Baltic J. Modern Computing, Vol. 4 (2016), No. 1, pp. 34–50

Visualization of Large-Scale Application Testing Results

Rūdolfs OPMANIS, Paulis ĶIKUSTS, Mārtiņš OPMANIS

Institute of Mathematics and Computer Science, University of Latvia
Rainis blvd. 29, Riga, LV1459, Latvia
rudolfs.opmanis@gmail.com paulis.kikusts@lumii.lv
martins.opmanis@lumii.lv

Abstract. In this paper, we propose an interactive visualization technique for analysis of results of large-scale software regression testing. Results of this analysis identify the most probable root causes of deterioration or improvement. The provided multi-level visualization solution allows to evaluate project status and efficiently manage project lifecycle. This technique can be used on top of almost any testing framework with little or no adjustment.
Five expressive visual constructions called views are designed and contain a consolidated testing overview or part of detailed testing information. We defined the interactive behavior and transitions between views for several user groups, such as managers, quality assurance personnel, and developers.
Our approach is most efficient for processing of large-scale testing results with deep hierarchical structures of test groups and environment attributes.

Keywords: Regression testing, large-scale hierarchical data visualization, test result visualization.

1 Introduction

Visualization of software testing covers a broad area from designing of test cases, planning of test runs, management of test performing, till analysis of results of completed testing. In literature, we can find many approaches, for example description of visualization concepts (Wang et.al., 2012, Bolton, 2015), specialized tools (SmartBear, 2015, Chen et al., 2008, Jones et al., 2002), and universal tools (Jenkins, 2011, Mundigl, 2009, TeamCity, 2006).

In this paper, we propose an interactive visualization technique for analysis of results of regression testing of software. This technique can be used on top of almost any testing framework with little or no adjustment.

In our previous paper (Opmanis et al., 2015) we investigated a way how to compare large-scale hierarchical testing results of two different builds (revisions) of the same software and identify the most probable root causes of deterioration. The analysis was carried out for the real project with 1682 tests grouped into 145 basic test groups that were executed in 6327 test runs on 57 builds in 18 environments.

In the current paper, we describe ways how testing results can be visualized involving our analysis. "Visualization . . . helps us cope with information overload, saves us time, and reaches us on an innate level. Well-crafted visualizations are compelling and beautiful to look and to intellectually enjoy." (Steele, 2010, p. 365).

An adequate information visualization solution must be expressive and sufficiently simple, at the same time not misleading the reader or creating unwilling associations, and must allow access to any important piece of information. Taking this into account and following the recommendations given in (Steele, 2010, Tufte, 2009) we provide an interactive visual way to evaluate project status, allow efficiently manage project lifecycle and optimize resource allocation across various projects. Expressive visual constructions called views are designed and contain a consolidated testing overview or part of detailed testing information. Proposed views are explained here as advanced mockups. We already implemented some of them.

In (Chen and Ince, 2007) authors investigated a related problem of visualization of regression testing results. The data model is similar to ours and uses a hierarchy of test groups and several testing environments named platforms. Testing results are visualized using tables with color and letter coded statuses of test executions. We are approaching different solution when just testing results are known without direct access to the source code of tested software. Our multi-level visualization solution uses advanced root cause analysis (Opmanis et al., 2015) also based on a hierarchy of attributes of testing environments and a larger set of possible test execution statuses.

The approach described in this paper may be shortly characterized as "analyze first, visualize after". It differs from the visual analytics approach where visualization is the first step, and human performed analysis is done using created visualizations, e.g., heat maps, that are essentially larger graphical constructions if compared with ours. Applying such a technique in regression testing is described in (Engstrom et al., 2014).

Our analysis and visualization solutions are aimed at results of large-scale testing based on deep hierarchical structures of test groups and environment attributes. With such hierarchies, our approach is most efficient, and it would be worth to integrate these solutions into automated tools. One of the viable candidates is the widely used continuous integration server Jenkins (Jenkins, 2011) so extending their current visualization capabilities of software development process. Figure 1 shows an example visualization from Jenkins providing an unfiltered historical build statistics of testing results.

This paper is organized as follows: Section 2 describes data model of the testing result analysis, Section 3 briefly summarizes requirements that we had for the testing result visualization. In Section 4 we discuss visualization approaches that we used and how it helps various types of users. Conclusions and directions of possible future work are described in Section 5.

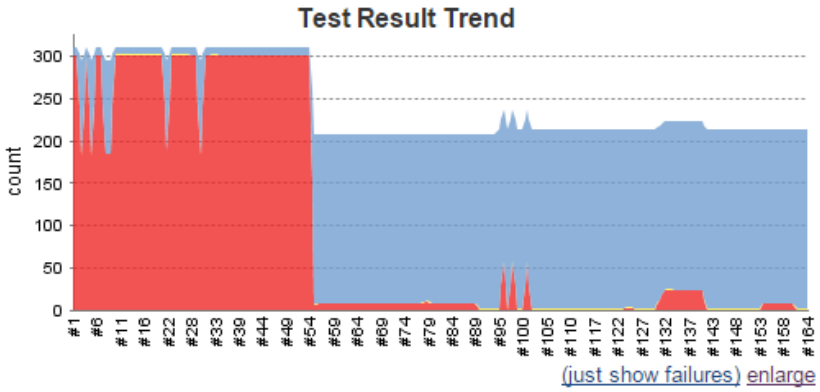

Fig. 1. Typical test result visualization solution provided by Jenkins.

2 Analysis of testing results

Data analysis is a necessary stage that must be performed before their visualization. According to (Tulp, 2012, Fry, 2004) the analysis phase comprise parsing, filtering, mining, and statistics. All these steps are carried out in our analysis of testing results and described in the paper (Opmanis et al., 2015).

In the following subsections we briefly explain the data model and the result of analysis to be visualized.

2.1 Data model of analysis

To analyze testing results we are using a data model which consists of:

- hierarchically grouped tests,
- attributed test execution environments,
- software builds being tested,
- results of executed tests aggregated in the same hierarchy as tests.

A result of each test comprises a status how the test ended. Optional information may contain stack trace in case the test execution crashed, or screenshot in case it returned the wrong result or running time if test finished correctly.

In Figure 2 we provide a simple example of a test group hierarchy. There are two separate groups of tests checking data import and login procedure. Each group is divided into more specific subgroups testing particular feature, e.g. data import from various XML sources. For further we assume that each lower level test group contains exactly one test with the same name.

Environments are described using attributes. A set of attribute values characterizes every particular environment. Also, these sets together with their subsets are organized in a hierarchy. As a simple example, we use four environments characterized by two attributes: operating system and architecture. Corresponding attribute value sets are

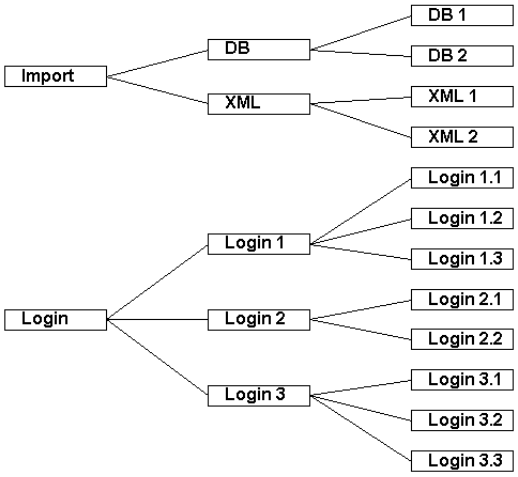

Fig. 2. Example testgroup hierarchy.

{Linux, 16bit}, {Linux, 32bit}, {Windows, 32bit}, and {Windows, 64bit}. The hierarchy also contains subsets {Linux}, {16bit}, {32bit}, {64bit}, and {Windows}. Elements in the hierarchy are connected by the parent-child relation "be a subset of" and this hierarchy is shown in Figure 3.

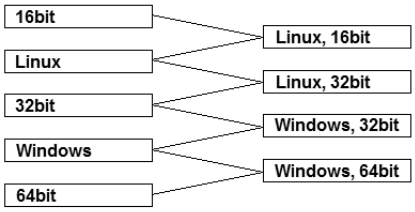

Fig. 3. Example hierarchy of attribute value subsets.

Examples below corresponds to the described example hierarchies.

In our previous paper (Opmanis et al., 2015) we described data model in depth. Now we extend the proposed approach of analysis also for improvements that can be easily done just by replacing the corresponding comparator.

2.2 Matrices of analysis results

After execution of our algorithm of analysis of testing results we build two matrices: deterioration matrix and improvement matrix. Rows and columns of each matrix correspond to the most significant objects in the hierarchy of attribute value subsets and test groups correspondingly. These rows and columns are named after corresponding hier-

archy elements. According to procedures described in (Opmanis et al., 2015) in each matrix attribute value subsets named in the rows do not intersect. Also, test groups named in the columns have no common predecessors in the hierarchy of test groups.

We say that object of the hierarchy is represented by some row (column) if this object is a descendant of the object corresponding to this row (column). If there are attribute value subsets not represented by rows, the additional row "*Other subsets*" is added. Similarly, if there are test groups not represented by columns, the additional column "*Other groups*" is added to the matrix. So each element of the both hierarchies becomes represented by unique row (column) of the matrix.

The value of the matrix element corresponding to the attribute value subset a, and the test group g is the number of deteriorations (or improvements) where environment attribute value set has a as a subset and test group has g as a predecessor. By calculating values, each deterioration (or improvement) adds 1 to exactly one matrix element.

Any of these matrices may be empty.

Example data shown in Tables 1 and 2 illustrates the common situation when there are deteriorations and improvements at the same time. The matrices are obtained by two independent processes, i.e. a comparator for improvements is not simply a logical complement of a comparator for deteriorations. As a result, the sets of the named rows (columns) in the both matrices are completely different.

Table 1. Example deterioration matrix.

Attribute value subsets	*Testgroups*		
	Import	Login 1	*Other groups*
{Windows, 64bit}	3	3	1
{Linux}	6	6	0
Other subsets	2	0	0

Table 2. Example improvement matrix.

Attribute value subsets	*Testgroups*		
	XML 1	Login	*Other groups*
{32bit}	1	4	1
Other subsets	0	2	0

Natural idea is to merge both matrices to show deteriorations and improvements together in a visually compact way. However, there is no obvious way how to do this when (as in our example) the sets of the named rows (columns) in the matrices are different. So we consider a few possible options.

The simplest way to merge matrices is creating a new one by listing names of rows and columns from the original two and copying corresponding deterioration and improvement values with "$-$" or "$+$" sign. If the initial matrices contain rows (columns) representing the same set of hierarchy elements, values in these rows (columns) are merged without duplicating. In our example the row *Other subsets* represents {Windows, 32bit} in the first matrix, and {Windows, 64bit} and {Linux, 16bit} in the other.

Therefore, both rows are presented in the merged matrix. Note, that there may be two numbers in one merged matrix cell: one for deterioration and one for improvement. Merged matrix of our example is shown in Table 3.

Table 3. Simply merged example matrices.

Attribute value subsets	*Testgroups*					
	Import	Login 1	*Other groups*	XML 1	Login	*Other groups*
{Windows, 64bit}	-3	-3	-1			
{Linux}	-6	-6	0			
Other subsets	-2	0	0			
{32bit}				$+1$	$+4$	$+1$
Other subsets				0	$+2$	0

Unfortunately, the matrix obtained in such a way has several drawbacks:

- Two equally titled rows "*Other subsets*" represent different sets of attribute value subsets,
- Two equally titled columns "*Other groups*" represent different sets of test groups,
- Test groups corresponding to columns have common predecessor. For example, test group "Import" is predecessor of itself and of "XML 1", but information about this subhierarchy is distributed among several columns that contradict the idea of compactness,
- Matrix elements are partitioned in blocks and there are large empty regions.

Another way how to combine original matrices and to avoid listed drawbacks is starting by more sophisticated merging the rows (columns) ensuring mutual nonintersecting and keeping as much as possible of the original ones. Values of deteriorations and improvements are recalculated in the same way as in original matrixes and both filled in the matrix separated by "/". For our example, we can get the matrix shown in Table 4. The names of the first two rows are the same as in the original matrices. The names of the columns "XML 1" and "Login 1" comes from the original matrices. Notation "$a - b$" in the column name denotes all a objects excluding that of b. So the first two columns together corresponds to the original column "Import" and the last two together to the original column "Login". It must be noted, that there may be several different resulting matrices. For example, instead of the two columns "Import $-$ XML 1" and "Login $-$ Login 1" we could create one column "*Other groups*" reducing a number of columns. However, correspondence of two neighbor columns to the initial column is lost. Despite the fact that this approach has no previous drawbacks, the main shortcoming is that main deteriorations and improvements are scattered in the merged matrix, and this again contradicts the idea of compactness.

To avoid drawbacks mentioned above we decided to use original deterioration matrix as is and recalculate only the number of improvements and add via "/" as the second values in the same matrix. That, in fact, means that deteriorations are considered to be more important than improvements. For the example matrices, the result is shown in Table 5.

Table 4. Merging based on merging rows and columns of example matrices.

Attribute value subsets	*Testgroups*			
	XML 1	Import − XML 1	Login 1	Login − Login 1
{Windows, 64bit}	$1/0$	$2/1$	$3/0$	$1/0$
{32bit}	$1/1$	$4/3$	$3/1$	$0/1$
{Linux, 16bit}	$1/0$	$2/0$	$3/0$	$0/1$

Table 5. Merging if deterioration is considered to be more important.

Attribute value subsets	*Testgroups*		
	Import	Login 1	*Other groups*
{Windows, 64bit}	$3/0$	$3/0$	$1/1$
{Linux}	$6/1$	$6/0$	$0/3$
Other subsets	$2/1$	$0/1$	$0/1$

Note that several matrix elements contain pairs of nonzero values of deteriorations and improvements what is the direct consequence of the independence of corresponding comparators.

This matrix is the input for our visualization described further and regardless a used testing system data must be prepared in this format.

3 Requirements for visualization of analysis data

We want to provide adequate visual overview of project status for the several user groups, such as:

- Managers
- Quality Assurance (QA) personnel
- Developers

This intent demands to specify requirements for visualization solution. In addition to general visualization principles reviewed in Section 1, there are also known suggestions focused on software (Kienle et al., 2007, Tulp, 2012).

In correspondence with (Kienle et al., 2007) we adapt such quality attributes as information scalability, customizability, interactivity, and usability and such functional requirements as views and filters.

Concerning questions stated in (Tulp, 2012) answers are: we deal with a hierarchical dataset in the specific format, data do not need to be scraped, and dataset complexity is low. The analysis phase (Opmanis et al., 2015) predecessing the visualization ensures that there are no outliers, the structure of the dataset is directly usable for the visualization, there is no need for additional data transformations and cleaning.

From the "process of understanding data" (Fry, 2004) our visualization deal with "represent" and "interact", where previous steps are passed during the analysis phase.

We specify the main needs of each user group:

- Managers have to be able to glance quickly over the project status.
- QA personnel have to be able to verify that testing is happening as expected.
- Developers have to be able to look at the results of the analysis and examine testing results till atomic level.

The main functional requirement: appropriate testing result visualizations and interactive scenarios for the each user group must be provided.

Quality requirements:

- Use simple enough means of visualization like concise charts and tables.
- Ensure context based and user dependent ergonomic interactivity.
- If amount of visual elements is too large to simultaneously show them all, show just part of them together with navigation fields giving access to the hidden parts.

In the next Section, we describe how these requirements are fulfilled using common set of graphical views and transitions among them.

4 Visualization of testing results

Different data visualization techniques and principles are described and some of them may fulfill our requirements. Treemaps (Itoh and Koyamada, 2003, Itoh et al., 2004) and sophisticated hierarchical data structures (Yang et al., 2003) are used for large scale hierarchical data visualization. In (WEB (a), Mundigl, 2009) the authors demonstrate how to present testing results in a compact way and describe configurable dashboards and reports.

Following "Today's data visualization tools go beyond the standard charts and graphs .... The images may include interactive capabilities, enabling users to manipulate them or drill into the data for querying and analysis." (WEB (b)) and taken into account suggestions from sources examined above we elaborate our view types and interactive transitions between them.

4.1 View types

All views are used to visualize and reveal interconnections between data that otherwise would be hidden or harder to find. In cases when a view shows something unexpected, that is not consistent with expectations of the user it is important to allow to explore the suspicious area in more details.

Our solution offers an interactive dashboard with multiple views. Each view contains a chart or a table and can be configured to filter required data.

Views are designed using widely used visualization techniques and chart types to simplify implementation and to allow to use available general purpose charting libraries with no or small customizations.

We introduce the following view types:

- Matrix chart of testing result analysis
- Timeline chart

– Table of testing results
– Test execution details
– State of testing

Each view is described below in details.

4.1.1 Matrix chart of testing result analysis. This chart visualizes the matrix defined as the input of visualization at the end of Section 2.2. The matrix contains pairs of numbers of deteriorations and improvements of two testing sessions for most significant test groups and environment attribute subsets.

For this chart, we offer three kinds: two simple charts showing a number of deteriorations and improvements separately, and one more sophisticated combined chart. Our proposal for the combined chart is to show for each matrix element only number of deteriorations (if there are such) and to show improvements only if there are no deteriorations. This proposal follows the decision made above that deteriorations are considered more important than improvements. We consider combined chart as the basic one with the possibility to easily switch among the named three.

These charts are designed to show the matrix in a visually expressive way if compared with a textual output of matrix content. Deteriorations and improvements are expressed as circles of different colors where the number inside a circle is the number of the corresponding deteriorations (improvements) while a radius of a circle grows with the growth of this number. The combined matrix chart corresponding to Table 5 is given in Figure 4, assuming that the name of the project under investigation is “ABC” and two builds compared are named as “Build 6.4.0.09” and “Build 6.4.0.12”.

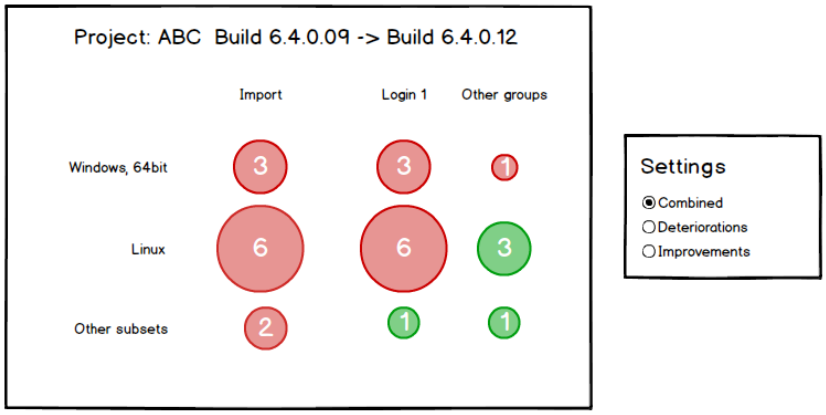

Fig. 4. Matrix chart of testing result analysis.

As it can be read from the chart, worse results for several test groups are observed for all environments. At the same time, there is an essential improvement for {Linux} attributed environments corresponding to test groups that are not distinguished. This situation may be caused by successful fixing some bugs shown up in the {Linux} attributed environments at the same time causing serious problems in other parts of the

software and influencing also program execution in other, i.e. {Windows, 64bit}, environments. Similarly, we can see that several tests started to fail on {Linux} attributed environments while there is a minor improvement in the others.

4.1.2 Timeline chart. This bar chart shows how testing results of the project evolved over time for all or particular test group, testing environment, and test execution status. Ordinary test execution statuses are "passed", "failed", "not completed", "predictably incorrect", "runtime error NNN", etc. Each bar corresponds to the testing session of a build, and its height corresponds to the number of tests with chosen status. This chart can also remind about events that happened at some moment outside the testing process and might influence the testing results, e.g. "system update" or "blackout". These events may be helpful to deduce external root cause of a problem.

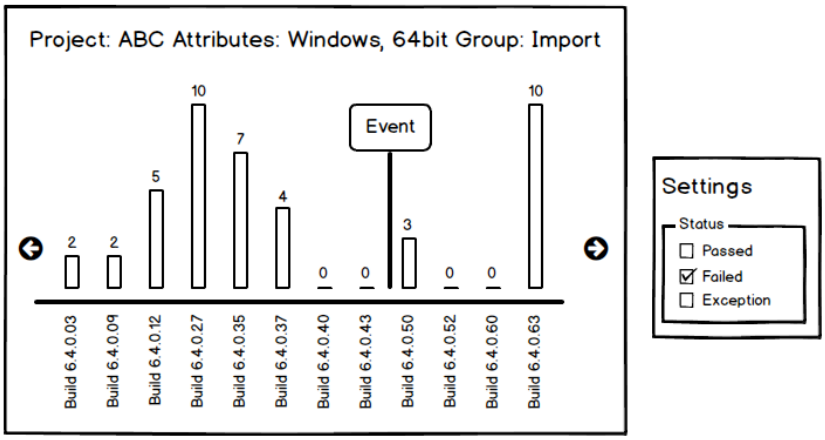

Fig. 5. Timeline chart.

Figure 5 demonstrates an example timeline chart for all data import tests under project "ABC", all testing environments that have Windows operation system and 64bit architecture. Bars in the chart shows a number of tests with status "failed". We can see that for "Build 6.4.0.03" there were only two failed tests, and later in "Build 6.4.0.27" 10 tests failed. Between "Build 6.4.0.27" and "Build 6.4.0.40" developers gradually fixed all bugs, and there were no failed tests. Between "Build 6.4.0.43" and "Build 6.4.0.50" some external event happened, and that might cause some tests to fail.

4.1.3 Table of testing results. This color-coded table demonstrates results of a specified testing session of tests of particular test group executed in a particular testing environment. Different colors encode test execution statuses. Figure 6 showed an example of a result table for all tests from data import test group when they were executed on "Build 6.4.0.50". Each row describes the result of execution of one test and specifies test, hardware, running time (if completed) and execution status. In general, only columns "Test" and "Status" are mandatory while the others are optional and configurable.

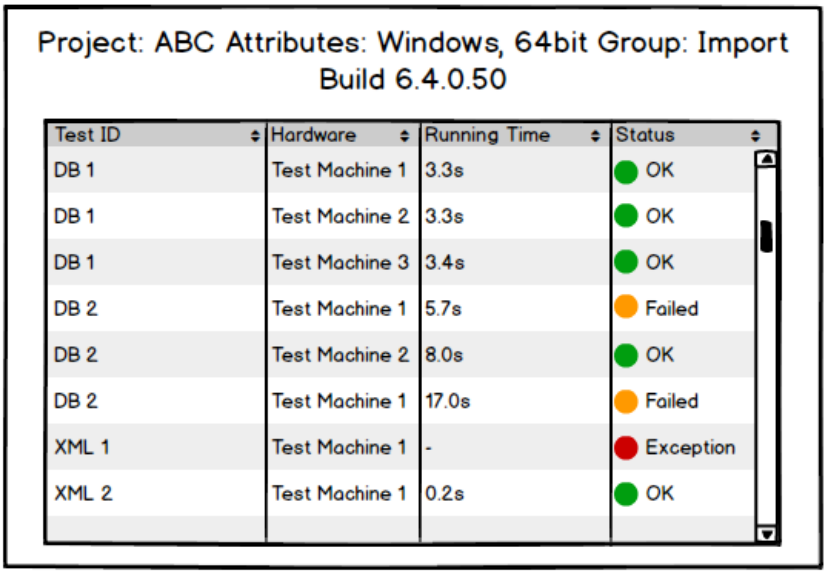

Fig. 6. Table of testing results.

4.1.4 Test execution details. This view shows all information about the execution of a particular test from the specified testing session and contains few mandatory fields and an arbitrary number of optional ones. The mandatory fields contain information about the test executed, the environment where testing took place, and the status of the test execution. Optional fields might contain information about hardware, running time, testing logs, screenshots of the application execution, and others.

Figure 7 shows an example view with execution details for the test “DB 2” which was run on “Test Machine 1” in the environment with attributes {Windows, Windows 7, 64bit, Firefox} and test execution ended with the incorrect result. Screenshot, a reference to log, running time and start time of testing are also shown.

4.1.5 State of testing. This pie chart gives primary insight in the overall status of testing. It is used to visualize aggregated test results of a certain testing session with the possibility to specify also test group or subset of environment attributes. Number of slices is low: just two or three allowing to focus on main characteristics. The size of each pie slice shows how many tests ended with the corresponding status: passed, failed or (optionaly) inconclusive.

The value in the center is cumulative characteristic and is calculated from the available values where the exact way of calculating is configurable. Figure 8 shows aggregated test results for all tests from the testing session of the example project “ABC” when “Build 6.4.0.16” was tested. Center’s 57% denotes the percentage of all tests ended with status “Passed”.

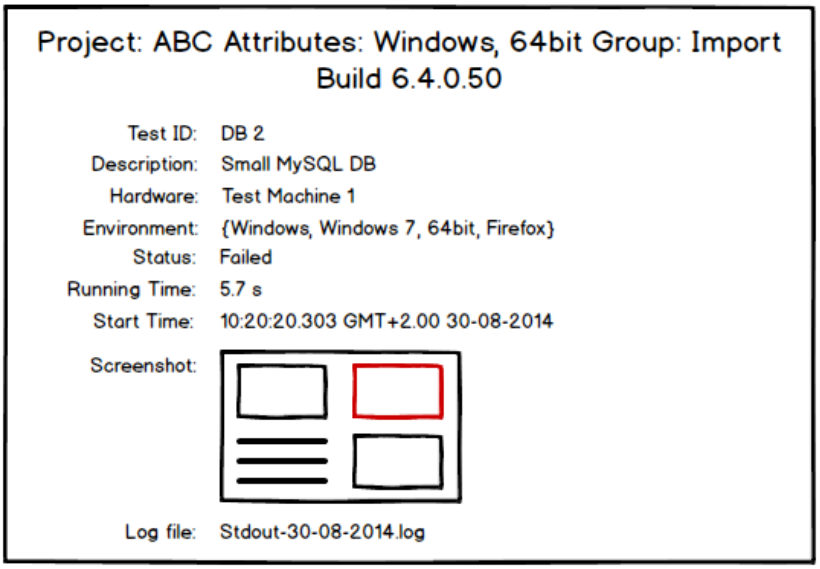

Fig. 7. Test execution details.

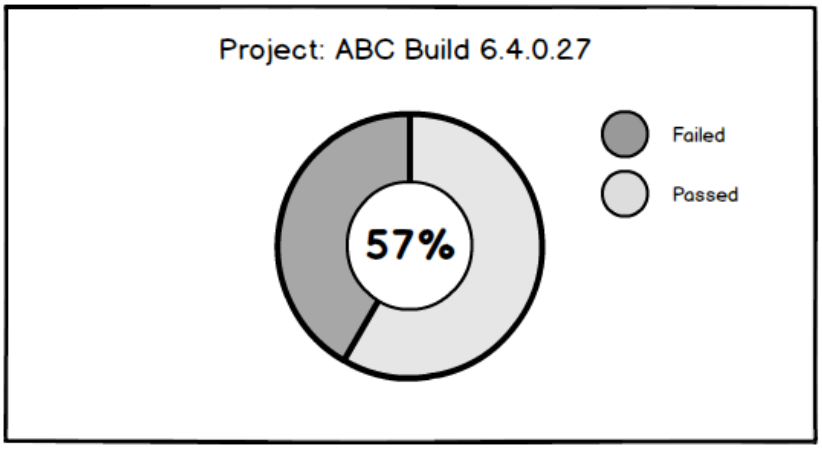

Fig. 8. State of testing.

4.2 Transitions between views and interactive behavior

To allow to reach as many details as necessary, we offer a set of transitions between previously proposed view types. A click on an appropriate visual element causes a particular transition. Our goal is to ensure that the user can always move to a more detailed view till test execution details are reached. All transitions in the direction from less detailed views to more detailed are summarized in Figure 9 where view types are represented by rectangles while arrows denote transitions between them. Labels on the arrows identify view elements that cause corresponding transitions. Human figures depict different user roles. Arrows from these figures to rectangles denotes entry points for different user groups while labels on arrows describe parameters that must be specified at the start of the system.

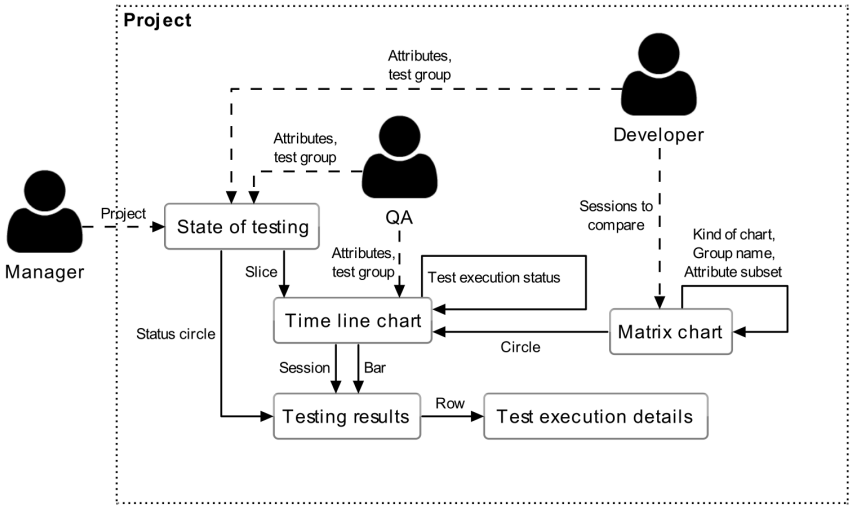

Fig. 9. Interactive transitions between views.

The latest testing session of the project is set by default. Each transition uses parameter values from the previous view.

In the views, there are the following types of interactive elements.

– Matrix chart of testing result analysis
  - Title of the column (the name of a test group)
  - Title of the row (the name of an attribute subset)
  - Circle (the element of the matrix)
  - Radio button (the kind of chart)

  Comment on interactions: The clicks on the elements of first two types causes transition that modifies matrix to substitute the clicked-on object with its direct descendants hiding other rows (columns). Elements of the modified matrix are recalculated according to the updated set of rows (columns). Example transition when

clicked on the label “Linux” of the matrix chart at Figure 4 results in the modified matrix chart shown in the Figure 10.

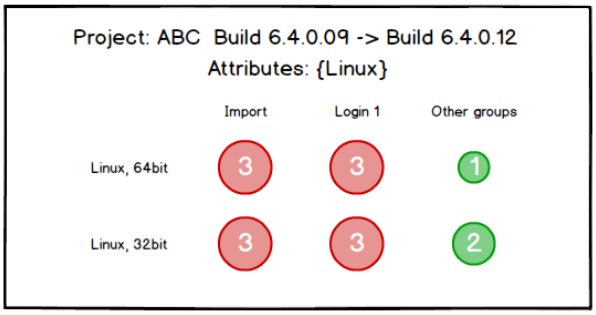

Fig. 10. Modified chart from Figure 4 when clicked on the label “Linux”.

- Timeline chart
  - Label (the name of a testing session)
  - Bar (the number of tests with chosen status)
  - Marker (the event mark)
  - Checkbox (the chosen execution status)

  Comment on interactions: There are two transitions to the Table of testing results. By clicking on a label all testing results from the corresponding session are shown. By clicking on a bar just testing results with a particular status are shown. If several checkboxes are checked, each bar is stacked representing the corresponding number of tests. Clicking on the event mark causes showing a more detailed description of the event (if available).
- Table of testing results
  - Row (the summary of a test execution)
- Test execution details – *none*
- State of testing
  - Slice (the number of tests with particular execution status)
  - Circle (the label of a particular testing status)

5 Conclusions

In this paper, we present a new interactive visualization solution that gives insight into the current status of regression testing of a particular software project. Designed interactive behavior ensures reaching the most detailed view of an execution of a particular test by few transitions. To observe the strength of the proposed approach for large-scale testing data, deep hierarchies of test groups and attribute value sets should be used.

Five view types having various levels of detail are designed to offer means of interaction for such user groups as managers, quality assurance personnel, and developers. A screenshot of a dashboard containing implementations of some of the proposed views is shown in Figure 11. The dashboard comprises five states of testing views and three timeline charts with various levels of detail.

The proposed solution achieves the initial goal and gives insight into quality dynamics of the sequence of builds and allows to find the main deterioration (improvement) points by few clicks.

Fig. 11. Screenshot of the dashboard.

Interesting upcoming tasks are:

- implementing the solution as plugins for other testing systems,
- using analysis data from outside,
- managing the dynamics of modifications of the test group hierarchy between testing sessions.

Moreover, in the future some additional problems may be investigated:

- assigning a formal semantics and more sophisticated behavior to events,
- extending the set of allowed transitions between existing types of views,
- designing of new types of views.

6 Acknowledgements

This work was supported by ERDF project 2014/0013/2DP/2.1.1.1.0/14/APIA/VIAA/034.

References

Bolton M. (2015). DevelopSense: Oracles from the Inside Out, Part 8: Successful Stumbling. http://www.developsense.com/blog/ (Accessed December 23, 2015)

Chen., R., Ince, T. (2007). Visualizing Regression Test Results. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.366.3623&rep=rep1&type=pdf (Accessed August 26, 2015)

Chen, J.C., Krishnamoorthy, S., Nguyen, B.N. (2008). ViViz - A Visualization Tool to Support GUI Testing. https://wiki.cs.umd.edu/cmsc734_09/images/8/88/Viviz.pdf (Accessed December 23, 2015)

Engström, E., Mantylä, M., Runeson, P., Borg, M. (2014). Supporting Regression Test Scoping With Visual Analytics. In: Software Testing, Verification and Validation (ICST), 2014 IEEE Seventh International Conference on , IEEE, pp. 283–292 http://ieeexplore.ieee.org/xpl/login.jsp?arnumber=6823890 (Accessed August 28, 2015)

Fry, B.J. (2004). Computational Information Design. PhD thesis, Massachusetts Institute of Technology. http://benfry.com/phd/dissertation-110323c.pdf (Accessed December 23, 2015)

Itoh, T., Koyamada, K. (2003). HeiankyoView: Orthogonal Representation of Large-scale. In: Hierarchical Data, International Symposium on Towards Peta-Bit Ultra Networks (PBit 2003).pp. 125–130

Itoh, T., Yamaguchi, Y., Ikehata, Y., Kajinaga, Y. (2004). Hierarchical Data Visualization Using a Fast Rectangle-Packing Algorithm. IEEE Transactions on Visualization and Computer Graphics **10**(3) (May 2004) pp. 302–313, http://dx.doi.org/10.1109/TVCG.2004.1272729

Jenkins (2011). An extensible open source continuous integration server. https://jenkins-ci.org/ (Accessed August 28, 2015)

Jones, J.A., Harrold, M.J. and Stasko, J. (2002). Visualization of Test Information to Assist Fault Localization. In: Proceedings of the 24th International Conference on Software Engineering, ICSE '02, New York, NY, USA, ACM (2002) pp. 467–477 http://doi.acm.org/10.1145/581339.581397

Kienle, H.M., Müller, H.A.: Requirements of Software Visualization Tools: A Literature Survey. (2007). In: Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007. 4th IEEE International Workshop on, IEEE (2007) pp. 2–9, http://ieeexplore.ieee.org/xpl/login.jsp?arnumber=4290693 (Accessed December 23, 2015)

Mundigl, R. (2009). Clearly and Simply, Software Project Dashboards Episode 2. http://www.clearlyandsimply.com/clearly_and_simply/2009/12/software-project-dashboards-episode-2.html (Accessed December 23, 2015)

Opmanis, R., Ķikusts, P., Opmanis, M. (2015). Root cause analysis of large scale application testing results. Preprint (August 2015). https://dspace.lu.lv/dspace/handle/7/31010 (Accessed August 26, 2015)

SmartBear, Automated Test Visualizer, (2015). http://smartbear.com/product/testcomplete/ features/test-visualizer/ (Accessed December 23, 2015)

Steele, J., ed. (2010). Beautiful Visualization. 2 edn. O'Reilly Media, Inc.

TeamCity, (2006). https://www.jetbrains.com/teamcity/ (Accessed December 23, 2015)

Tufte, E.R. (2009). The Visual Display of Quantitative Information. 2 edn. Graphics Press LLC (2009)

Tulp, J.W. (2012). The Process of Creating Data Visualizations. http://blog.visual.ly/the-process-of-creating-data-visualizations/ (Accessed December 23, 2015)

Yang, J., Ward, M.O., Rundensteiner, E.A. (2003). Interactive hierarchical displays: a general framework for visualization and exploration of large multivariate data sets. Computers & Graphics, pp. 265–283

Wang, H., Zhang, X., Zhou, M.: MaVis: Feature-Based Defects Visualization in Software Testing. In: Engineering and Technology (S-CET), 2012 Spring Congress on, IEEE (2012), pp. 1–4 *http://ieeexplore.ieee.org/xpl/login.jsp?arnumber=6341917* (Accessed December 23, 2015)

WEB (a). XebiaLabs, Test Results Management. https://xebialabs.com/solutions/test-results-management/ (Accessed December 23, 2015)

WEB (b). TechTarget, Data visualization definition. http://searchbusinessanalytics.techtarget.com/ definition/data-visualization (Accessed December 23, 2015)

Received October 16, 2015 , revised January 13, 2016, accepted February 2, 2016