An Analytical View of Test Results Using CityScapes

, , Markus Borg[1], Andreas Brytting[2], Daniel Hansson[3]
[1]RISE SICS AB, Lund, Sweden, markus.borg@ri.se
[2]KTH Royal Institute of Technology, Stockholm, Sweden, andreas.brytting@gmail.com
[3]Verifyter AB, Lund, Sweden, daniel.hansson@verifyter.com

Abstract— In this paper we map test results from a real ASIC project on to the file structure of the design under test and present it as a cityscape. In the cityscape each house is a file where its height reflects the number of commits to that file. The color reflects the fraction of bad commits.

We identify error prone areas (red "bad" neighborhoods) as well as the most active areas (tall "downtown" areas). The cityscape also allows us to identify potential test coverage holes (tall green buildings) where there are a lot of activities but no failures.

Keywords—regression testing; test results; big data; cityscape

I. INTRODUCTION

Verification of large systems leads to challenging amounts of test results, which is further amplified by high degrees of automation [8]. Traditionally, verification results are analyzed manually either directly from verification logs, test result matrices, or in simple aggregated views such as project dashboards [15]. One approach to alleviate analysis of test results, e.g. from nightly verification runs, is to enable visual analysis [8], defined as "the science of analytical reasoning facilitated by interactive visual interfaces" [6]. In this paper, we propose visualization of verification results using a city metaphor, i.e., presenting a *cityscape*, and we present a proof-of-concept visual prototype.

One of the most recognized approaches to visualizing object-oriented source code is to depict software systems as three-dimensional cities, aka. cityscapes. Wettel and Lanza developed the pioneering tool CodeCity [18], but several similar solutions have been proposed in the last decade. Wettel and Lanza also presented empirical evidence from a controlled experiment with human subjects: their cityscape enabled both faster and more accurate completion of tasks related to software maintenance [19]. CodeCity maps software metrics to visual properties of "buildings" as follows: #methods is mapped on the height, #attributes on the base size, and lines of code on color intensity.

Our work is inspired by CodeCity, but we adapt the visual properties to the context of test result analysis. In this paper, we present how we use the Unity game engine [17] to create a visual prototype of test results from a large ASIC project. The visual prototype enables general exploration of the test results, as well as particular support for two important tasks for verification engineers: 1) identification of *error-prone parts* of the source code, and 2) recognition of potential *coverage holes*, i.e., parts of the code that might not have been tested sufficiently.

The rest of the paper is organized as follows: Section II introduces visualization in general and software visualization in particular. We also present a selection of the most relevant previous work. Section III describes the input test data and presents our method to create visual prototypes. In Section IV, we present screenshots from our visual prototype, focusing on error-prone parts and coverage holes, and discusses our findings from an initial user evaluation. Finally, Section V summarizes our work and outlines plans for future work.

II. BACKGROUND AND RELATED WORK

A. Fundamentals of visualization

Visualization includes any technique for creating images or diagrams to communicate a message. Creating successful representation of data has been a research target for decades. Tufte, one of the most cited visualization scholars, has formulated a number of principles and guidelines [16]. Tufte's advice include: 1) focus on the data: above all else, show the data, i.e., "maximize the data-ink" ratio, do not visualize thinks that is not present in the data and 2) avoid "chartjunk", i.e., visual elements that are not necessary to comprehend the data. Adhering to Tufte's recommendations often mean avoiding 3D and artistic color schemes, but in this paper, we present an example in which a visual representation in colorful 3D indeed is useful.

Software visualization means using visualization to show either static perspectives of software systems or their dynamic run-time behavior. Diehl divides this into three categories [7]: 1) structure refers to static parts and relations of the software system, i.e., the source code modules and the static call graph, 2) behavior refers to the execution of the software, i.e., memory allocation or communication between objects in object-oriented languages, and 3) evolution refers to the development process leading to a software system, i.e., the changes to the source code or the results from testing during development. The work we present in this paper is an example of software visualization belonging to the evolution category. We propose to bring visualization techniques proposed in software engineering contexts to the domain of ASIC verification.

B. Related Work on Cityscapes for Software Visualization

Wettel and Lanza pioneered using cityscapes in static software visualization [18]. They developed the tool CodeCity to enable interactive analysis of object-oriented software as 3D cities. In CodeCity, classes are represented as buildings in the city, while the packages are presented as districts. Each building has a height mapped to the number of methods in the corresponding class and the base size shows the number of attributes. In the tool, a number of different source code metrics can be visualized using colors of the buildings, e.g., the number of lines of code. The authors conducted a controlled experiment [19] with 10 individual tasks related to software maintenance and report that their subjects completed them better (+24% correctness) and faster (-12% completion time) compared to a control group working with Eclipse and Excel. However, the improvement was only seen on tasks that required a big picture overview of the system under study, not on focused tasks based on detailed information.

Several other authors have developed approaches to visualize software as cityscapes. Biaggi developed Citylyzer, a Java port of CodeCity as an Eclipse plugin [4]. Codstruction is another cityscape visualization plugin for Eclipse, tailored to support comprehension of software written in Java [5]. A third example of an Eclipse plugin for source code visualization is Manhattan, developed by Bacchelli et al. [1]. Garcia et al. developed a visualization for global software development with the goal to support decision-makers in allocating global development resources [10]. Their cityscapes focus on showing metrics related to product quality and development site productivity. Balogh and Beszédes developed CodeMetropolis, a cityscape that is created using the computer game Minecraft [2]. The authors decided to use a computer game for visualizing the cityscape because of its "high quality graphics and expressive power" combined with Minecraft's support for third party software integration. Merino et al. also used a game engine to create cityscapes [12]. Their tool CityVR uses Unity to create an immersive interactive experience using virtual reality.

Some authors have also used cityscapes to visualize test results. Balogh et al. proposed using their tool CodeMetropolis to visualize test-related metrics [3], e.g., the traditional code coverage and advanced metrics such as partition, specialization, and uniqueness of test cases. Sosnówka developed another approach to visualize the system under test well [14], focusing on low level test cases. He presents the number of test cases, execution status, modification dates, and number of executions on his cityscape.

The publications by Merino et al. [12] and Balogh et al. [3] constitute the most similar previous work. Both publications use game engines to create cityscapes for software visualization, CityVR is even developed in Unity. Balogh et al.'s work, on the other hand, uses a game engine to visualize aspects related to software testing. Consequently, our work combines aspects from both publications into a visual prototype, created by Unity, enabling visual analysis of regression test results. Moreover, in contrast to previous work, we focus our work on ASIC projects.

III. METHOD

The goal of our work is to enable visual analysis of test results to provide faster insights from automated verification activities.

A. Input Data

We collected five months of test results from a real ASIC project. This data included information about which commits to the revision control system that had caused regression test failures. This data is automatically available in the automatic debug tool PinDown [13], which was used in this project, but it could be generated in other ways as well, e.g. by a continuous integration tool such as Jenkins [11]. In addition, we collected the entire commit history of the revision control system including which files had been updated in each commit.

By combining these two data sets we produced a list of updated files. For each file we listed the number of commits (presented as the height of the buildings in the cityscape) and the percentage of good vs. bad commits (the color of buildings in the city scape).

B. *Generating the cityscape*

We use the game engine Unity [17] to create the test result cityscape. The motivation for using a game engine, and in particular Unity, is threefold. First, our previous research on visual analysis of regression test results stressed the importance of interaction [8] – and interaction is the primary purpose of a game. Second, contemporary game engines scale to visualize very large realistic worlds, i.e. very large quantities of historical test results can be presented. Third, Unity is an established solution that powers games such as Cities: Skylines and Pokémon GO, but still it offers a user-friendly game development platform with support for C# and JavaScript.

The visualization prototype we developed in Unity parses a csv-file containing the input data. Our prototype then generates a 3D cityscape and Unity provides intuitive mouse and keyboard navigation as the default option. We map the data to the cityscape as follows: a building represents a file, #commits are mapped to the height, and the percentage of successful commits is indicated by color from green to red. Finally, buildings are organized in neighborhoods according to the folder structure.

The user can navigate the cityscape using the standard WASD control scheme and a mouse. This control scheme has been the de-facto scheme in PC gaming since mouselook became standard in 3D games[1], especially first-person shooters, and thus we believe it should lower the entry bar at least for a subset of users. The mouse is used to direct the camera and the four WASD keys are mapped to movement as follows: W - move forward, S – move backward, A – move left (without turning), and D – move right (without turning). In addition, we implemented height navigation using the up and down arrow keys.

IV. RESULTS

We used 5 months of commit data from a real ASIC project and to create a visual prototype in the form of a cityscape. During these 5 months there were in total 1,544 commits out of which 68 commits were bad, i.e., resulted in a failed regression test case. This data was provided by PinDown, an automatic debugger of regression failures. There were in total 11,953 file updates made in these 1,544 commits. Each file update was marked as bad if it was part of a bad commit, even if the actual bug may have been introduced in another file update in the same commit. This means that a file update may be incorrectly marked as bad, just because it happened to be in a bad commit, but the hope was that for a large data set this would be manageable noise that would not falsify the global analysis. We continue by presenting three screenshots from the cityscape and then an initial evaluation with two senior verification engineers.

A. *Screenshots from the cityscape*

A cityscape provides a unique overview of the design under test. Traditionally, results from regression testing are presented in a test result matrix, cf. Figure 1. Cells in the matrix show the test verdict from executing a test case (or test suite) from a specific day. The bottom row in the figure shows the number of commits to the project during that day. For example, on the Wednesday, tests 3, 5, and 6 failed – possibly reflecting the many commits the day before.

Figure 2 shows a screen-shot from our visual prototype. The game engine allows the user to fly around the city and examine buildings and neighborhoods closer to understand the status of the design and the project. The backbone of the analysis is the following: 1) the higher the houses, the more commits to that file, and 2) the redder the color, the more failures associated with that commit.

[1] http://www.pcgamer.com/how-wasd-became-the-standard-pc-control-scheme/

	Mon	Tue	Wed	Thu	Fri
test1	pass	pass	pass	pass	pass
test2	pass	pass	pass	fail	pass
test3	pass	pass	fail	pass	pass
test4	pass	pass	pass	pass	pass
test5	fail	pass	fail	pass	pass
test6	fail	pass	fail	pass	pass
test7	pass	pass	pass	pass	fail
test8	pass	pass	pass	pass	pass
#commits	10	21	10	5	5

Figure 1. Example of a test result matrix.

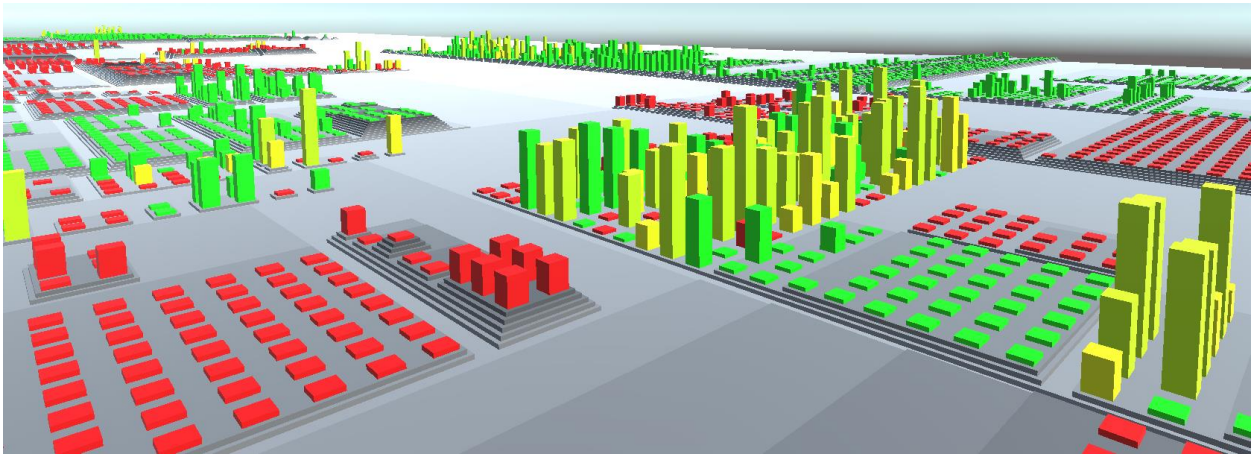

Figure 2. Cityscape overview of a design under test.

The cityscape allows intuitive identification of particular areas of interest. Figure 3 shows a close-up of an *error prone area*, i.e. a folder containing several files, most of which have been subjected to bad commits. Most of the files are still red, meaning the problems have not yet been fixed.

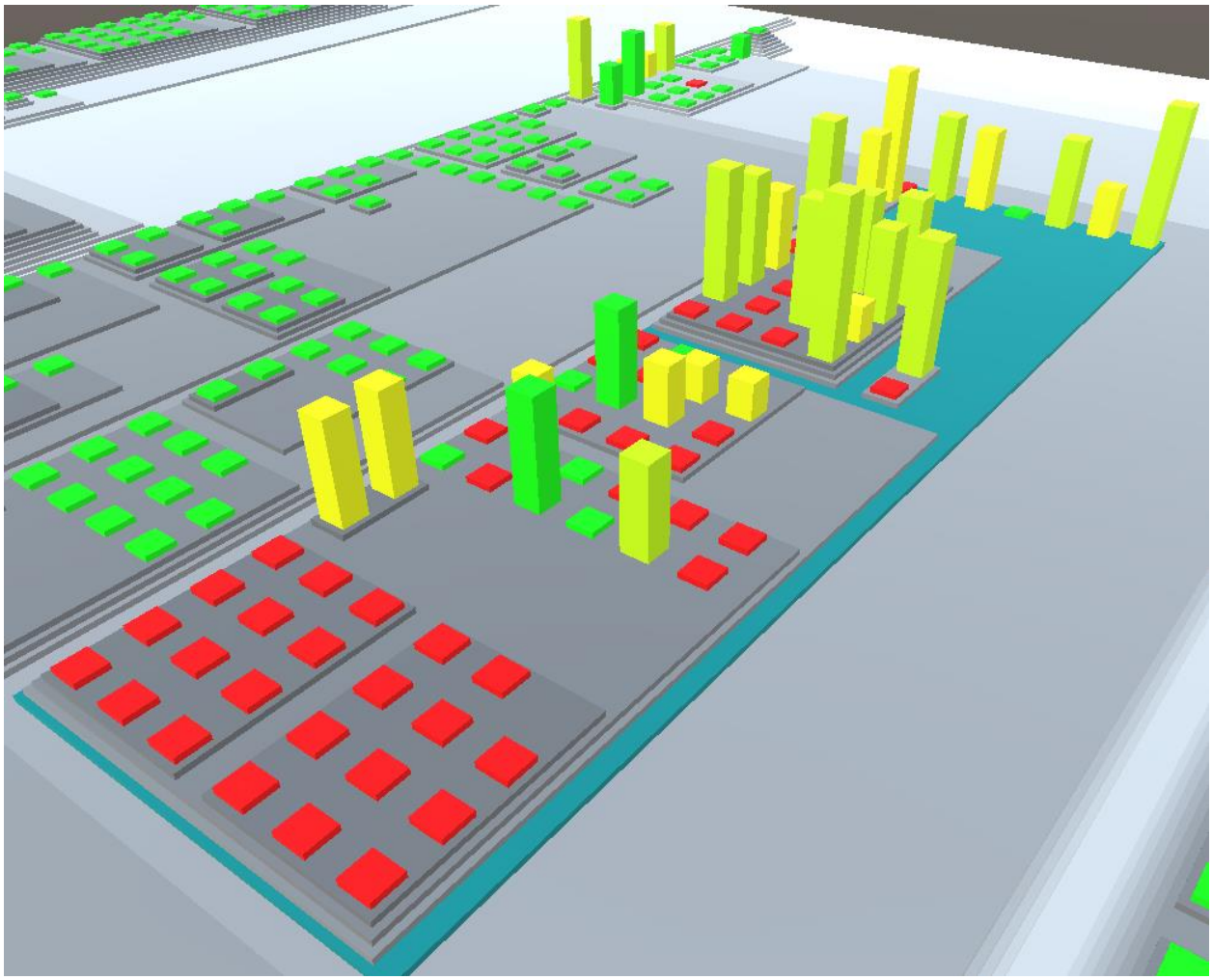

Figure 3. An error-prone area.

Figure 4 depicts an area which is fully green, meaning that despite the high number commits (as indicated by the height of the buildings) not a single file has ever been present in a regression bug during the five months this data set covers. That could be good news, an area where prudent engineers never make any mistake. However, it may also be that the regression test suites do not cover this area at all.

We conclude that the novel cityscape visualization of test results enables analyzability beyond traditional test result matrices. Our future work will focus on developing the interactive navigation in the city, and turn the visual prototype into a separate test results analysis tool.

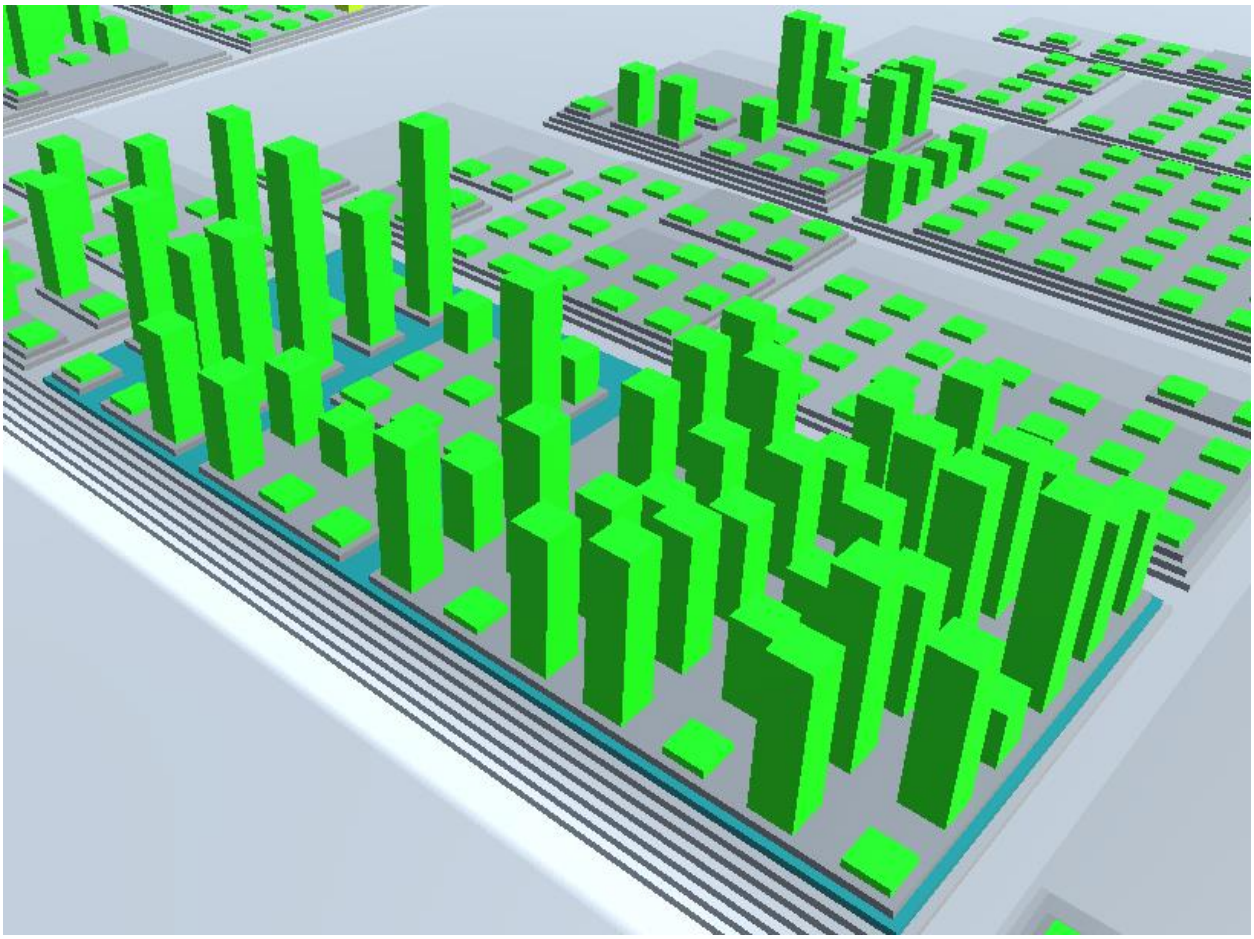

Figure 4. Potential test coverage holes.

B. Initial user evaluation

We have conducted an initial validation of our general visualization approach. As an initial user evaluation, constituting a sanity check of our cityscape view of the design under test, we presented our visual prototype to two senior verification engineers: the senior verification lead for the project visualized and the corresponding ASIC line manager. Both engineers knew the underlying data very well and were experienced in ASIC verification. Thus, they represent the group of target users we envision for our future test results analysis tool.

The evaluation was designed as an informal one-hour meeting guided by four open-ended questions:

1. Are the potential test coverage holes correctly identified?
2. Are the error-prone areas correctly identified?
3. Is this a useful presentation technique?
4. What improvements would you like to see?

Regarding the first question, they concluded that the presentation technique does correctly identify potential test coverage holes. The largest potential coverage hole that was identified was due to a test bench that was under development but that was not yet used in this specific project. Consequently, there were a lot of code updates to this test bench but no regression failures as the test bench was not run in this project (cf. Figure 4 showing high green buildings). This did not provide new information to them for this specific project, as they were well aware of the status of this test bench, but it did validate that this presentation technique can identify potential coverage holes.

Regarding the second question, the two verification engineers concluded that error prone code is also correctly identified. They were a bit surprised to see that the most error prone folders were not the RTL (Register Transfer Level) code with most of the project updates, but instead one specific part of the test bench, which stood out. They

decided to check why that specific area was so error prone and investigate whether something could be done to improve the quality of that area during development.

Regarding the third question, the verification engineers consider the cityscape perspective a useful presentation technique. They expressed an interest in regular analysis, e.g., analyzing each project every 6 months in this way in order to get a high-level view of the project state and of which areas that could be improved. Also at the end of each project this analysis would be useful as a part of the post-mortem analysis they do in order to identify potential improvements for future projects.

Finally, in relation to the fourth question, they identified some key improvements. First of all, it is very important to be able to remove individual commits from the input data set. Two commits in the data sets were big mergers updating a lot of files, which introduced a lot of noise that confused them in the beginning of the evaluation. These commits were correctly marked as bad as some of the file updates in these two mergers were bad, but because so many files were updated, most of the updates were actually good, but incorrectly marked as bad. This caused a lot of confusion at first as some of these file updates could not even theoretically introduce a test failure. This was solved by filtering out these two commits, whereby all this noise was removed making the analysis much easier. The lessons learned from this was that being able to filter out specific commits is very important and needs to be intuitive. This finding is in line with conclusions by Engström et al. [8], i.e., users must be able to interact with the visualization, to filter results and focus attention accordingly.

Apart from the filtering, the two verification engineers also mentioned some less critical improvement suggestions. First, they requested more information on how to navigate through the cityscape, something that was not always easy. We plan to add semi-transparent control instructions in a future version of the tool. Second, the verification engineers highlighted the need to generate a text reports from the visual prototype. When issues have been identified, the next step is to follow up by email and to such emails they want to attach a text report as a technical backup so that the person at the receiving end of the email understands the status.

All in all, we consider the initial user evaluation as successful and conclude that the cityscape brings a novel perspective for test analysis. The verification engineers would like to use it, provided that the cityscape can be automatically generated.

V. SUMMARY AND CONCLUDING REMARKS

Automated regression testing inevitably leads to large amounts of test results. It is no trivial task for verification engineers to stay on top of all testing activities. Inadequate overview of the testing process threatens productivity, and might lead to information overload, i.e., the amount of test results exceeds the verification engineer's processing capacity. On the other hand, successful overviews of test activities could enable new levels of analyzability.

We propose a novel approach to coping with large amounts of test results in ASIC verification. Using software visualization, we enable the analysis of test results provided by PinDown, a tool for automatic testing and debug. We implement our ideas in a visual prototype, demonstrating our ambition to generate cityscape presentations of test results. Our visual prototype is implemented in Unity, a 3D game engine that enables convenient interaction with user controls well-established in PC gaming, i.e., WASD keys and mouselook.

As a proof-of-concept, we developed a visual prototype for an ASIC project with five months of commit data. The visual prototype can be used by a verification engineer for general exploration of the test results. Furthermore, it can be used for two specific tasks related to the analysis of test result: 1) identification of error-prone parts of the source code and 2) recognition of potential coverage holes, i.e., parts of the code that have not been tested enough by the regression test suites.

We also present an initial user evaluation of our visual prototype with two senior verification engineers. The evaluation showed that the cityscape can correctly identify both coverage holes and error-prone areas of an ASIC project. While the overall evaluation result was positive, i.e., the two verification engineers were positive and encouraged us to continue development, they also provided valuable suggestions for the future. For example, they requested an export function to generate textual reports directly from the visual prototype and they would like to see navigation instructions on screen. Based on the evaluation, we conclude that the cityscape presentation of regression test results is a promising approach and we plan to continue our work.

We have planned several improvements to our visual prototype. First of all, we will implement the improvement suggestions proposed during our initial user evaluation, such as simplified user navigation and text report generation. Second, we plan evolve the visual prototype into a self-explanatory tool. Several minor improvements must be implemented to turn our ideas into a mature tool, e.g., improved navigation, filtering and selection options, and tooltips. Third, we want to help the user to quickly identify relevant areas and buildings in the cityscape. We still believe the most important use cases are to help verification engineers to locate potential coverage holes and error-prone areas. Finally, we want to design a more comprehensive tool evaluation study, preferably combining focus

groups with experienced verification engineers and controlled experiments with larger numbers of (probably student) subjects.

Another area of for future research is to add functional coverage data to this type of presentation. Having both test results and functional coverage data would provide a more complete picture, especially in the area of potential coverage holes.

ACKNOWLEDGMENTS

This work has been financially supported by the ITEA3 initiative TESTOMAT Project (www.testomatproject.eu) through Vinnova – Sweden's innovation agency.

REFERENCES

[1] A. Bacchelli, F. Rigotti, L. Hattori, and M. Lanza. "Manhattan - 3D City Visualizations in Eclipse", Eclipse IT, 2011.
[2] G. Balogh and A. Beszedes. "CodeMetropolis-code visualisation in MineCraft", In Proc. of the 13th International Working Conference on Source Code Analysis and Manipulation, 2013.
[3] G. Balogh, T. Gergely, A. Beszedes, and T. Gyimothy. "Using the city metaphor for visualizing test-related metrics", In Proc. of the 23rd International Conference on Software Analysis, Evolution, and Reengineering, 2016.
[4] A. Biaggi, "Citylyzer – A 3D visualization plug-in for Eclipse", BSc Thesis, University of Lugano, http://www.inf.usi.ch/faculty/lanza/Downloads/Biagg2008a.pdf, 2008.
[5] Codstruction, Java visualization tool, https://codstruction.wordpress.com/about/ (Oct 31, 2017)
[6] K. Cook and J. Thomas, "Illuminating the path: The research and development agenda for visual analytics," Pacific Northwest National Laboratory (PNNL), Tech. Rep. PNNL-SA-45230, 2005.
[7] S. Diehl, "Software visualization – Visualizing the structure, behavior, and evolution of software", Springer, 2007.
[8] E. Engström, M. Mäntylä, P. Runeson, and M. Borg. Supporting Regression Test Scoping with Visual Analytics, In Proc. of the 7th International Conference on Software Testing, Verification and Validation, pp. 283-292, 2014.
[9] N. Erman, V. Tufvesson, M. Borg, P Runeson, and A. Ardö, "Navigating Information Overload Caused by Automated Testing: A Clustering Approach in Multi-Branch Development", In Proc. of the 8th International Conference on Software Testing, Verification and Validation, pp. 1-9, 2015.
[10] F. Garcia, M. Moraga, M. Serrano, and M. Piattini. "Visualisation environment for global software development management" IET Software 9(2), pp. 51-64, 2015.
[11] Jenkins, Continuous Integration Tool www.jenkins.i
[12] L. Merino, M. Ghafari, C. Anslow, and O. Nierstrasz. "CityVR: Gameful software visualization", In Proc. of the 5th Working Conference on Software Visualization, 2017.
[13] PinDown, Automatic Debug, www.verifyter.com
[14] A. Sosnowka. "Test City metaphor as support for visual testcase analysis within integration test domain", In Proc. of the Federated Conference on Computer Science and Information Systems, 2013.
[15] C. Treude and M. Storey, "Awareness 2.0: staying aware of projects, developers and tasks using dashboards and feeds", In Proc. of the 32nd International Conference on Software Engineering, pp. 365-374, 2010.
[16] E. Tufte, "Envisioning information", Graphics Pr, 1990.
[17] Unity, Game Engine, https://unity3d.com/
[18] R. Wettel and M. Lanza, "Visualizing software systems as cities", In Proc. of 4th International Workshop on Visualizing Software for Understanding and Analysis, pp. 92-99, 2007.
[19] R. Wettel, M. Lanza, and R. Robbes, "Software Systems as Cities: A Controlled Experiment", In Proc. of the 33rd International Conference on Software Engineering, pp. 551-560, 2011.