VISMELLS: An Interactive Visualization for Identifying and Evaluating the Effects of Code Smells on Software Projects

Isaac de Jesus Silva*, Matheus Sampaio R. Santos*, Leandro Lopes Ramos*, Luis Paulo da S. Carvalho*
* Instituto Federal da Bahia (IFBA)
Address: Av. Amazonas, 3150, Zabelê
Cite/State: Vitória da Conquista, Bahia,
Postcode: 45.075-265
{yzaak.silva, msampaio.mail, leandrolopesramos1992, luispscarvalho}@gmail.com

***Abstract*—*Context*: Code Smells indicate potential problems in the making of software projects. If considered harmful, they must be removed from the source code. Thus, it is important to enable developers to spot the source code artifacts that have been affected by Smells. *Problem*: however, there is a lack of adequate approaches to make the identification of Code Smells easier while providing details about their occurrences. *Goals*: in order to help developers to identify Code Smells, we have developed VISMELLS, an interactive visualization. *Method*: we elicited requirements with the purpose of defining the functionalities of VISMELLS. After implementing the visualization we went through a qualitative analysis of its application with the help of professional software developers and computer science students. Lastly, we used the data obtained from the evaluation to determine if VISMELLS met the targeted requirements. *Results*: as a result, we were able to confirm that VISMELLS fulfilled its intended purpose as it received good reviews from testers.**

***Index Terms*—Code Smells, Visualization, VISMELLS**

I. Introduction

Code Smell is a concept that refers to patterns or aspects of a project that can cause problems in the development and maintenance of a system [1] [2]. Although several researchers have already studied Code Smells, demands related to their detection and visualization have not been fully satisfied yet [3] (more details in Section IV).

Oliveira *et al.* [4] claim that Code Smell identification can cause evaluation failures on a software source-code if done manually. They mention the case of inexperienced developers, who can face difficulties to confirm whether a given element is affected or not by a Code Smell.

A way to help developers verify and understand the occurrence of Code Smells in software is the use of software visualization. Koschke [5] defines Software visualization as the mapping between software artifacts and graphical representations. As a motivation for the use of visualizations, he highlights the fact that software is invisible to developers and need to take a visual form so that they can manage it. According to Carneiro [6], visualization allows the representation of a complex conceptual structure through an intuitive visual display that reveals patterns of interest to users. Thus, visualizations abstract from users the cognitive effort required to interpret patterns, and he/she just needs to use her/his perception abilities.

Considering the perception-related advantages associated with software visualizations, this paper aims to evolve the Visminer platform [7], using the Zoomable Circle Packing chart from D3.js[1] library, to create the VISMELLS visualization. D3.js has been used by many approaches as a standard library to create visualizations. For instance, Rodriguez *et al.* [8] used D3.js to create a tool to enable the visualization of refactoring opportunities in SOA applications. Specifically, the goal of VISMELLS is to present Code Smells and the values of the software metrics used in their detection in a visual manner.

In order to implement VISMELLS, we considered the following requirements, which were obtained from consulting with specialists in the academic area of development and software engineering (Section III contains more details about how we elicited the requirements):

1) Requirement 1 (**RQ1**): the visualization must allow the visual identification of classes and methods affected by Code Smells. Thus, making it easier for software developers to diagnose the loss or increase of source code quality by visualizing either the occurrence or absence of Code Smells.
2) Requirement 2 (**RQ2**): the visualization must inform the values of the metrics used in Code Smell detection, enabling a visual evaluation of their quantification.
3) Requirement 3 (**RQ3**): the visualization must allow filtering by: (a) methods and classes affected by Code Smells; and (b) project's versions throughout its evolution. By allowing filtering methods and classes, the system facilitates their identification, enabling the user to give them specific attention during the identification of Code Smells. Filtering by the version of the project allows verifying in which version the Code Smells appeared while allowing the monitoring of their development as software systems evolve.

We confirm the fulfillment of the requirements in Section VI, which presents an evaluation on the use of VISMELLS.

[1] https://d3js.org/

We based the evaluation on a survey submitted to professionals and students in the area of Information Technology and Computer Science. As a result, we found out that the visualization was well accepted by the users.

The rest of this paper is divided into the following sections: the method used in this study is explained in the Section II; details about how we elicited the requirements can be found in Section III; Section IV presents a study on papers that report works similar to this; the details about how the Vismineer platform was used to create VISMELLS are described in Section V; Section VI presents information about how VISMELLS was evaluated; threats to validity are highlighted in Section VII; and conclusion and future work are covered in Section VIII.

II. Method used to develop and evaluate VISMELLS

In this section, we highlight the activities of the method we used to carry out the development and evaluation of VISMELLS. The method is shown in Figure 1.

Requirements elicitation was the first activity we performed during VISMELLS development. This activity led to the definition of the features and functional principles that should be fulfilled by VISMELLS. We studied the state of the art during the second activity of the method to check the existence of similar previous efforts that had the same requirements. The third activity concerned the implementation of VISMELLS. After the development phase, The users evaluated VISMELLS. With this purpose in mind we conducted a qualitative analysis during the fourth activity. The fifth step is related to the compilation of the data obtained from the analysis. We obtained statistical data from the qualitativy analysis and used it to determine if VISMELLS fulfilled the expected requirements.

III. Requirements for VISMELLS

According to Kienle and Miller [9] visualization tools are more effective when they meet some functional requirements that are considered important to improve user understanding. As valid functionalities, they mention: filtering options, focus by zoom and colorization. When these requirements are met, the user can interact with the tool and they are not overloaded with excessive amount of information. This can be made possible by: (i) filtering data through the visualization, which can allow the selection of what the user wants to view and, (ii) changing the focus of the visualization in an interactive way, as, for instance, highlighting items of the visualization, making possible for users to differentiate objects by their color.

With the purpose of eliciting VISMELLS requirements, we contacted experts with professional and academic profiles comprised of university teachers, software programmers, and system analysts. Provided that the participants' experience varies between 3 to 30 years, we are sure that we considered an adequate spectrum of opinions among people who have been working for both long and short periods of time in the area of Information Systems (six experts participated in the evaluation). The requirements were introduced to them and, later on, a survey was applied to collect their opinions. The survey contained 5 answers for each requirement. Each answer corresponding to a degree of agreement relative to the relevance of the requirements. That is to say, for each requirement, they should choose an option among 5: very relevant, reasonably relevant, indifferent, little relevant, not relevant. The list below describes the requirement and the agreement/disagreement scenario:

1) Requirement 1 (**RQ1**): VISMELLS must allow the visual identification of classes and methods affected by Code Smells. Therefore, helping software engineers to diagnose the loss or increase of the quality of the source code considering the occurrence/absence of Code Smells. 5 out of 6 participants answered that this requirement is "Very Relevant" while 1 of them answered it is "Reasonably Relevant".
2) Requirement 2 (**RQ2**): VISMELLS has to inform the values of the software metrics used to detect Code Smells in relation to methods and classes affected by them, allowing a visual evaluation of metrics quantification. 4 out of 6 answered it is "Very Relevant" and two of them answered it is "Reasonably Relevant".
3) Requirement 3 (**RQ3**): VISMELLS has to allow the filtration of Code Smells by: (a) methods and classes affected by them; and (b) versions of the project through the software evolution. By allowing to filter methods and classes, VISMELLS can support the identification of Smells in a way that the user can focus his/her attention on the mitigation of the potential negative effects of Code Smells. The filtration by the version of the project allows users to check in which version a certain Code Smell appeared. 5 of 6 answered it is "Very Relevant" and 1 answered it is "Indifferent".

From the requirements evaluation we observed that the experts agreed towards their importance. Understanding how the requirements would be evaluated by software development experts was important to make us sure that VISMELLS would have valid and useful functionalities. In order to comply with these requirements, we embedded in VISMELLS the following features:

1) Coloring of selected Code Smell: when the user selects a particular Code Smell, its occurrences will be visually differentiated by a specific color (red) in the graphics (Requirement **RQ1**).
2) Filtration by: (a) version of the project (or tags): it is possible to choose a version of the software that the user wants to investigate; and (b) affected objects: the user can visualize only the objects that contain the selected Code Smell (Requirements **RQ1** and **RQ3**).
3) Highlighting and detailing by zoom: it is possible to apply the "zoom in" option, which allows the visualization of details pertaining Code Smells and the metrics used to detect them (Requirement **RQ2**).

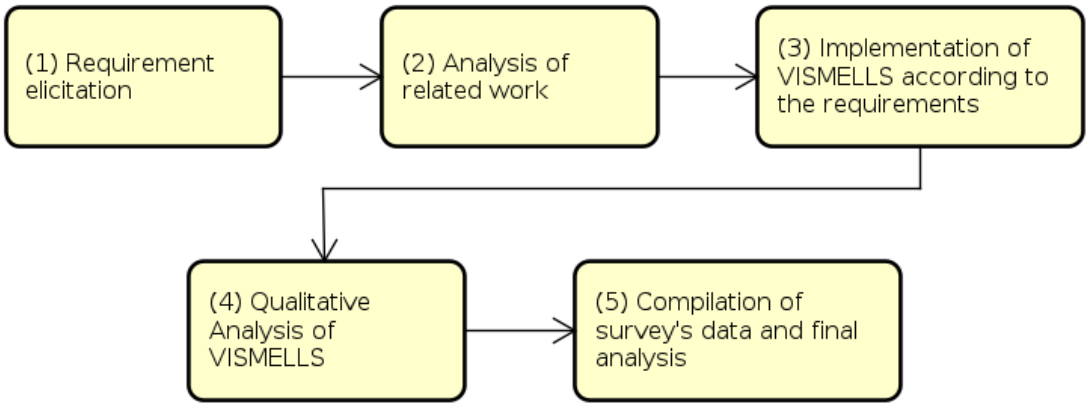

Fig. 1. Method used to develop and evaluate VISMELLS.

IV. Related work

To carry out the work described in this paper we deemed as important to check the existence of previous efforts that have focused on providing mechanisms for Code Smells visualization. The following paragraphs discuss such efforts.

Paiva [10] compares three Code Smells detection tools in order to: (a) check for an increasing in the number of smells found in software projects in the course of their evolution; and (b) evaluate the accuracy of each detection tool. Although all the three tools show the amount of detected Code Smells, it is not pointed whether they can display the values of the metrics used in the detection. Regarding evaluating the evolution of Code Smells, the application of the tools is made on each version of the evaluated software separately, in a non-interactive way.

Murphy-Hill and Black [11] present Stench Blossom plugin, that allows an interactive visualization of Code Smells in the Eclipse environment. The plugin provides three visualization perspectives: Ambient View, Active View and Explanation View. The perspectives allow the programmer to know the level of incidence, identification and technical details of Code Smells. However, the possibility of an evolutionary visualization of Code Smells is restricted to visualizing only the version that is currently in use by the programmer.

Parnin *et al.* [12] implemented a visualization catalog for code smells detection through their tool: NosePrints. Although it has been tested in several commercial and open source systems, the tool does not detail the values of the software metrics used in the detection, nor it allows filtering the visualizations by the software system's version.

Vidal *et al.* [13] present JSpIRIT, a tool that allows the identification of technical debt in the form of Code Smells. JSpIRIT allows developers to add strategies for detecting new smells, and to prioritize them through configuration settings. However, JSpIRIT does not have visualizations that relate detected smells to their metrics. As well, it does not allow filtering the detection by the version of the analyzed project.

Conceição *et al.* [14] present Crowdsmelling, a tool that relies on the concept of collective intelligence. This concept uses a code smells database, which is populated by user collaboration through a plugin integrated to the Eclipse IDE. The Code Smells are measured from a software project and the visualization is created by an integrated tool called Smelly Maps. However, the paper does not inform the possibility of visualizations between different versions of the software.

Carneiro *et al.* [15] present SourceMiner, a plugin for Eclipse IDE that has multiple visualization modes. SourceMiner makes it possible to view project data according to concerns: dependency, inheritance, security and persistence. Code Smells can be evaluated according to each concern. The paper does not tell whether a user can view the metrics applied in the detection of the Code Smells.

In Table I there is a comparative of the related works, where it is shown whether they meet or not the requirements considered in Section I.

V. Code Smells visualization using VISMELLS

A. VISMELLS architecture

The architecture, presented in Figure 2, shows that VISMELLS depends on two modules from the Visminer platform:

1) Repository Miner (RM): RM is the module responsible for mining, calculating and analyzing software projects. RM is also responsible for registering all information about metrics and Code Smells in a non-relational database, MongoDB[2]. To accomplish this, RM uses a Java library called Java Development Tools (JDT) that allows the analysis of Abstract Syntactic Trees (AST) obtained from the source code of software systems.
2) Visminer Dashboard (VD): VD is the module that supports the development of visualizations by allowing the (re)use of visualization libraries, such as D3.js. It also enables the integration of visualization libraries with web applications with the assistance of a JavaScript Framework, AngularJS [3].

We chose the Zoomable Circle Packing chart to support the visualization of Code Smells affecting classes and methods

[2]https://www.mongodb.com/
[3]https://angularjs.org/

TABLE I
REQUIREMENTS X RELATED WORK

Related work	RQ3	RQ2	RQ1
Paiva [10]	Does not meet	Not informed	Meets
Murphy-Hill and Black [11]	Does not meet	Not informed	Meets
Conceição *et al.* [14]	Does not meet	Not informed	Meets
Parnin *et al.* [12]	Does not meet	Does not meet	Meets
Vidal *et al.* [13]	Does not meet	Does not meet	Does not meet
Carneiro *et al.* [15]	Does not meet	Does not meet	Meets

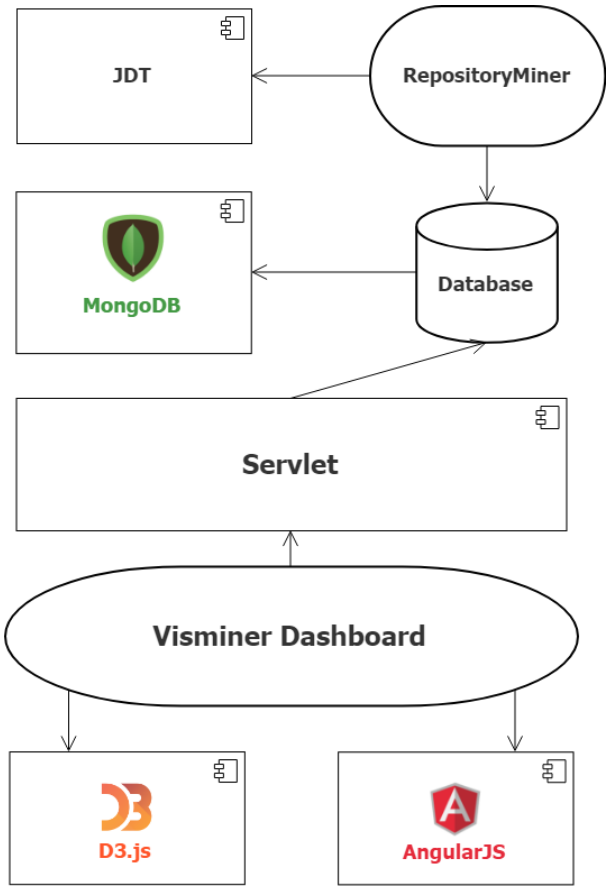

Fig. 2. VISMELLS architecture

(Requirement **RQ1**). In the visualization shown in Figure 8 (all figures referenced by this section can be found at the end of this paper), the larger circle represents a version of a software. The inner circles display packages that belongs to the software. The circles inside the package-related circles represent classes and their metrics (Requirement **RQ2**). The metrics are illustrated by the white circles. The radius of each circle is the value measured for a metric used to detect the presence of a Code Smell. The size of the white circles is proportional to values of the metrics, which allows the visual recognition of their influence on the measurement of source code artifacts. Classes that were identified as God Classes are represented by red circles. The red color is used to highlight the occurrence of smells.

One important requirement for making VISMELLS was the possibility to filter Code Smells occurrences in different versions of the analyzed projects (Requirement **RQ3**). This is possible, because RM is able to retrieve information about versioning from tags stored on version control systems. In Figure 10 it is possible to see the filtering options of VISMELLS. In the 'Tags' field (highlighted by letter A), the user can select a version from the project under analysis. In the 'Code Smells' field (highlighted by letter B) it is possible to select a smell. Available smells are: God Class, Long Method, Brain Method, Brain Class and Complex Method. The 'Only affected objects' option (highlighted by letter C), if selected, will enable the retrieval of only classes or methods affected by the selected Code Smell. If this option is not selected, the visualization will show all classes or methods, but emphasizing occurrences of smells with red colored circles.

The visualization provides understanding of the details pertaining each detected Code Smells through the zoom functionality. It is possible to browse between application levels (packages, classes, and methods) to visualize the metrics values as shown in Figure 9. Letter 'A' highlights the zoom applied upon a package. In this package, the red circles represent classes identified as God Class. By zooming into VISMELLS, the user can see which metrics was used to mine smells, as pointed out by letter 'B' letter (Requirement **RQ2**).

VI. QUALITATIVE ANALYSIS

The qualitative analysis was performed with 15 participants. Our intention was to validate VISMELLS. The participants were composed of professionals and students with expertise in Information Technology and Computer Science. They used VISMELLS to analyze a software project and answered a survey containing 5 questions[4]. The following subsections describe the steps of our qualitative analysis.

A. Analysis Setup

To carry out our qualitative analysis, we established some necessary conditions to give participants the support to understand how to use and evaluate VISMELLS. We provided:

1) A web server in which VISMELLS was installed and configured. The web server was made accessible through public IP, so that users could select versions of a software project and visualize Code Smells on their browsers.

[4]the questions are described in this section's figure's legends.

2) A video to: (i) explain the theory about Code Smells, (ii) show VISMELLS and demonstrate how to use it. The purpose of the video was to provide a theoretical material to guide the users. Available in: http://goo.gl/AmDVZO.
3) A tutorial with images and text explaining how to use VISMELLS, so that they could have a better understanding about how to evaluate it.

B. Analysis Results

After the conclusion of the qualitative analysis, we quantified the survey's data and used it to evaluate if the tool efficiently met its initial requirements (as described in Section III). We also used the results of the evaluation to identify deficiencies.

The analysis of the information presented in Figure 3 allowed us to visualize the opinion of the participants regarding the visualization of Code Smells. Out of the 15 answers obtained in the research, about 93.3% of the participants fully agreed that it was possible to visualize software projects' Code Smells with the help of VISMELLS. 6.7% participants partially agreed that the tool met the Code Smells visualization goal. Considering such results it is possible to conclude VISMELLS achieved the objective to be an assisting tool regarding the visualization of Code Smells (**RQ1**).

In the graphic presented in Figure 4, participants were asked if VISMELLS allowed them to identify when Code Smells began to appear in the project. Out of the 15 participants, 66.7% answered that they fully agreed that the tool helped them to visualize the Code Smells' origin, while 20% said that they partially agreed and 13.3% said that they were indifferent about this feature. Although it was not unanimous, such numbers showed that the participants agreed that the tool supported them in identifying the appearance of Code Smells in the source code of software projects. We are positive that participants' answers support the affirmation that VISMELLS fulfills **RQ3**.

Figure 5 shows how the participants agreed/disagreed about VISMELLS having presented the metrics of Code Smells in an explicit adequate way. Out of the 15 responses, 93.3% fully agreed that the metrics were clearly detected and 6.7% partially agreed with the affirmation. The results showed the participants' agreement that the tool allows users to visualize the values of the metrics used to detect the Code Smells (fulfilling **RQ2**).

Figure 6 shows the degree of agreement of the participants in respect with VISMELLS making the visualization of the classes and methods affected by Code Smells possible. The agreement was unanimous, with 53.3% of the participants fully agreeing and the rest agreeing partially. This indicates that the tool achieves the objective of evidencing the elements of code affected by Code Smells (**RQ3**).

Figure 7 reveals that it was difficult for the participants to understand the information visualized with the help of VISMELLS. There was a divergence of opinions among the participants, where 39% disagreed (6.7% totally disagreed and 33.3% partially disagreed) about the opinion that they had difficulties in understanding the information. 26.7% were indifferent and 33.4% (6.7% fully and 26.7% partially) answered that the information from the graphics was difficult to understand. The difficulties about the interpretation of the visualizations can be explained by the fact that some of them were not familiar with the concepts related to Code Smells and software metrics. Perhaps, a future reevaluation of VISMELLS must include a thorough enlightenment of the participants about such concepts.

VII. Threats to Validity

This section highlights the limitations and threats related to the development and evaluation of VISMELLS.

Threat to the Construct: this type of threat is associated with the relationship between theory and observation. The main threat dwells in the fact that not all specialists that participated in the qualitative analysis had knowledge about Code Smell until the day they evaluated VISMELLS. Even though they were trained about the main concepts related to Code Smells, their lack of experience on this knowledge area may have affected some answers provided by them. Consequently, this can reinforce the necessity of considering further evaluation steps, as the one suggested by [4]: the inclusion of difficulties of inexperienced developers when they have to identify smells in software projects.

Threat to the Internal Validity: internal validity concerns factors that could have influenced the results of our study. We relied on the opinions provided by six specialists to elicite VISMELLS' requirements. Later on, we used a different group of people to carry out our qualitative analysis. This meaning, the specialists who helped us to shape the requirements were not present in the evaluation. This does not reflect the usual software engineering approach of validating software systems by consulting the people who originally define a system's requirements. While we are positive that having 15 participants to evaluate VISMELLS helped us to ensure its applicability, we should have included the six original ones in the evaluation with the purpose of confirming with more accuracy that the requirements were fulfilled.

Threat to the External Validity: this threat is related to the degree to which our findings can be generalized. The fact the participants tested VISMELLS accessing it through public IP can be considered a threat. As they could evaluate it at home, this situation does not reflect a scenario in which developers work in groups to identify smells. Thus, although we have not considered such threat to be too critical to invalidate the qualitative analysis, we do not encourage extrapolating the results to fully reflect a real usage scenario.

Threat to the Conclusion: conclusion validity comprises reasons why conclusions based on an analysis may be incorrect. The evaluation of VISMELLS was based on only one example of software project, *i.e*, just one software project was mined and analyzed to detect Code Smells, PagSeguro-Java8[5].

[5]https://github.com/pagseguro/java

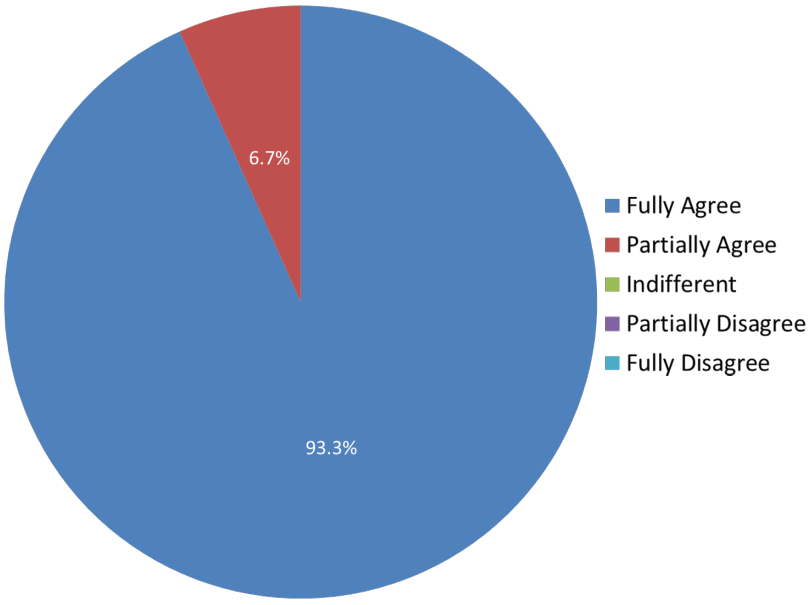

Fig. 3. Question "Has the tool made possible the visualization of Code Smells?"

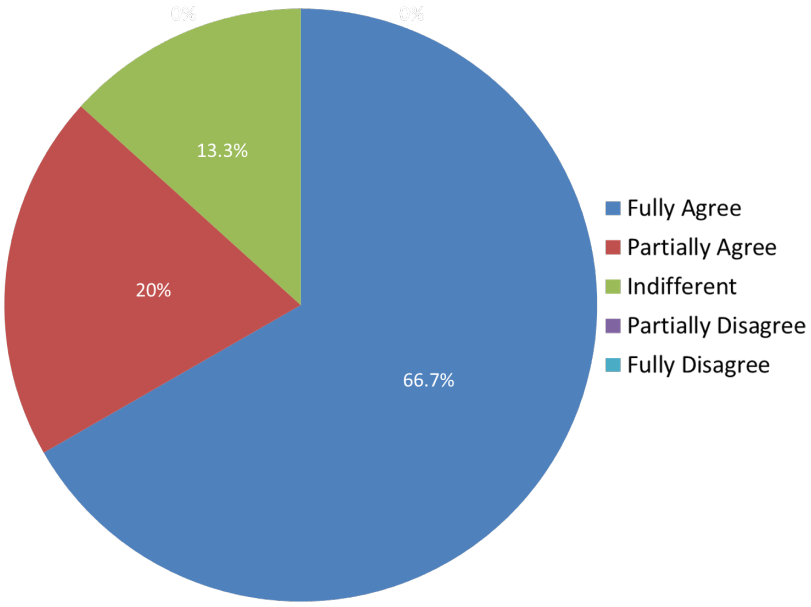

Fig. 4. Question "Has the tool made possible to identify when Code Smells appeared in a software project?"

This can cause a threat in the sense that it was not offered to participants a variability of software projects to provide them conditions to evaluate other examples of use of VISMELLS. We will look further to evaluate VISMELLS on different types of project and rerun our qualitative analysis.

VIII. Conclusion and Future Work

This paper describes the development and the validation of VISMELLS, an interactive visualization for the identification of Code Smells. VISMELLS aims at supporting interaction of users with information related to the occurrence of Code Smells in software projects. We used the visual library, D3.js, and its chart, Zoomable Circle Packing, to develop VISMELLS. The making of VISMELLS also counted on its integration with Visminer, a platform for mining and analyzing software projects. The motivation behind this work resides on the fact that there is still a demand to help software developers to identify smells. As such demand has not been fully satisfied yet an approach that uses software visualization would be appropriate to meet this need.

After defining VISMELLS' requirements and developing the visualization, we evaluated VISMELLS during a quali-

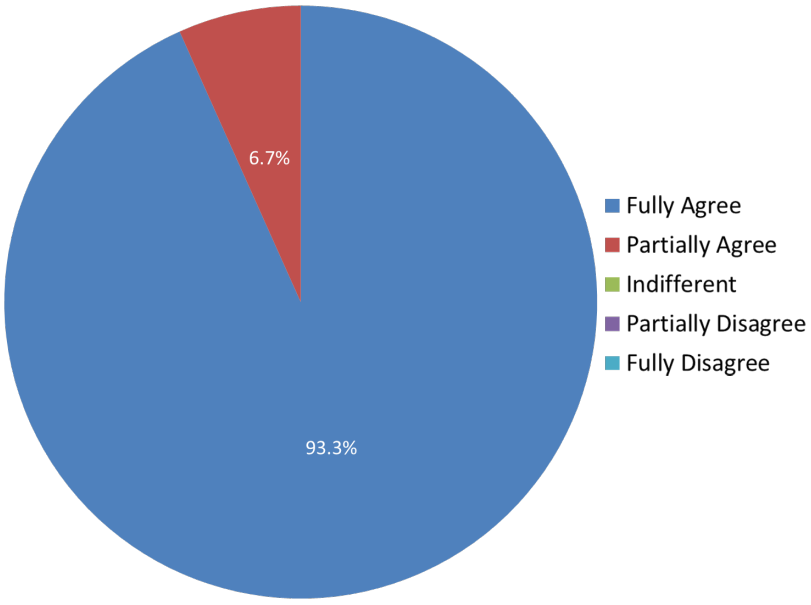

Fig. 5. Question "Has the tool clearly presented the metrics used to detect Code Smells?"

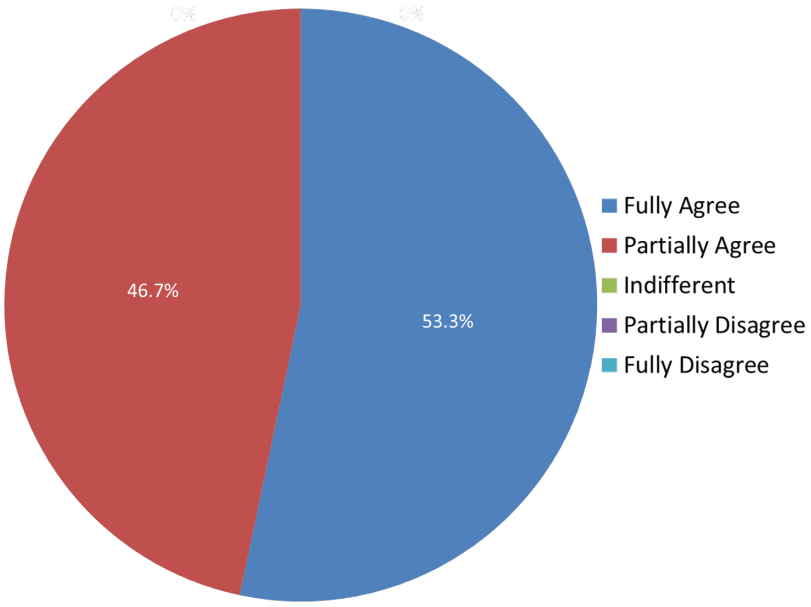

Fig. 6. Question "Has the tool highlighted the identified Code Smells, including classes and/or methods that can potentially, in the future, be affected by Code Smells?"

tative analysis. We based the analysis on tests that counted on the participation of users. At the end of the evaluation users filled answers into a survey. The analysis of the survey's data concluded that VISMELLS can facilitate the discovery of Code Smells. VISMELLS also enables the visualization of values from software metrics used to detect Code Smells. In general, the visualization fulfilled the initial goals. We highlight how each requirement was fulfilled in Table II.

As future work, we highlight the application and tests of new visualization metaphors using D3.js. We also intend to improve the display of information related to Code Smells in a way to solve the difficulties in the use of VISMELLS as pointed out by some participants during the qualitative analysis. As well, it is important to incorporate the detection of other Code Smells and new filtering options (*e.g.*, classes and methods by name).

ACKNOWLEDGMENT

We would like to thank the professors and students at the Federal Institute of Bahia for their valuable contribution in the

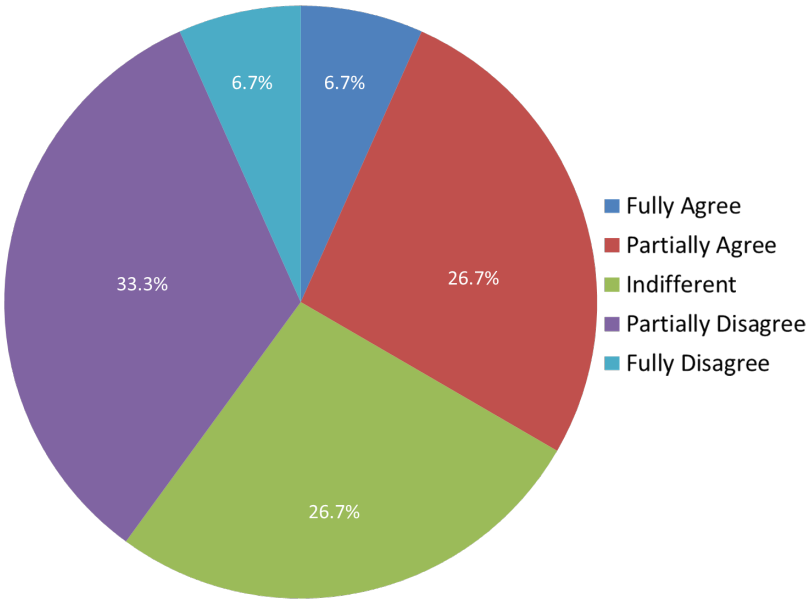

Fig. 7. Question "Were there any difficulties to understand the display of information by the visualization?"

TABLE II
REQUIREMENTS FULFILLMENT

Requirement	How we fulfilled it
RQ1	VISMELLS allows the visual identification of classes and methods affected by Code Smells through the use of the library D3.js and the Zoomable Circle Packing chart
RQ2	VISMELLS informs the values of software metrics used to detect Code Smells in methods and classes
RQ3	VISMELLS allows the filtering of methods and classes affected by Smells. It also provides ways to filter the visualization to show data of specific versions of software projects

making and validation of VISMELLS.

REFERENCES

[1] S. M. Olbrich, D. S. Cruzes, and D. I. K. Sjberg, "Are all code smells harmful? a study of god classes and brain classes in the evolution of three open source systems," in *2010 IEEE International Conference on Software Maintenance*, Sept 2010, pp. 1–10.

[2] W. Ribeiro, V. Braganholo, and L. Murta, "A study about the life cycle of code anomalies," in *2016 X Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS)*, Sept 2016, pp. 71–80.

[3] M. Steinbeck, "A new approach of visualizing code smells," *Softwaretechnik-Trends*, vol. 36, no. 2, 2016.

[4] R. Oliveira, B. Estácio, A. Garcia, S. Marczak, R. Prikladnicki, M. Kalinowski, and C. Lucena, "Identifying code smells with collaborative practices: A controlled experiment," in *2016 X Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS)*, Sept 2016, pp. 61–70.

[5] R. Koschke, "Software visualization in software maintenance, reverse engineering, and re-engineering: A research survey," *Journal of Software Maintenance*, vol. 15, no. 2, pp. 87–109, Mar. 2003. [Online]. Available: http://dx.doi.org/10.1002/smr.270

[6] G. d. F. Carneiro, "Sourceminer: Um ambiente integrado para visualização multi-perspectiva de software," Ph.D. dissertation, Federal University of Bahia (UFBA), 2013.

[7] T. S. Mendes, D. Almeida, N. Alves, R. Spínola, R. Novais, and M. Mendonca, "Visminertd – an open source tool to support the monitoring of the technical debt evolution using software visualization," in *17th International Conference on Enterprise Information Systems*. ICEIS, 2015.

[8] G. Rodriguez, A. Teyseyre, . Soria, and L. Berdun, "A visualization tool to detect refactoring opportunities in soa applications," in *2017 XLIII Latin American Computer Conference (CLEI)*, Sept 2017, pp. 1–10.

[9] H. M. Kienle and H. A. Muiller, "Requirements of software visualization tools: A literature survey," 2007. [Online]. Available: http://ieeexplore.ieee.org/document/4290693/

[10] J. P. E. F. C. S. Thanis Paiva, Amanda Damasceno, "Experimental evaluation of code smell detection tools," in *3th Workshop on Software Visualization, Evolution and Maintenance*. SBC, 2016.

[11] E. Murphy-Hill and A. P. Black, "An interactive ambient visualization for code smells," in *Proceedings of the 5th International Symposium on Software Visualization*, ser. SOFTVIS '10. New York, NY, USA: ACM, 2010, pp. 5–14. [Online]. Available: http://doi.acm.org/10.1145/1879211.1879216

[12] C. Parnin, C. Görg, and O. Nnadi, "A catalogue of lightweight visualizations to support code smell inspection," in *Proceedings of the 4th ACM Symposium on Software Visualization*, ser. SoftVis '08. New York, NY, USA: ACM, 2008, pp. 77–86. [Online]. Available: http://doi.acm.org/10.1145/1409720.1409733

[13] S. Vidal, H. Vazquez, J. A. Diaz-Pace, C. Marcos, A. Garcia, and W. Oizumi, "Jspirit: a flexible tool for the analysis of code smells," in *2015 34th International Conference of the Chilean Computer Science Society (SCCC)*, Nov 2015, pp. 1–6.

[14] Conceicao, G. d. F. Carneiro, and F. B. e. Abreu, "Streamlining code smells: Using collective intelligence and visualization," in *2014 9th International Conference on the Quality of Information and Communications Technology*, Sept 2014, pp. 306–311.

[15] G. d. F. Carneiro, M. Silva, L. Mara, E. Figueiredo, C. Sant'Anna, A. Garcia, and M. Mendonca, "Identifying code smells with multiple concern views," in *2010 Brazilian Symposium on Software Engineering*, Sept 2010, pp. 128–137.

APPENDIX
VISMELLS IN ACTION

We further provide samples from VISMELLS's visualization as it is used to analyze PagSeguro's source code.

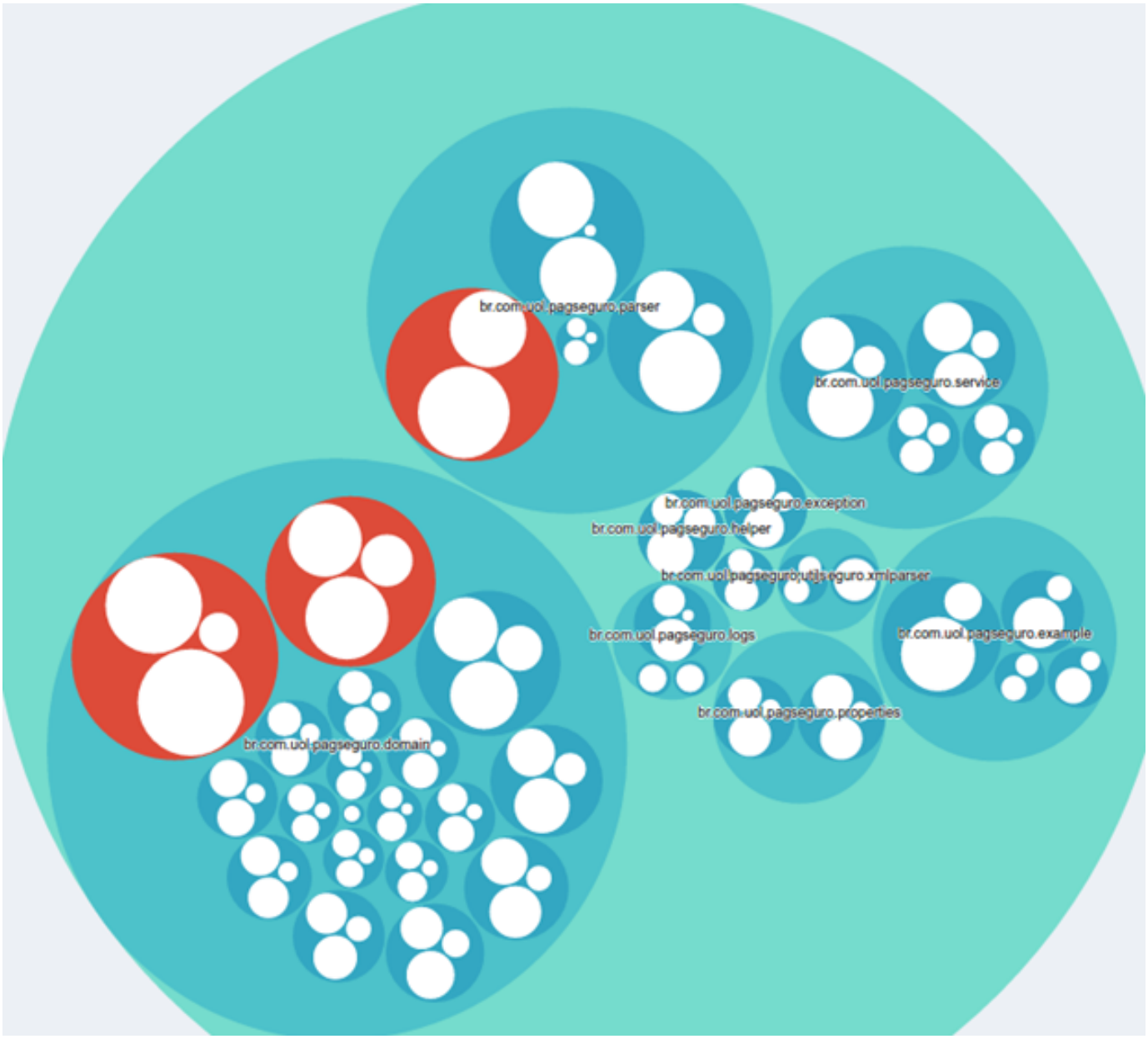

Fig. 8. Highlighting the occurrence of Code Smells.

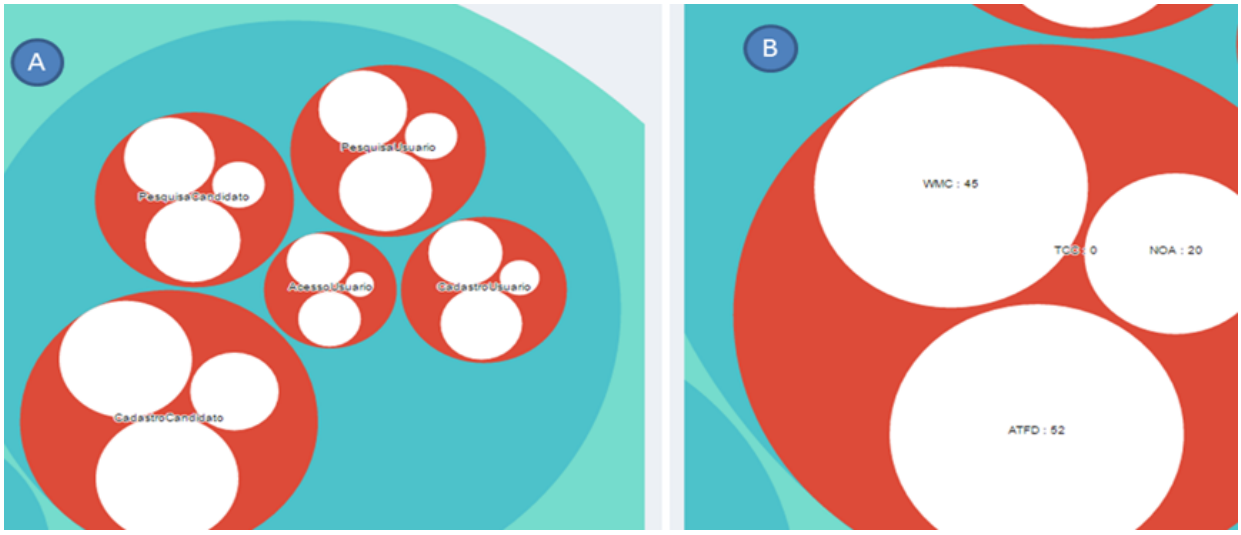

Fig. 9. Zooming into the metrics.

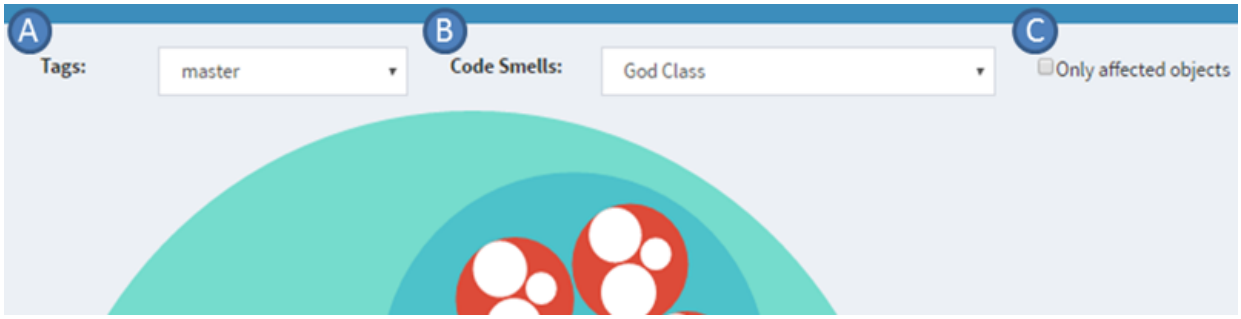

Fig. 10. Filtering options.