constellation:

1. Test execution age → mapped to the colour
2. Description length → mapped to the height.
3. Modification age → mapped to size.

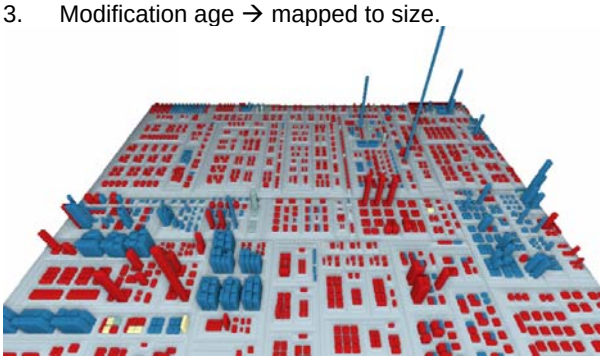

Fig.7. Test City 2 based on LLTC for Test Project

Before diving into details, the first impression we get by looking at the overview of the Test System (Figure 6 and 8) is that, even if the system looks well-organized, in spite of the numerous disharmonious artefacts: we see very dark districts, where the tests which were executed more than 365 days ago are localized and districts of increased number of high building, even skyscrapers, in which several very important and common tests are defined.

The skyscrapers are giving us the impression how many of existing LLTC are described much better than the other ones. After a short analysis based with a focus on the interesting building, we could find out that the biggest part of those buildings and districts are representing part of the regression test for the area, which has a very big importance to the project (see Figure 9 for zooming – building yellow marked).

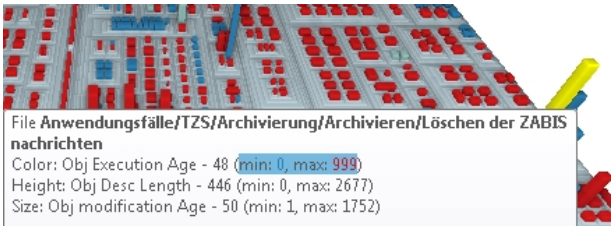

Fig.8. Zooming interesting area of Test City, looking for description length

The third attribute or mapped metric was in our case the modification age, which has not added in this case a value to the analysis. Information received from both views of the Test City has proven the information's from the first stage of our analysis.

The first look through the test bases gave us a very good impression about the quality and areas of the system undergoing the test, even without knowing the system itself. This was possible within a few minutes counted from metrics import to perform an analysis of the system. Necessary data for LLTC adaptation and/or reorganization can be taken/exported based on zooming information in the interesting areas/districts.

The thresholds for recognition of obsolete LLTCs, which are to be taken under consideration, are strictly connected to the application lifetime. As soon as the application update cycle is very short (below 1-2 months), we can assume that found data, which has not been created, modified or executed longer than 1 year ago, is obsolete. This does not have to be always true, given however a possibility to easily fetch unwanted anomalies within the test base. Setting threshold to the other values gave us the possibility to recognize specific patterns and exclude undesired data from the test suite. Other examples can be projected to the description length. We assume, a well-defined LLTC as soon as its description length contains at least 100 characters, first without checking the content. Everything below, we would treat as data to be investigated. On the other hand if there are too many characters we can assume, found LLTC is very complex, as its description needs to be very long.

Related work

Since the early days of software visualization, software has been visualized at various levels of detail, from the module granularity seen in Rigi [8] to the individual lines of code depicted in SeeSoft [9].

The increase in computing power over the last 2 decades enabled the use of 3D metric-based visualizations, which provides the means to explore more realistic metaphors for software representation. One such approach is poly cylinders [10], which makes use of the third dimension to map more metrics. As opposed to this approach in which the representations of the software artefacts can be manipulated (i.e., moved around), our code cities imply a clear sense of locality, which helps with viewer orientation. Moreover, our approach provides an overview of the hierarchical (i.e., test folder, test object, package) structure of the testware.

The value of a city metaphor for information visualization is proven by papers, which proposed the idea, even without having an implementation. [11] Proposed this idea for visualizing information for network monitoring and later [12] proposed a similar idea for software production. Among the researchers who actually implemented the city metaphor, [17; 18; 19] represented classes are districts and the methods are buildings. Apart from the loss of package information (i.e., the big picture), this approach does not

scale to the magnitude of today's software systems, because of its granularity.

The 3D visual approach closest in focus to ours is [13], which uses boxes to depict classes and maps software metrics on their height, colour and twist. The classes' box representations are laid out using either a modified tree map layout or a sunburst layout, which split the space according to the package structure of the system. The authors address the detection of design principles violations or anti-patterns by visually correlating outlying properties of the representations, e.g., a twisted and tall box represents a class for which the two mapped metrics have an extremely high value. Besides false positives and negatives, the drawbacks of this approach are that one needs different sets of metrics for each design anomaly and the number of metrics needed for the detection oftentimes exceeds the mapping limit of the representation (i.e., 3). The detection strategies [14] were introduced as a mechanism to formulate complex rules using the composition of metrics-based filters, and extended later [15] by formalizing the detection strategies and providing aid in recovering from detected problems.

Conclusions and future work

Test case management, test analysis and test creation are the most important tasks within the whole test management process. It is very hard to concentrate the analysis on a small set of the test, as it is not getting a potential win against the requirement spectrum. Performed visualization has shown us how easy in use and efficient it can be presented as a method for test analysis. Finding an obsolete LLTC based on available metrics and set thresholds is very comfortable and does not require deep system knowledge, even if system is very complex. To get a fast overview about a large number of test cases without deep knowledge of the test base is very important, especially if analysis is to be performed through external or new organization. This saves needed time and allows a very fast overview in "high management" capable way. This can act as a base for further and deeper analysis and test reorganization activities. Additionally we have observed that the person performing an analysis is tending to point its view on maximum two metrics in time and not searching for further information on the third one. This behaviour was driven via visualization framework and its available mapping attributes and partly human laziness.

Our future directions will focus on the points listed below:

1. Extension for more APIs to Test Management tools available on the market.
2. Comparison for analysis outcome when using same metrics but different Visualization Metaphors.
3. Visualization for metrics within the timeline.
4. Extend number of evaluated metrics, e.g. to get possibility to find out duplicates.

REFERENCES

[1] Richard Wettel, Michele Lanza, S.: Program Comprehension through Software Habitability. *In Proceedings of 15th International Conference on Program Comprehension*, (ICPC 2007). IEEE Computer Society. (2007)

[2] Dickinson, W. The Application of Cluster Filtering to operational testing of Software. *Doctoral dissertation*. Case Western Reserve University, (2001)

[3] David Zaen Leon Cesin, Profile analysis techniques for observation-based software testing, *Doctoral dissertation*, Case Western Reserve University, (2005)

[4] ISTQB, Syllabus, *http://istqb.org/download/attachments/2326555/Foundation+Level+Syllabus+%282010%29.pdf* (2010)

[5] IEEE, 1059-1993 - *IEEE Guide for Software Verification and Validation Plans,* http://standards.ieee.org/findstds/standard/1059-1993.htm (1993)

[6] Dickinson, W. The Application of Cluster Filtering to operational testing of Software. *Doctoral dissertation*. Case Western Reserve University (2001)

[7] David Zaen Leon Cesin, Profile analysis techniques for observation-based software testing, *Doctoral dissertation*, Case Western Reserve University (2005)

[8] Muller, H., and Klashinsky, S.:. Rigi: a system for programming-in-the-large. In Proceedings of ICSE 1988, 80–86, *ACM Press* (1988)

[9] Eick, S., Graves, T., Karr, A., Marron, J., and Mockus, S.: Does code decay? Assessing the evidence from change management data. *IEEE Transactions on Software Engineering 27*, 1, 1–12 (1998).

[10] Marcus, A., Feng, L., and Maletic, J. I. S.: 3d representations for software visualization. *In Proceedings of SoftVis 2003*, 27–36, ACM Press (2003).

[11] Santos, C. R. D., Gros, P., Abel, P., Loisel, D., Trichaud, N., and Paris, J. P. S.: Mapping information onto 3d virtual worlds. *In Proceedings of the IV International Conference on Information Visualization 2000*, 379–386, (2000).

[12] Panas, T., Berrigan, R., and Grundy, J. S.: A 3d metaphor for software production visuali-zation. IV 2003 - *International Conference on Computer Visualization and Graphics Applications*, 314, IEEE CS Press, (2003).

[13] Langelier, G., Sahraoui, H. A., and Poulin, P. S.: Visualization-based analysis of quality for large-scale software systems. In Proceedings of ASE 2005, 214–223, *ACM Press*, (2005)

[14] Marinescu, R. S.: Detection strategies: Metrics-based rules for detecting design flaws. In Proceedings of ICSM 2004, 350–359, *IEEE CS Press* (2004)

[15] Lanza, M., and Marinescu, R. S.:. Object-OrientedMetrics in Practice. *Springer* (2006)

[16] Richard Wettel, Software Systems as Cities, *Doctoral Dissertation*, Faculty of Informatics of the Università della Svizzera Italiana (2010)

[17] Ivar Jacobson, Grady Booch, James Rumbaugh, The Unified Software Development Process, *Addison-Wesley Professional*, (1999) ISBN 0-201-57169-2

[18] E. J. Weyuker and B. Jeng, Analyzing partition testing strategies, *IEEE Transactions on Software Engineering 17*, (1991), 703–711.

[19] F. T. Chan, T.Y. Chen, I.K. Mak, and Y. T. Yu, Proportional sampling strategy: guidelines for software testing practitioners, *Information and Software Technology*, (1996), 775–782

[20] T.Y. Chen and Y. T. Yu, On the expected number of failures detected by subdomain testing and random testing, *IEEE Transactions on Software Engineering*, (1996), 109–119

[21] F. T. Chan, T.Y. Chen, and T.H. Tse, On the effectiveness of test case allocation schemes in partition testing, *Information and Software Technology*, (1997), 719–726.

[22] V. Averbukh, M. Bakhterev, Interface and visualization metaphors, HCI'07 Proceedings of the 12th international conference on Human-computer interaction: interaction platforms and techniques, *Springer-Verlag*, (2007), ISBN: 978-3-540-73106-1.

***Author**: mgr inż. Artur Sosnówka, Zachodniopomorski Uniwersytet Technologiczny, Katedra Inżynierii Oprogramowania, ul.Żołnierska 49, 71-210 Szczecin, e-mail: arsosnowka@wi.zut.edu.pl*