Advances in the characterization of cognitive support for unit testing

The bug-hunting game and the visualization arsenal

Marllos P. Prado
Instituto Federal de Goiás
Universidade Federal de Goiás
Goiânia GO, Brazil
Email: marllos.prado@ifg.edu.br

Auri M. R. Vincenzi
Universidade Federal de São Carlos
São Carlos SP, Brazil
Email: auri@dc.ufscar.br

***Abstract*—In the previous edition of ISSRE we used a straightforward hunting metaphor to call the attention of the research community to the importance of considering human aspects in software testing—in particular, to research challenges that affect practitioners in the industry. Our focus was on the unit testing level. As a result, we proposed a framework to indicate research directions in three cognitive support dimensions defined to this level of testing. In the present work, we advance in this area considering the visualization domain. We demonstrate an immediate application of our framework to help derive information on how visualizations for unit testing are currently addressing each cognitive support dimension. Our results demonstrate the lack of concern with adequate procedures for both planning and validation of the visualizations regarding cognitive support aspects. These issues affect the capacity of the proposals to support users and put under risk the solutions' reliability.**

I. Introduction

The cognitive support can be understood as the assistance that the tool provides to the users in their mental tasks [1]. Considering this, what are the consequences if the user's mental tasks are not adequately facilitated by an existing tool? What happens if these users are actually, professionals involved in the quality assurance process of a product as complex as software? Motivated by these questions, our research team has tried to call the attention of the community to the lack of current research addressing these kind of problems, starting by the most elementary level of test: the unit testing.

The testing professional is an essential part of the loop for many practical solutions and the research of adequate support for tester's mental tasks deserves the same attention that is dedicated to the research of any technical aspect of automation. The reason is simple: for any new degree of automation reached, the humans can re-adapt themselves and orchestrate the automated solution to solve gaps where the automation alone is still expensive or immature. Bringing back the hunting game metaphor that we adopted in Prado et al. [2], testers can be seen as hunters and bugs as preys. The hunters must optimize their chances of catching preys, especially the most valuables. Trying to substitute humans by automated weapons is few adaptable and ineffective to find skittish game animals. In contrast, using only humans natural force is not a safe option. Thus, we advocate the research of weapons' features that could improve professionals' skills and help them master their capability of discovering the bugs.

We bring new elements to the bug-hunting scenario in the present paper: before we researchers spend our resources and effort proposing any new artifact to help the testers in their hunt, let us inspect the vast weaponry currently available and evaluate their features. The chosen category is the visualization arsenal, considering its natural potential to aid in cognitive issues. For this purpose, we investigate: how these tools were planned; what are the promissory features offered; and how they have been evaluated on their usefulness to aid the professionals of testing. Considering our interest in unit testing, we defined a strategy in three steps: first, we selected unit testing tools from visualization literature; second, we applied the cognitive framework proposed in Prado et al. [2] to derive, relate and analyze the visualizations. Finally, we summarize and present the cognitive support gaps and opportunities identified. The strategy adopted shows an immediate application of our framework in discovering some of these cognitive support gaps. The knowledge summarized in this paper should help the research community in the understanding and discussion of important questions regarding this immature aspect of unit testing tools and serves to improve tools' reliability in future proposals.

In Section II, the motivations, background and the strategy to conduct the research are presented. Section III presents the results and analysis for the cognitive support aspects studied. Finally, Section IV concludes and discusses further research.

II. Selecting the visualization arsenal: background and strategy

The visualization can amplify the cognition in several different ways. Card et al. [3] classify six major categories: (i) increasing user's memory and processing resources; (ii) reducing search for information; (iii) enhancing the recognition of patterns; (iv) enabling perceptual inference operations; (v) perceptual monitoring and (iv) encoding information in a manipulable medium.

Two concepts must be understood from the visualization vocabulary in order to determine if a given visualization is useful to the user: *expressiveness*, which corresponds to the capacity of the visualization to transmit all (and only) the information of interest to the user and *effectiveness*, which corresponds to how clearly the user understands the information represented [4] [5]. Considering these definitions, a fundamental question we should ask when proposing any new visualization is: is this solution *expressive* and *effective* to the target audience? Both of these attributes are intimately related to the cognitive support that will be provided. According to Kosslyn et al. [6], we visualize the objects involved in daily problems, in order to answer questions and find solutions for them—that is, we mentally represent the visual images associated to the problem being solved. Now, imagine the consequences if the visualization proposed to solve a certain problem brings more or less information than needed (*expressiveness* problem) or is harder to understand (*effectiveness* problem) than the raw data that it represents–in our context, the source code for example?

A similar analogy could be made in the testing field: "How one discovers a bug occurrence of type A in a program B, using a test set C?". Considering two different visualizations to help solving this problem, how to determine the more *effective* choice? Could one be better to facilitate this bug-reveling task? How to compare them? These are non-trivial problems that may affect the user's behavior in the use of the visualizations and should be considered in their planning and validation.

Some good practices are encouraged by the visualization literature to orient the visualization definition process. One alternative is the adoption of a visualization technique designed and evaluated for a similar problem. Another is to define a new visualization based on a reference model. A reference model can be understood as a guidance to map the raw data (e.g. program or test case code) into visual structures and their controllers. Moreover, it serves to discuss, compare and contrast visualizations [3].

Besides the above-mentioned general aspects, we also need to pay attention to cognitive support issues related to the specific problem addressed by the visualization—in our case, unit testing tasks or artifacts. In the framework presented in Prado et al. [2], three dimensions were proposed to orient the cognitive support research on unit testing tools: (i) *The Tester Artifact Comprehension dimension* which addresses factors which affect users in the task of abstracting and understanding unit testing artifacts; (ii) *The Tester Orientation and Guidance dimension* which addresses factors which influence user's decision on unit testing tasks; (iii) *The Tool Interaction Usability*, which addresses usability aspects in the tool's interface.

Considering these concerns, we defined a strategy to advance in the characterization of unit testing cognitive support needs and opportunities in the visualization domain, as follows:

1) Obtain works which apply visualization in software testing in general and select those related to unit testing.
2) Read, analyze and interrelate the proposals' aspects according to the cognitive support dimensions defined in Prado et al. [2].
3) Discuss identified gaps and opportunities regarding the planning and validation of the visualizations from a cognitive support perspective.

The details of steps 1 and 2 are presented in the following subsections. The step 3 is presented in the Section III.

A. Step 1: Selecting visualizations with the "unit test caliber"

The papers selected for analysis were obtained from a systematic mapping study previously conducted by our research team. The goal of this mapping was to identify papers about visualization tools or techniques that support software testing activity in general.

In the first stage of the systematic mapping study (Identification Step) a total of 649 papers were returned from the three considered digital libraries (232 from ACM, 348 from IEEE and 69 from Science Direct). A total of 76 papers were detected as duplicated resulting in 573 papers actually considered.

In the second stage of the study (Selection Step) the review process was carried based on the papers' titles, abstracts and eventually - when these were not enough - based on the reading of the introduction and/or conclusion sections of the papers. Papers were initially read by one reviewer, in order to determine which paper should be kept or discarded. Then, the reviewer gave his advising. After, another reviewer made his judgment considering the same set of papers. Papers accepted/rejected by both reviewers were automatically accepted/rejected for the next step. In case of divergent opinions, the paper was set as accepted to be read on the next phase. Additionally, in cases when two or more papers from the same author and describing the same technique were found, only the most recent publication was considered. After completing the analysis of the 573 papers, 485 were considered rejected and 88 were accepted and passed to the following step.

In the last stage (Extraction Step) each of the 88 identified papers on the previous phase were fully read and classified according to the inclusion and exclusion criteria. From the 88 studies, 3 were considered duplicated, 21 were rejected and 64 were accepted. The classification criteria defined in the protocol of the systematic mapping study were concomitantly evaluated at this step, after each accepted paper was read (see Figure 1).

From the total of 64 papers obtained in the final stage of the mapping, 12 address the unit testing level. Among the 12 unit testing papers, 11 include a tool proposal. Only 1 work was dedicated to experimental study. From the papers related to tools, the works of Muto et al. [7] and Grechanik et al. [8] are evolutions of Muto et al. [9] and Conroy et al. [10], respectively. For simplicity in these cases the oldest papers were merged with their evolutions in our study.

The artifacts from the systematic mapping study (search string, protocol, data from extraction phase) are available for consultation at [11].

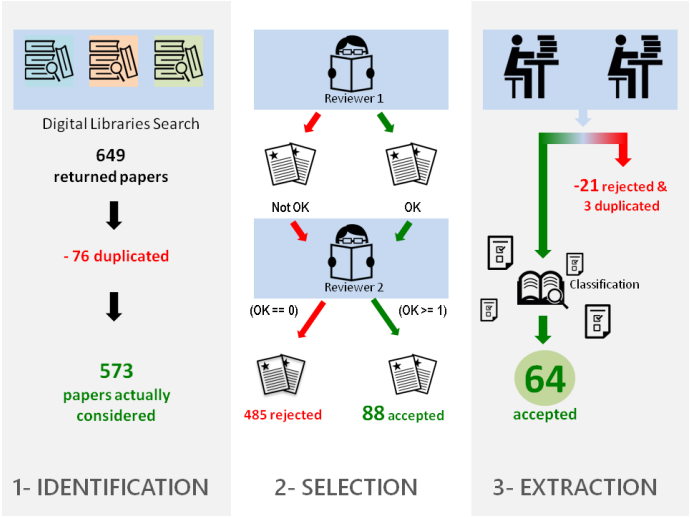

Fig. 1. Infographic summarizing the systematic mapping process.

B. Step 2: The framework application

The task of designing a tool that addresses cognitive support requirements is a centered user approach. As a consequence, to evaluate the cognitive support offered by a tool already developed, the focus of analysis needs to be in the pathways of interaction between the tool and user rather than the tool itself (be it based on a visualization or not). In the present work we don't have the presumption of detecting all the possible cognitive support issues in the analyzed tools. In fact, such evaluation would demand at least one separated and comprehensive research project for each single tool and the participation from both the tool's designers and the respective user groups of interest. Such evaluation is beyond the scope of this paper. Instead, our work consciously sets its boundaries in showing the most salient cognitive support features and issues identifiable in the tools based on the dimensions indicated by our framework [2].

The framework proposed in Prado et al. [2] emerged from a review of several studies in the literature that involved the participation of users from practice in the unit testing activity. Consequently, we consider that the derivation of the visualization proposals from step 1 in the three dimensions of the framework is an important step towards the characterization of how cognitive support is being addressed by these tools. The result of this derivation also serves to arrange the tool's cognitive support aspects by similarity and facilitates their comparison. We adopted the following process to apply this derivation: for each considered paper we (i) fully read the paper, (ii) understood the visualization in the context of the paper's problem and (iii) answered the following research questions for each corresponding framework dimension:

Tester Artifact Comprehension dimension: (R.Q.1) How the visualization addresses the problem of reformulating unit testing artifacts to facilitate their comprehension?

Tester Orientation and Guidance dimension: (R.Q.2) How the visualization addresses the problem of orienting tester on unit testing tasks?

Tool Interaction Usability dimension: (R.Q.3) How usability issues are addressed in the visualization tool?

III. Results and Analysis

This Section is organized by visualization type. For each type, the results are discussed by framework dimension.

A. Visualizations for structural testing

1) Tester Artifact Comprehension: In the work of Muto et al. [7] the visualization reformulates unit testing artifacts by representing the importance of units under test, their passage rate and static checking results using a combination of digraph and pie-chart. Nodes are classes and edges indicate their methods' caller-callee relationships. Each node is also a pie-chart (see Figure 2) that represents either the result of unit testing or static checking passage rate of the corresponding class. The thickness (weight) of the edges indicates the frequency that the methods from one class calls another class. The node's weight corresponds to the sum of edges' weight arriving in it. The heavier the node (more important), the higher position it assumes in the graph layout.

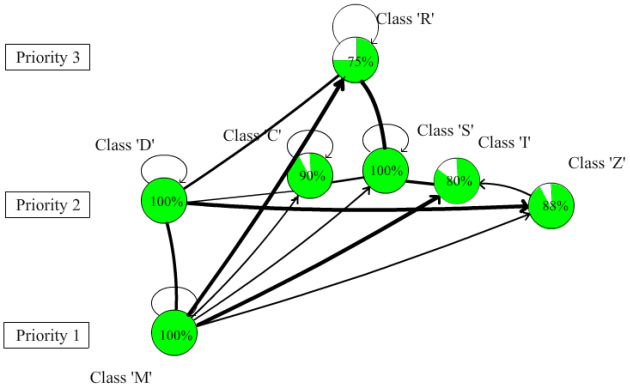

Fig. 2. Example of the visualization model of Muto et al. [7](redrawn and adapted)

The visualization proposed by Karam and Abdallah [12] represents the functions to be tested (units) as control flow graphs (CFGs) and data flow graphs (DFGs). The coverage of the program attributes (statements, control or data flow interaction) are visually mapped by colored visual attributes. For example, the nodes (program statements) only traversed by passing test cases are colored in green; the nodes only traversed by failing test cases are colored in red; the nodes traversed by both passing and failing test cases are colored in yellow. Statements, control commands, loops and variable definition-uses are represented by symbols and customizations in the traditional graph representation.

The visualization proposed by Estefo [13] (see Figure 3) represents the differences between pairs of test case execution

profiles using a polymetric view visualization technique [14]. Each view (called blueprint) generated by the tool shows the differences between a pair of test cases executions for the classes and methods invoked by them. In the blueprint diagram, classes are surrounding boxes, methods are inner boxes and edges represent calls between methods. Red and blue colors are attributed to each compared test case. Inner boxes only red or only blue are solely executed by the test case with the corresponding color. Yellow boxes are executed by both. The height and width of the boxes indicate respectively, the number of times that red and blue test cases executed the method.

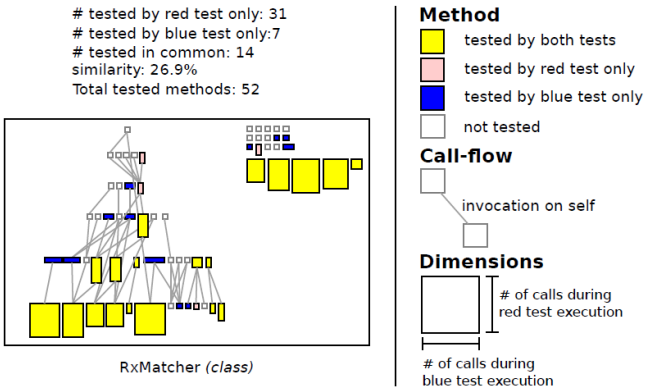

Fig. 3. Blueprint of Estefo et al. [13] showing the differences between a pair of test cases executions (reused with permission from Estefo et al. [13])

2) Tester Orientation and Guidance: In the work of Muto et al. [7] the algorithm used to layout the digraph and the subject's comments reported in the experiment section indicate that the visualization helps to restrict the scope to the classes that need to be prioritized in the search for bugs.

According to Karam and Abdallah [12] the coloring of their graph "lets the viewer spotlight only those program attributes and focus more clearly on them" that is, they try to restrict the focus of the user to subsets of interest in the testing required elements considered.

The colors and shapes of the boxes represented in the blueprints of Estefo's visualization [13] serve as references to the users in the grouping and selection of test cases that are potentially similar, considering their impact in each method. The identification of similar test cases helps in the refactoring tasks.

3) Tool Interaction Usability: None of the works provide information regarding the tool's usability.

4) Analysis: Although the visualization of Muto et al. [7] shows useful to reveal the discrepancies between unit testing and static checking results, we detected a potential risk to the pie-chart's *expressiveness*. In their work, the unit testing passage rate is calculated using the class statement coverage result. However, statement coverage is recognized as an easy to satisfy criterion, even for weak tests [15]. Assuming a case that a pie-chart node is 100% fulfilled for both unit testing and static checking results, this node may represent a class with no error. But it may also represent a class with errors tested by low-quality test cases and low-quality static checking annotations that keeps resulting in 100% coverage. Both are divergent meanings with the same visual mapping. Similarly, the visualization of Karam and Abdallah [12] seems susceptible to *expressiveness* problems. The program attributes traversed by both passed and failed test cases are colored in yellow. However, their paper did not mention the differentiation of the yellow coloring for cases when different percentages of failing and passing test cases occur over the same exercised program attribute. Thus, our interpretation is that for cases e.g. when 10% of the test set pass and 90% fail traversing a certain program attribute would have the same visual mapping (for the considered program attribute) of cases when 90% of the test set pass and 10% fail traversing it. We tried to double-check this information with the first author but we did not get an answer until the time of submitting this paper. A variation in the yellow darkness for example, could be a possible way to distinguish the exemplified cases and could add important information for the tester e.g. to track how close they are from changing an yellow program attribute to green or red.

The yellow color in the boxes of Estefo's visualization [13] does not present such *expressiveness* problem because each blueprint only corresponds to one pair of test cases.

The intention of Muto et al. [7] and Karam and Abdallah [12] in restricting the users' focus to important factors in the graphs' structure can be better explained by a decision-making heuristic called "Elimination by Aspects" [16]. This strategy can be applied when a person have to make decisions considering a great number of choices. Instead of mentally manipulating and pondering all the aspects of each alternative, he focuses in one attribute per time as a minimum criterion. This enables him to iteratively exclude the options that do not satisfy this criterion and prioritize choices. Alternatively, the visualization of Estefo [13] uses the boxes' colors and shapes similarities to support testers in the grouping and recognition of methods that are similarly affected by pairs of test cases. Such exploration of the visual shapes recognition and of the grouping by similarity can be better explained by the Bertin's retinal variables [17] and Gestalt principles[18]. However, the three works present limited information about the design and evaluation of the visualizations regarding users' cognitive aid. To ensure an adequate cognitive support, it would be important, for example: to perform a prior user study (adopting methods like interviews, surveys, focus group analysis) to understand how the users previously dealt with such problems in their unit testing daily tasks; what outcomes from those user studies motivated the definition or customization of the visualization features in a particular way; in case the decision making and recognition strategies that we mentioned were intentionally enforced by the visualizations, what aspects observed in the studies with the users motivated their implementation in the proposed form; to assess the extent to which their visualizations adequately benefits the users and what are the possible risks regarding visualization

interpretation.

Lawrence et al. [19]—the only selected study completely dedicated to experimentation—found some important results regarding the influence of visualization for structural-testing in users' performance and behavior. One of their experiment's result for example, did not show significant differences in the number of faults found between the treatment group (using visualization of block coverage) and the control group (not using visualization for the same criterion). A second result showed that the number of test cases generated was not affected by the adoption of a visualization, although the variability in the amount of test cases wrote was reduced in the treatment group. Finally, another result suggested that the visualization can affect the users' behavior by encouraging them to overestimate the effectiveness[1] of their tests. According to the authors, this was possibly due the positive visual feedback obtained when the criterion adequacy was reached. Considering the last two findings, the influence of visualization for structural testing over users are divergent: it can orient the behavior of the testers in negative ways e.g. inducing them to stop tests earlier because of the overestimation effect. However, visualization can be useful for standardization purposes once it reduces the variability in the number of generated test cases.

B. Visualizations to test legacy system

1) Tester Artifact Comprehension: The visualization proposed by Conroy et al. [10] helps in the testing of web services generated from legacy GUI systems (abbreviated as GAPs). The visualization provides a way to visually link the GUI elements from the legacy system's screens to the corresponding web service interface, serving as a connection to transfer input/output values between them (see Figure 4). The user defines the GUI elements of interest using drag-and-drop actions from the GAPs' screen to the visualization tool. After that, the user can link the GAPs and Web Service by drawing arrows between the getters/setters that represent the GAPs' GUI elements in the tool and the target web service interface. This mechanism helps in the capturing of useful data for the definition of the initial unit test set.

The visualization of Lienhard et al. [20] helps the users in the identification of relevant parts of legacy code for testing purpose (execution units). The visualization is composed of two main parts: (i) the execution trace and (ii) the test blueprint (see Figure 5). The execution trace uses a bi-dimensional view to represent method executions along the time for a running exercise of the program. The visualization adopts black and gray colors to show if the methods (represented by the vertical rectangles) were already covered by a test, or not. The rectangle's position indicates their occurrence in time and the vertical size indicates the time spent executing. The test blueprint part uses a graph visualization—similar to the UML object diagram—to represent the methods selected as an execution unit candidate by the user. It is used to represent objects that existed before the start of the execution unit

[1]Used as in the original work i.e. to indicate the capacity of finding errors.

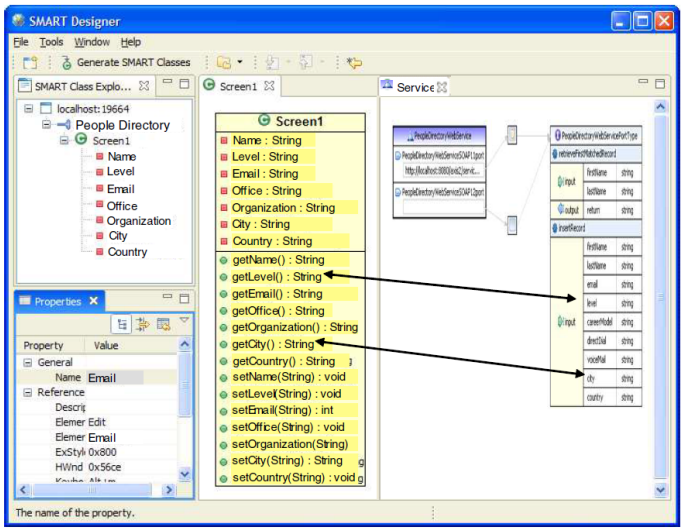

Fig. 4. Visualization of Conroy et al. [10] (reused with permission from Conroy et al. [10])

(nodes with labels in regular typeface); objects instantiated by the execution unit (nodes with labels in bold typeface); references between object that have been accessed during the running of the execution unit (a black arrow between the objects); references between object that already existed before the execution unit was run (a gray arrow between the objects) and side effects produced (a pop-up window displayed when the cursor is moved over the objects).

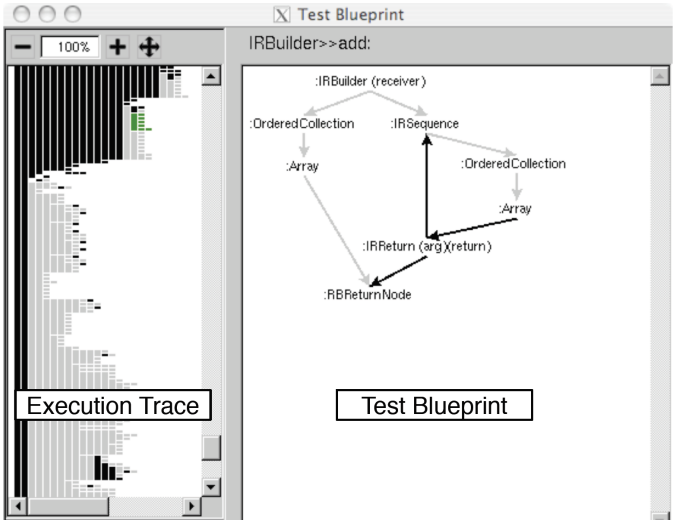

Fig. 5. Visualization of Lienhard et al. [20] showing the execution trace and the test blueprint part (reused with permission from Lienhard et al. [20])

2) Tester Orientation and Guidance: In the work of Conroy et al. [10] the visual link of the legacy system's interface to the corresponding web service is a mechanism to enable the reuse of the legacy system users' knowledge in the derivation of important input/output values. These values captured by the tool and converted into unit test cases orient testers in the

establishment of their initial test cases baseline.

The visualization of the execution trace part in the work of Lienhard et al. [20] tries to facilitate the task of electing an execution unit by setting some references for this purpose—e.g. the black and gray color of the rectangles indicating their coverage and the occurrence in the time, indicated by the rectangle's position. It also helps users to restrict their problem scope by providing interaction mechanisms such as filtering the amount of information in the execution trace view (e.g. show only methods from a particular package/class). The visualization of the test blueprint part provides references for the developer in the tasks of: (i) implementing the fixture of the test case; (ii) executing the unit under test; and (iii) writing the assertions. For the fixture implementation for example, the nodes with labels not in bold are objects that need to be created.

3) Tool Interaction Usability: None of the works provide information regarding the tool's usability.

4) Analysis: Conroy et al. [10] say that they evaluated their proposal through a feasibility study. They say that their experience confirms the benefits of the tool but they do not give information regarding subjects' participation. Thus, we could not obtain details regarding the *effectiveness* or *expressiveness* of the visualization with users. In contrast, the work of Lienhard et al. [20] has a limited number of subjects but tries to answer important questions related to the *effectiveness* of the visualization (e.g. "How straightforward is it to use [the visualization] tool?") and its *expressiveness* (e.g. "Does the Test Blueprint concisely communicate the required information?").

It would be pertinent to investigate the extent to which the visualization of Conroy et al. [10] supports testers in the application of the "Satisficing" [21] strategy. This strategy is often applied when the resources are limited and the person needs to find the trade-off between the selection of choices that satisfy his requirements—in this case, the unit test cases related to frequently used operations and obtained by their tool—and the minimization of possible losses e.g. time, people or other resources necessary to understand the legacy code. Lienhard et al. [20] also do not directly assess users' decision-making strategies in their validation. However, they show concerns with results affected by such decisions. For example, their second case study tries to determine "How do tests written using [their] approach differ from conventional tests written by an expert". One of their results showed that some important assertions were discovered because of their visualization and that they had been previously ignored.

C. Visualizations linking unit testing to system scope

1) Tester Artifact Comprehension: The visualization proposed by Wang et al. [22] represents a "feature-based testing model" i.e. a model that represents the system under test, its test cases and defects. The visualization in the tool consists of two parts: "system structure tree" and "result charts". The system structure tree part represents the feature-based testing model. A leaf node represents a feature, which corresponds to an atomic element of the system for testing purpose (a unit). Internal nodes represent modules of the system—a set of features or recursively, a set of modules. The nodes are colored from red to green and express the rate of defects divided by test cases of each feature—greener for low rate and deeper red for high rate. The result chart part is an histogram used to represent the "complexity coverage"—a metric used by the authors to represent the number of test cases and features for each feature complexity type. Consequently, users need to pre-classify the features in one of the three types of complexity defined—high, medium and low—based on the complexity of the implementation and business logic.

The visualization proposed by Cottam et al. [23] represents unit testing results from the MPI Testing Tool (MTT) in a aggregated form to enable high level interpretation by the MPI community. Their visualization compress test suite data from a N-dimensional space into a two-dimensional grid representation. The x-axis represents the test suite architecture and OS information and the y-axis represents the bitness and compiler family. The background color of each grid's cell is used to map the state of the "trivial" test suite—a test suite whose failure implies in the failures of all other tests. Glyphs in the foreground of the grid's cell visually map the results of each non-trivial test suite. These glyphs are also pie-charts used to indicate the passage rate of the non-trivial test suites.

2) Tester Orientation and Guidance: The visualization of Wang et al. [22] orients users in finding the bugs by simultaneously giving a global and a fine-grain view of the system under test. Interaction mechanisms are provided to fold and unfold nodes, enabling the user to open or restrict their focus on the nodes considered. The colors of the nodes indicate the seriousness of the problems in the features represented by the nodes. The histogram indicates if the current test cases are enough to each feature complexity type, helping the users to decide on whether to add more test case for each complexity type or not.

The visualization of Cottam et al. [23] was validated using only a specific combination of testing data available from the MTT. However, they indicate that the visualization can be adapted to represent a variety of data combinations. Considering the variety of data combinations that can be represented, the visualization enables the users to continually and accurately monitor the unit testing results and coverage for the MPI project in a compacted form.

3) Tool Interaction Usability: None of the works provide information regarding the tool's usability.

4) Analysis: The visualization of Wang et al. [22] does not show information about the users involvement in the planning stage. Moreover, their work do not report any validation.

The work of Cottam et al. [23] is a comprehensive demonstration of practices for the adequate support of user's cognition through visualization. First, the target public was considered since the initial design steps. This involved: understanding how information was previously obtained from raw data; classifying users into groups; understanding the kind of information needs for each group and prioritizing them. The users'

opinion was decisive for many choices made along the process of visually mapping the data such as: the visual metaphor applied; colors and color tones adopted; the double-encoding of the data to solve particular cases where the information comprehension could be affected (see Figure 6). The authors also evaluated the visualization regarding its expressiveness and effectiveness to the target users. This evaluation included questions to: (i) compare users interpretation using the visualization to the previous textual mode; (ii) discover users insights facilitated by the visualization; (iii) determine the capacity of the users to reach their goals using the visualization; (iv) check improvements in data representation and user assistance. Considering the variety of information that their visualization can provide and the goals they can serve, the evaluation of one or a few decision-making strategies would be an extensive and unfeasible task. Instead, the authors asked users to list insights provided by the visualization tool that could not be noted before. The authors also encouraged user's criticism over their visualization.

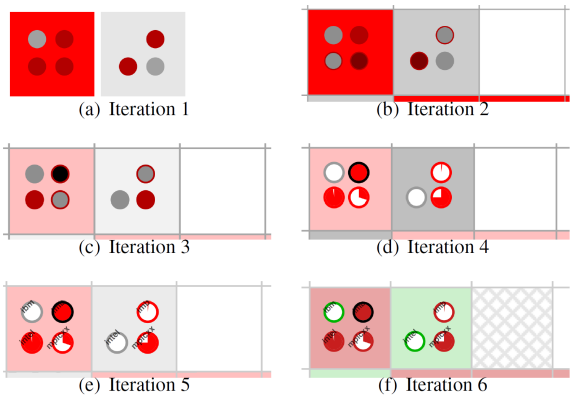

Fig. 6. The several iterations showing the evolution of the encoding of test suite in grid cells of Cottam et. al [23] (reused with permission from Cottam et. al [23])

D. Miscelaneous Visualizations

1) Tester Artifact Comprehension: The visualization proposed by Daniel and Sim [24] facilitates the task of selecting test inputs that kill mutants of Boolean Algebra Expressions (the units under test) by prioritizing the test inputs according to their likelihood to kill more mutants at a time. This way, a test input A that kills more mutants than a test input B is ranked first. The testing inputs are represented by colored cells in a matrix window. Each cell is colored using a 8-bit grayscale color. Darker boxes represent test inputs that can kill more mutants at a time. User can execute the test input by clicking in the boxes, which becomes red after that. The boxes are automatically updated after each test input execution. The visualization also provides an alternative view that indicates the faults (mutated Boolean Algebra expression) that can be detected by the currently selected test input.

The work of Romero et al. [25] proposes a tool to facilitate the running and analysis of unit testing of programs that use images as input data. Their visualization reformulates unit testing artifacts by making visible and trackable the images (input data) and the modifications that are made over them by the program under unit testing, in a periodical refresh rate. This permits that the user monitors the results of the test execution visually, instead of relying only in the assertions' results.

2) Tester Orientation and Guidance: In the visualization of Daniel and Sim [24] the representation of the colored cells and the dynamic update serve as a guidance for the non-expert user to progressively select a small subset of the test cases that kills all (or almost all) mutants. The information provided in the alternative view window serve as a reference to help the user in the identification of the faults in case they are detected by the test input in the future.

According to Romero et al. [25] their visualization helps the user to find and understand errors produced by image manipulation programs while unit test cases are executed over them. Mechanisms such as the capacity to show drawings generated by the program while the image is modified (in case the program is instrumented to do so), is enabled by a canvas. Moreover it also enables interaction mechanisms such as zoom for better analysis of image modifications.

3) Tool Interaction Usability: None of the works provide information regarding the tool's usability.

4) Analysis: None of the works mention the involvement of the target public in the visualization planning process. There are no reports about previous studies with potential users such as interviews, surveys or questionnaires to elicit requirements for the visualizations proposed. The contexts to which these proposals are applied are very specific. As a consequence, the cognitive support of such visualizations should be tailored to meet their users' specific needs and skills.

IV. Conclusions

This paper covers several aspects of unit testing visualization works that can influence the support of user's cognition.

In general, we noticed the lack of users' participation in the design and evaluation of the visualizations. Only one distinct work demonstrated the importance of end-to-end users' involvement to establish an adequate cognitive support.

We also identified underlying decision-making strategies and recognition principles that could be better explored and evaluated with the users.

None of the analyzed works have information about the usability of the visualization tools. We attribute that to two possible factors: some tools were in an early prototype stage in the time of the publication; and the scarce participation of users in the validations. Nevertheless, we note that usability is intrinsically related to the user's experience. Thus, the user's perception about how the visualization would work in his hands can influence his decision of adopting it for his practical needs.

Regarding the visualizations for structural unit testing, we identified *expressiveness* risks that could affect user's interpretation. Moreover, one experimental work demonstrated the

influence that the visualization may have over the users' behavior in the structural testing activity.

The identified gaps and opportunities show that the task of building reliable visualizations for unit testing requires concerns that are greatly focused in the users' participation and behavior. It cannot be restricted to issues like the underlying testing theory, mechanics of the tools, output results or aesthetic aspects for example.

We envision new studies that could sum to the knowledge summarized in this paper. For example, more research like Lawrence et al. [19] could help to improve the understanding of the boundaries set by the unit testing visualizations over users' behavior for different testing techniques. The benefits could extrapolate the cognitive support provided by the tools and influence the training and associated testing strategies. We also see as an urgent demand the conduction of more studies involving the target public in the visualization's construction. This applies to the initial and middle stages—when practical issues such as *expressiveness* can be evaluated—and to the final steps, where the impact of the visualizations in the users' behavior needs to be better understood.

Finally, it is important to clearly highlight that our analysis and claims do not represent a critique to the merit of any of the analyzed works. In fact, all of them are very grounded in existing theory of software testing and represented clear advances and contributions in the area. But it is important to emphasize that when dealing with cognitive support, the human-aspects need to be traversal in the research process. As a consequence, the target users need to be in the front-line, side-by-side with the researcher. Not one-step behind, or postponed to post-building steps of the tools. In the other hand, our work shows exactly the great potential we see in these tools and how their authors–and the researchers of future unit testing tools–could take advantage of our observations and put their tool one step closer from the technology transfer–which is frequently wished in our researches. Considering the current lack of such kind of study in software testing, our work evidence the fertile field of research that is open in this topic and serves to show how the community of researchers and practitioners could benefit of exploring it together.

ACKNOWLEDGMENT

We would like to thank FAPEG for the financial support provided in our research.

REFERENCES

[1] A. Walenstein, *Cognitive Support in Software Engineering Tools: A Distributed Cognition Framework*. PhD thesis, School of Computing Science, Simon Fraser University, May 2002.

[2] M. P. Prado, E. Verbeek, M. A. Storey, and A. M. R. Vincenzi, "Wap: Cognitive aspects in unit testing: The hunting game and the hunter's perspective," in *Software Reliability Engineering (ISSRE), 2015 IEEE 26th International Symposium on*, pp. 387–392, Nov 2015.

[3] S. K. Card, J. D. Mackinlay, and B. Shneiderman, eds., *Readings in Information Visualization: Using Vision to Think*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999.

[4] J. Mackinlay, "Automating the design of graphical presentations of relational information," *ACM Trans. Graph.*, vol. 5, pp. 110–141, Apr. 1986.

[5] C. Beshers and S. Feiner, "Autovisual: rule-based design of interactive multivariate visualizations," *IEEE Computer Graphics and Applications*, vol. 13, pp. 41–49, July 1993.

[6] S. M. Kosslyn, "Mental imagery," in *Visual cognition and action: Vol. 2.* (D. N. Osherson, S. M. Kosslyn, and J. M. Hollerbach, eds.), ch. 2, pp. 73–97, MIT Press, 1990.

[7] Y. Muto, K. Okano, and S. Kusumoto, "Improvement of a visualization technique for the passage rate of unit testing and static checking and its evaluation," in *Software Measurement, 2011 Joint Conference of the 21st Int'l Workshop on and 6th Int'l Conference on Software Process and Product Measurement (IWSM-MENSURA)*, pp. 279–284, Nov 2011.

[8] M. Grechanik, Q. Xie, and C. Fu, "Creating gui testing tools using accessibility technologies," in *Software Testing, Verification and Validation Workshops, 2009. ICSTW '09. International Conference on*, pp. 243–250, April 2009.

[9] Y. Muto, K. Okano, and S. Kusumoto, "A visualization technique for the passage rates of unit testing and static checking with caller-callee relationships," in *Parallel and Distributed Processing with Applications Workshops (ISPAW), 2011 Ninth IEEE International Symposium on*, pp. 336–341, May 2011.

[10] K. M. Conroy, M. Grechanik, M. Hellige, E. S. Liongosari, and Q. Xie, "Automatic test generation from gui applications for testing web services," in *2007 IEEE International Conference on Software Maintenance*, pp. 345–354, Oct 2007.

[11] M. P. Prado, A. M. R. Vincenzi, D. Nascimento, and H. A. Dantas, "Visualization supporting software testing activity: literature review artifact's page." [Online]. Available: http://vistest-implemented.rhcloud.com/index.html. [Accessed: Jul. 26, 2016].

[12] M. R. Karam and A. A. Abdallah, "Assisting in fault localization using visual programming constructs," in *Canadian Conference on Electrical and Computer Engineering, 2005.*, pp. 856–860, May 2005.

[13] P. Estefo, "Restructuring unit tests with testsurgeon," in *2012 34th International Conference on Software Engineering (ICSE)*, pp. 1632–1634, June 2012.

[14] M. Lanza and S. Ducasse, "Polymetric views - a lightweight visual approach to reverse engineering," *IEEE Transactions on Software Engineering*, vol. 29, pp. 782–795, Sept 2003.

[15] L. Inozemtseva and R. Holmes, "Coverage is not strongly correlated with test suite effectiveness," in *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, (New York, NY, USA), pp. 435–445, ACM, 2014.

[16] A. Tversky, "Elimination by aspects: A theory of choice.," *Psychological Review*, vol. 79, no. 4, pp. 281–299, 1972.

[17] J. Bertin, *Semiology of Graphics: Diagrams, Networks, Maps*. Redlands, Calif: Esri Press, 1 edition ed., Nov. 2010.

[18] W. D. Ellis, *A Source Book of Gestalt Psychology*. Psychology Press, 1999.

[19] J. Lawrence, S. Clarke, M. Burnett, and G. Rothermel, "How well do professional developers test with code coverage visualizations? an empirical study," in *2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)*, pp. 53–60, Sept 2005.

[20] A. Lienhard, T. Girba, O. Greevy, and O. Nierstrasz, "Test blueprints - exposing side effects in execution traces to support writing unit tests," in *12th European Conference on Software Maintenance and Reengineering, 2008. CSMR 2008*, pp. 83–92, 2008.

[21] H. A. Simon, "Rational choice and the structure of the environment.," *Psychological Review*, vol. 63, no. 2, pp. 129–138, 1956.

[22] H. Wang, X. Zhang, and M. Zhou, "MaVis: Feature-based defects visualization in software testing," in *2012 Spring Congress on Engineering and Technology (S-CET)*, pp. 1–4, 2012.

[23] J. A. Cottam, J. Hursey, and A. Lumsdaine, "Representing unit test data for large scale software development," in *Proceedings of the 4th ACM Symposium on Software Visualization*, SoftVis '08, (New York, NY, USA), pp. 57–66, ACM, 2008.

[24] P. Daniel and K. Y. Sim, "Dynamic fault visualization tool for fault-based testing and prioritization," in *Advanced Computer Science Applications and Technologies (ACSAT), 2012 International Conference on*, pp. 301–306, Nov 2012.

[25] J. D. Romero, M. J. Lado, A. J. Mndez, and M. P. Cota, "ITVT: An image testing and visualization tool for image processing tasks," in *5th Iberian Conference on Information Systems and Technologies*, pp. 1–6, June 2010.