Artur SOSNÓWKA

West Pomeranian University of Technology Szczecin

Testware reorganization with help of test city metaphor

Abstract. *Error detection within the testware means significant cost and time savings for execution and test maintenance. Both, the cost of compliance and non-compliance play an important role in the selection of the necessary test. With help of metrics coupled with testware objects and visualization, clear and easy manner may be presented to various artefacts of available test-data. This paper presents the results of research in test project in the field of industrial reorganization and optimization of the test database using "test city" visualization metaphor and basic test metrics.*

Streszczenie. *Wykrywanie błędów w bazie testowej może oznaczać znaczące oszczędności kosztów i koniecznego do ich przeprowadzenia czasu. Zarówno koszta zgodności jak i braku zgodności odgrywają ważną rolę w doborze koniecznych testów. Za pomocą sprzężonych z obiektami testware'u metryk i ich wizualizacji, w łatwy i czytelny sposób mogą zostać przedstawione różne artefakty dostępnych danych testowych. W artykule przedstawiono wyniki przeprowadzonych badań w projekcie testowym w przemyśle w dziedzinie reorganizacji i optymalizacji bazy testowej przy pomocy przenośni wizualizacyjej oraz podstawowych metryk testowych. (**Reorganizacja bazy testów za pomocą metafory „miasta testów”**).*

Keywords: Data Mining, Visualization metaphor, Testware, Test City, Test metrics, Test management, Test case visualization, Test selection.
Słowa kluczowe: Metafora wizualizacyjna, Miasto Testów, Metryki testowe, wizualizacja testów, selekcja testów.

doi:10.12915/pe.2014.02.27

Introduction

Testing, especially regression test-activity, during the whole Application Lifecycle, aims to find defects correlated to added, enhanced or adapted system or program functionality. Testing activity is very important during the maintenance, when the application or system is already long time in production. Software maintenance account in average two-third of the overall application costs [14] and the regression test is not a small part of the whole maintenance budget.

Regression testing (or program revalidation) is about to ensure that no new defects (called regression errors) have been introduced into previously validated code [12]. Although regression testing is usually associated with system testing after a code change, regression testing can be carried out at unit, integration, system or system integration testing levels.

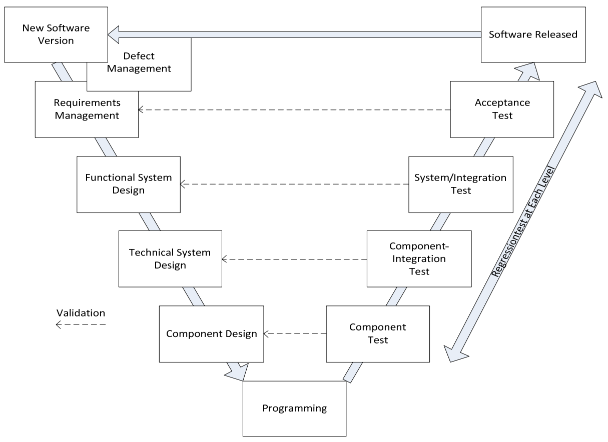


Fig.1. Activities during software maintenance and regression testing

The activities that take place during the maintenance phase can be easily showed as a sequence flow put on V-Modell [7]. Figure 1 show that after software is released, the failure reports and the change requests for the software are compiled in Requirements Management, and the software is modified to make necessary changes. Component, system, integration and acceptance tests are carried out for new functionality for modified parts of the code, while regression test cases are carried out to test the unchanged parts of the code that may be affected by the code change. Those tests are run for all phases and with different scope at each domain. After the testing is complete, the new version of the software is released, which then undergoes a similar cycle.

Regression testing is acknowledged to be an expensive activity. It consumes large amounts of time as well as effort, and often accounts for almost half of the software maintenance costs [12], [11]. The extents to which time and effort are being spent on regression testing are exemplified by a study [8] that reports that it took 1000 machine-hours to execute approximately 30,000 functional test cases for a software product. It is also important to note that hundreds of man-hours are spent by test engineers to oversee the regression testing process; that is to set up test runs, monitor test execution, analyse results, and maintain testing resources [8]. Minimization of regression test effort is, therefore, an issue of considerable practical importance, and has the potential to substantially reduce software maintenance costs.

Testware reorganization

In our previous papers we have present an idea for test cost reduction based on test reorganization with help of test-city visualisation metaphor. Use of an effective regression test selection technique within the existing testware can help to reduce the testing costs in environments in which a program undergoes frequent modifications.

Though substantial research results on regression test selection have been reported in the literature, several studies [9],[10] show that very few software industries deploy systematic test selection strategies or automation support during regression testing. The approaches that are most often used in the industry for identification of relevant regression test cases are either based on expert judgment, or based on some form of manual program analysis. However, selection of test cases based on expert judgment tends to become ineffective and unreliable for large software products. Even for moderately complex systems, it is usually extremely difficult to manually identify test cases that are relevant to a change. This approach often leads to a large number of test cases being selected and rerun even for small changes to the original program, leading to unnecessarily high regression testing costs.

What is probably more disconcerting is the fact that many test cases which could have potentially detected regression errors could be overlooked during manual selection. Another problem that surfaces during regression testing stems from the fact that testers (either from the same organization or from third-party companies) are usually supplied with only the functional description of the software, and therefore lack adequate knowledge about the code to precisely select only those test cases that are relevant to a modification [13].

Test selection performed on the large testware with thousands of the test cases is cost consuming and does not always assure non-existence for obsolete, re-testable and redundant test cases. Testware size reduction provides possibility to reduce and more effective usage for test-resources. In our approach we have per-formed testware analysis based on test city metaphor in one of the project in the industry and present the result within this paper.

Visualisation metaphor

A visualization metaphor is defined as a map establishing the correspondence be-tween 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 [3].

Lev Manovich has said: "an important innovation of computers is that they can transform any media into another". This gives us 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 information which is the most important just for him. Hence, any type of visualization has specific connotations, which may become metaphoric when seen in context of a specific data source. Metaphor in visualization works at the level of structure, it compares the composition of a dataset to a particular conceptual construct, and the choice of any visualization is always a matter of interpretation.

Numerous currently existing visualization systems are divided into three main classes:

- Scientific visualization systems [4];
- Information visualization systems [5];
- Software visualization systems [6].

Although all visualization systems differ in purposes and implementation details, they do have something common; they manipulate some visual model of the abstract data and are translating 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 basic and easy to understand "City metaphor" which is helpful for representation specific test data and allow easier test reorganization. After some of the previous research work, which is however not in focus of this paper, we settled our first attempt to the metaphor, which is very widely presented and is a part of Phd from Richard Wettel [2]. 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: Soft-ware is complex; there is no way around this.
- Classes are the cornerstone of the object-oriented paradigm, and together with the packages they reside in, the primary orientation point for developers [2].

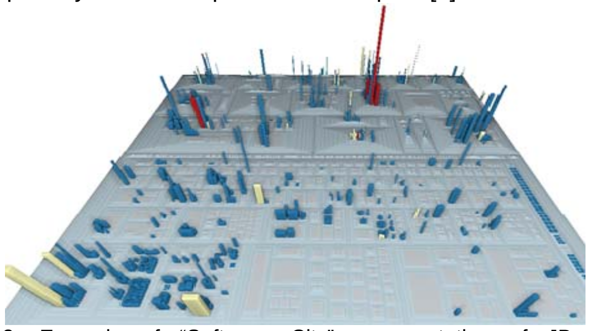


Fig.2. Example of "Software City" representation of JBoss application server

In our attempt we perform mapping between available LLTC **Błąd! Nie można odnaleźć źródła odwołania.** and its metrics to provide easy to understand and manage overview about the current state of testware.

Test metrics

To be able to perform data visualization, defined set of the static and dynamic data has to be prepared. Based on the necessary information's for LLTC we can extract following basic test metrics, which would be used later for mapping:

- Execution age,
- Execution status,
- Number of executions.

For our work we have established a new system 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. Those metric are imported via available API connections from the Test Management tool and evaluated to get required set of metrics. The test metrics are provided as a text file, e.g. CSV (Comma Separated Values), and imported into visualization frame-work. Extracted metrics is to be mapped into the chosen visualization metaphor as:

- Data physical properties (colour, geometry, height map-ping, abstract shapes)
- Data granularity (unit cubes, building border or urban block related).

Visualization framework allows us performing necessary representation for visualized information needed for Testware Reorganisation.

Testware reorganisation

In our research within test project we were able to perform testware reorganisation with help of visualisation metaphor. This allows us to gain information's which are valuable to prove our concept and create inputs for further work on possible visualization usage in test management domain.

In the next few figures we present results our work for test project with testware structure shown in the tables 1 and 2. Visualization parameters have been based on following basic, test metrics:

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

To provide real reference to the analysed testware, the districts (as square groups) of the Test City are mapped to the structure created by test teams and managed with help of the Test Management system (e.g. Test folder or Test object).

The Figures are showing testware characteristics for LLTC last executions as follows:

Green → execution not older than 370 days
Red → execution older than 370 days

Table 1. Testware artefacts in numbers.

Object Type	Quantity	Number of Executions	% LLTC	% Exec
LLTC	18293	37899	100,00%	100,00%
LLTC: exec > 370 days ago	11769	507	64,34%	1,34%
LLTC: exec < 370 days ago	6524	37392	35,66%	98,66%

Table 1 show the visualized artefacts in numbers which has been as well presented in the Figure 3, 4 and 5.

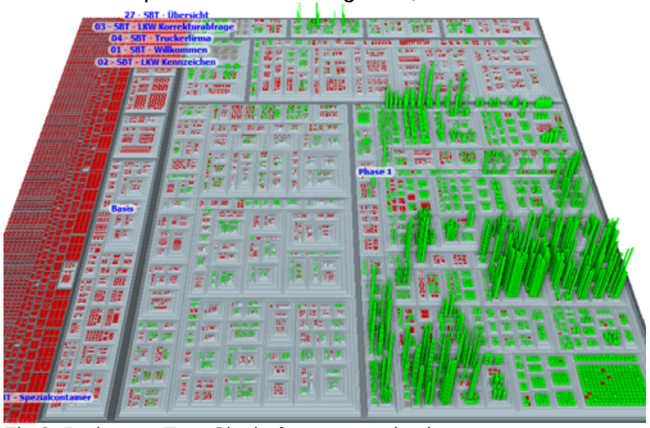


Fig.3. Project as Test City before reorganization

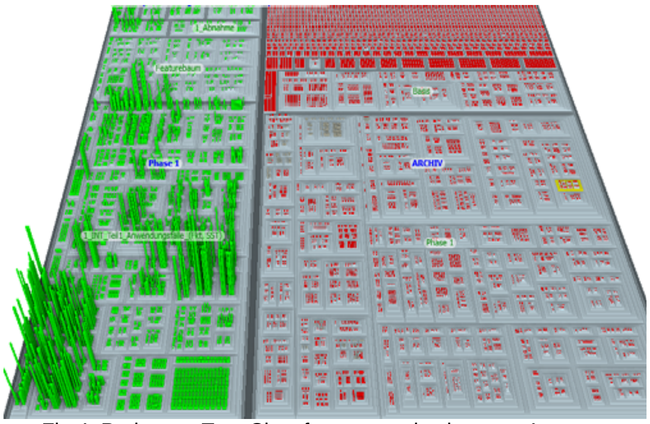
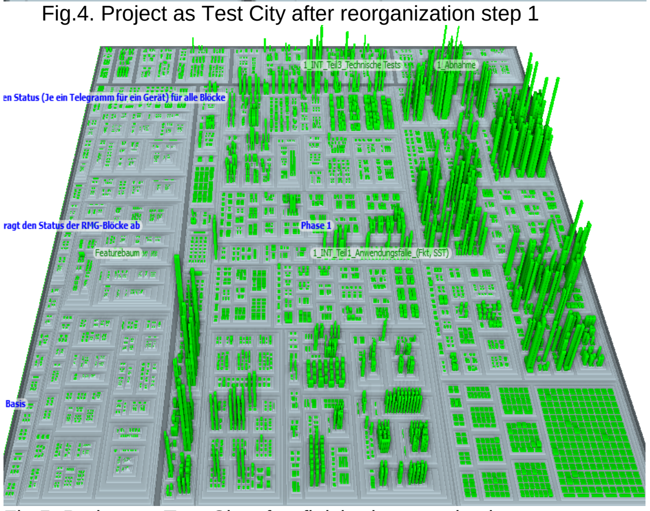


Fig.4. Project as Test City after reorganization step 1


1_INT_Teil3_Technische Tests
1_Abnahme
en Status (Je ein Telegramm für ein Gerät) für alle Blöcke
ragt den Status der RMG-Blöcke ab
Phase 1
Featurebaum
1_INT_Teil1_Anwendungsfälle_(Fkt, SST)
Basis
Fig.5. Project as Test City after finished reorganization

On presented Figures we can recognize how much reorganization has changed the structure and testware size.

During the reorganization more than 11000 LLTCs has been moved from its initial location to the archive and after testware Backup, completely removed from the database. 64% of not anymore used artefacts have been taken out from the project. This has allowed Test Manager to get very clean overview about the database and minimize testware maintenance costs for ~5%.

Testware reorganization has been performed in parallel with two approaches:

- Manually, through the tester and,
- Automatically, with help of created application

In both cases we have used input from visualization framework exported as list with LLTC unique IDs and allocated path within the database.

Table 2. Cost comparison

	Manuall execution	Automated execution
Time max (sec)	120,00	15,00
Time min (sec)	15,00	2,00
Average (sec)	67,50	8,50
Overall (sec)	794407,50	100036,50
Overall (h)	220,67	27,79
Overall (working days)	27,58	3,47

Comparison has showed that automated reorganization has brought cost savings against the manual approach equal to 24 working days in case of effort and 26 days in case of duration. The difference in savings in case of duration and effort is resulting from 24h working day for automation and 8h working day for a human.

Conclusions and future work

In this paper we have presented use of method with help of visualisation metaphor and basic test metrics combined to perform testware reorganization with small resource effort. Providing a visual mapping of correspondence data can be very useful method for analysis within big and long life projects. Achieved long term testware maintenance saving of 5% was additionally followed with eight times faster execution of the reorganization activities in comparison to traditional, manual approach. Results have proved usefulness for presented method in test domain.

Looking at the reorganization results presented at Figure 5, we can assume that additional improvements could be potentially done. This work can be a part of future work .The selection of other than presented metrics and its relevance for testware reorganization would be very useful and can be included.

REFERENCES

[1] ISTQB, Syllabus, 2010 - http://istqb.org/download/attachments/2326555/Foundation+Level+Syllabus+%282010%29.pdf

[2] Wettel, R., 2010, *Software Systems as Cities*, Doctoral Dissertation, Faculty of Informatics of the Università della Svizzera Italiana

[3] Buffaker, B., Hyun, Z., Luckie, M., 2010, *IPv4 and IPv6 AS Core: Visualizing IPv4 and IPv6 Internet Topology at a Macroscopic Scale in 2010*, http://www.caida.org/research/topology/as_core_network/

[4] Friendly, M., 2008, *Milestones in the history of thematic cartography, statistical graphics, and data visualization*, http://www.math.yorku.ca/SCS/Gallery/milestone/milestone.pdf

[5] González, V., Kobsa, A., 2003, *Benefits of Information Visualization Systems for Administrative Data Analysts*, Proceedings. Seventh International Conference, 331-336, Information Visualization, IV 2003

[6] Stasko, J.T., Patterson, C., 1992, *Understanding and characterizing software visualization systems*, Proceedings., 1992 IEEE Workshop, 3 – 10.

[7] Barry W. Boehm: *Software Engineering Economics*, Englewood Cliffs, NJ, Prentice-Hall, 1981. ISBN 0-13-822122-7

[8] H. Do, S. Mirarab, L. Tahvildari, and G. Rothermel.*The effects of time constraints on test case prioritization: A series of controlled experiments*. IEEE Transactions on Software Engineering, 36(5):593–617, September 2010.

[9] M. Grindal, J. Offutt, and J. Mellin. *On the testing maturity of software producing organizations*. In TAIC-PART ’06: Proceedings of the Testing: Academic & Industrial Conference on Practice And Research Techniques, pages 171–180, 2006.
[10] J. Guan, J. Offutt, and P. Ammann. *An industrial case study of structural testing applied to safety critical embedded software*. In Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering, pages 272–277, 2006.
[11] G. Kapfhammer. *The Computer Science Handbook*, chapter on Software testing. CRC Press, Boca Raton, FL, 2nd edition, 2004.
[12] H. Leung and L. White. *Insights into regression testing*.In Proceedings of the Conference on Software Maintenance, pages 60–69, 1989.
[13] A. Pasala, Y Fung, F. Akladios, A. Raju, and R. Gorthi. *Selection of regression test suite to validate software applications upon deployment of upgrades*. In 19th Australian Conference on Software Engineering, pages 130–138, March 2008.
[14] R. Pressman. *Software Engineering: A Practitioner’s Approach*. McGraw-Hill, New York, 2002.

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