Scaffolding MATLAB and Octave Software Comprehension Through Visualization

Ivan de M. Lessa, Glauco de F. Carneiro
Universidade Salvador (UNIFACS)
Salvador/Bahia, Brazil
ivan.lessa@gmail.com,
glauco.carneiro@unifacs.br

Miguel P. Monteiro
Universidade Nova de Lisboa (UNL)
NOVA LINCS
Lisbon, Portugal
mtpm@fct.unl.pt

Fernando Brito e Abreu
Instituto Universitário de Lisboa
(ISCTE-IUL)
Lisbon, Portugal
fba@iscte-iul.pt

***Abstract*— Multiple view interactive environments (MVIEs) provide visual resources to support the comprehension of a specific domain dataset. For any domain, different views can be selected and configured in a real time fashion to be better adjusted to the user needs. This paper focuses on the use of a MVIE called *OctMiner* to support the comprehension of MATLAB and GNU/Octave programs. The authors conducted a case study to characterize the use of *OctMiner* in the context of comprehension activities. Results provide preliminary evidence of the effectiveness of *OctMiner* to support the comprehension of programs written in MATLAB and Octave.**

Keywords – software visualization; MATLAB/Octave; software comprehension.

I. INTRODUCTION

Multiple view interactive environments (MVIE) provide resources to support data analyses and unveiling information that otherwise would remain unnoticed [1][4]. This work is focused on MATLAB [8] and Octave [11] programs, following reports in the literature that indicate a lack of support for the comprehension of programs coded in these languages. We contribute to fill this gap by implementing a MVIE named *OctMiner*. Following previous research on this topic [2][9], we conducted a case study using *OctMiner* to support the comprehension of MATLAB/Octave programs, which aims at characterizing the MVIE support to identify crosscutting concerns.

This paper is structured as follows: section II describes key functionalities of *OctMiner* and its architecture; section III presents two case studies to characterize *OctMiner* as a means to support MATLAB/Octave program comprehension; section IV proposes a set of usage strategies to be performed with OctMiner for comprehension purposes. Finally, section V presents the final considerations and outlines opportunities for future work.

II. MULTIPLE VIEW INTERACTIVE ENVIRONMENTS

Visualization is a means of providing perceivable cues to several aspects of the data under analysis to reveal patterns and behaviors that would otherwise remain unhighlighted and unnoticed [13]. Card et al. [1] proposed a well-known reference model for information visualization. According to them, the creation of views goes through a sequence of successive steps: pre-processing and data transformations, visual mapping and view creation. Carneiro and Mendonça [3] extended this model to adapt it to the context of MVIEs. Figure 1 shows the extended model, emphasizing that the visualization process is highly interactive. Moreover, it enables the combined use of resources of a multiple view interactive environment. The process starts with original (raw) data obtained from a repository that undergoes a set of transformations to be organized into data structures suitable for information exploration. This process is called *data transformation* [3]. Next, the data structures are used to assemble visual data structures. Those structures organize data properties and visual information properties in ways that facilitate the construction of visual metaphors. This step defines the mapping from real attributes – which are derived from the data properties, software attributes, in our case – to visual attributes such as shapes, colors and positions on the screen. This process is called visual mapping [3]. It is important to highlight that these activities do not deal with rendering, but rather with building suitable data structures from which the views can be easily computed and rendered. The final step, presented in Figure 1, is the view transformation, aimed at drawing the information on the screen to produce the views. In this step, a specific visual scene is actually rendered on the computer screen [3].

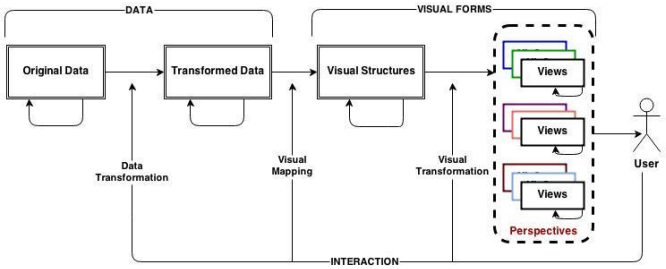

Figure 1. An Extended Reference Model for MVIEs [3]

Nunes et al. [10] proposed a toolkit implemented as a Java Eclipse plugin from which MVIEs could be developed. The plugin provides a basic structure that allows the creation and inclusion of new resources and functionalities to develop MVIEs. Figure 2 presents the way the toolkit was used and extended by other plugins to comprise the SourceMiner MVIE. This MVIE was originally developed to support the comprehension of Java source code. As can be seen in the figure, the extension points of the toolkit.aimv plugin enable the inclusion of new plugins to the MVIE. Each of the extension points conveyed provides an interface with methods and their respective signatures. In the case of *OctMiner*, we needed to access and transform raw data – the Abstract Syntax Tree (AST) of MATLAB/Octave programs – to a format compatible with the

DOI reference number: 10.18293/SEKE2015-073

visual data structure. According to the extended reference model for MVIEs, this is a requirement to feed the views.

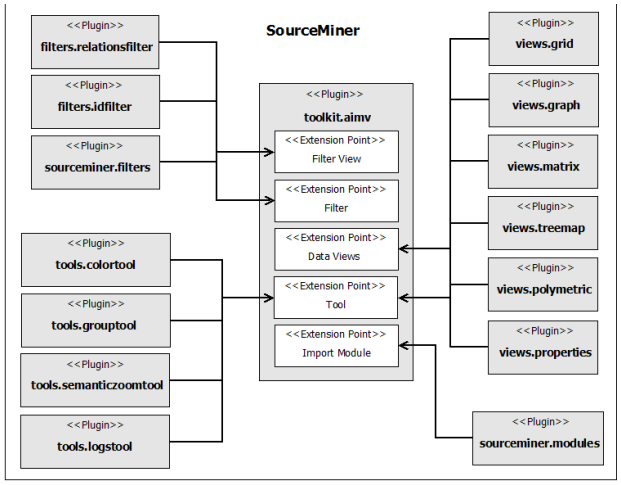

Figure 2. The MVIE SourceMiner [10]

Figure 2 presents a set of plugins that comprise the SourceMiner MVIE. The following guides are available to help MVIE developers: (1) Data Transformation: to extend the plugin Import Module to implement the plugin *sourceminer.modules*; (2) Creating and Applying Filters to extend the plugins Filter and Filter View; (3) Creating Tools to extend the plugin Tools; (4) Creating Views to extend the plugins Data Views and Tools. These guides are available at [14].

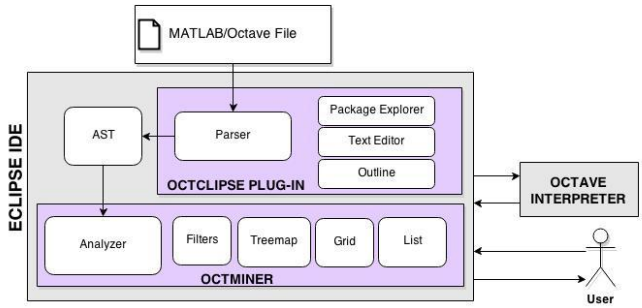

Figure 3. *OctMiner* Architectural Overview [7]

The goal of the toolkit is to provide an infrastructure to develop MVIEs for different domains. The domain targeted in this paper comprises programs written in MATLAB/Octave.

A. THE MATLAB AND OCTAVE PROGRAM LANGUAGES

MATLAB is an interpreted language very popular among students and researchers of physics, biomedical engineering and related areas. It is not uncommon that a young engineer is fluent in using MATLAB, but hardly familiar with C, and even less of Fortran [5][15]. MATLAB has been used to teach linear algebra, numerical analysis, and statistics. Since the MATLAB language is proprietary, a similar language, named Octave was developed, and is distributed under the terms of the GNU General Public License. It was originally conceived in 1988 to be a companion programming language for an undergraduate-level textbook on chemical reactor design. Due to the similarities between these languages, it is possible to interpret MATLAB programs in the interpreter of the GNU/Octave with no major problems. The main differences among the two languages are as follows: i) Some similar routines can have different names in each language; ii) Comments in MATLAB are written after "%" while in Octave you can use both "%" and "#"; iii) In MATLAB the control blocks (while, if and for) as well as the functions delimiter all finish with "*end*" while in Octave you can also use "*endwhile*", "*endif*", "*endfor*" and "*endfunction*" respectively; iv) In MATLAB the not equal to operator is "~=" while in Octave "!=" is also valid; v) MATLAB does not accept increment operators such as "++" and "—", while Octave accepts them.

B. THE AIMV OCTMINER

The main motivation for representing concerns manifested in MATLAB/Octave code in a MVIE is the enhancement of the comprehension activities. The plugin structure supporting the MVIE toolkit is the same as presented in Figure 2. The main difference is that in this case the focus is on MATLAB/Octave rather than Java. Figure 3 depicts the main four elements of *OctMiner*: the Eclipse IDE RAP/RCP (Rich Clients and Rich Ajax Applications), the *Octclipse* plugin, the Octave interpreter and the MVIE toolkit proposed in [10]. The Eclipse IDE enables its extension through the use of plugins. The MVIE toolkit does this to provide its functionalities as well as enabling the tailoring of the MVIE tailoring for the analysis of data from different domains, e.g., the data gathered from MATLAB/Octave programs.

We implemented an Analyzer module as presented in Figure 3, which is analogous to *sourceminer.modules* – see Figure 2. It is an extension of the Import Module, whose goal is to import and convert data from the original data repository to be represented in the multiple views. The *Octclipse* plugin also provides an Octave development environment built on top of Eclipse's Dynamic Languages Toolkit. This environment enables programmers to create Octave scripts (*.m files), edit them in a multi featured text editor, run the Octave interpreter and see results displayed in the IDE's console. *OctMiner* is available at [14].

To provide a short illustration of the visualization capabilities of *OctMiner*, Figure 4 shows a typical visualization scenario. Part A is the Project Explorer, presenting all the repository files; Part B is the Outline, showing the functions and variables of a given file, when it is selected in the Project Explorer. Part C provides editing access to the routine's code. Part F is a filter dashboard. Parts D, E and G are views corresponding to several different visualization metaphors. For instance, the Treemap view (G) provides panoramic view, e.g., of how names of routines are distributed in the file repository. Colours represent different concerns (be they crosscutting or no). We use the term "token" to refer to routine names from the MATLAB/Octave systems. The List view (E) presents a list of the files from the repository. The Grid view (D) is be used to identify the tokens

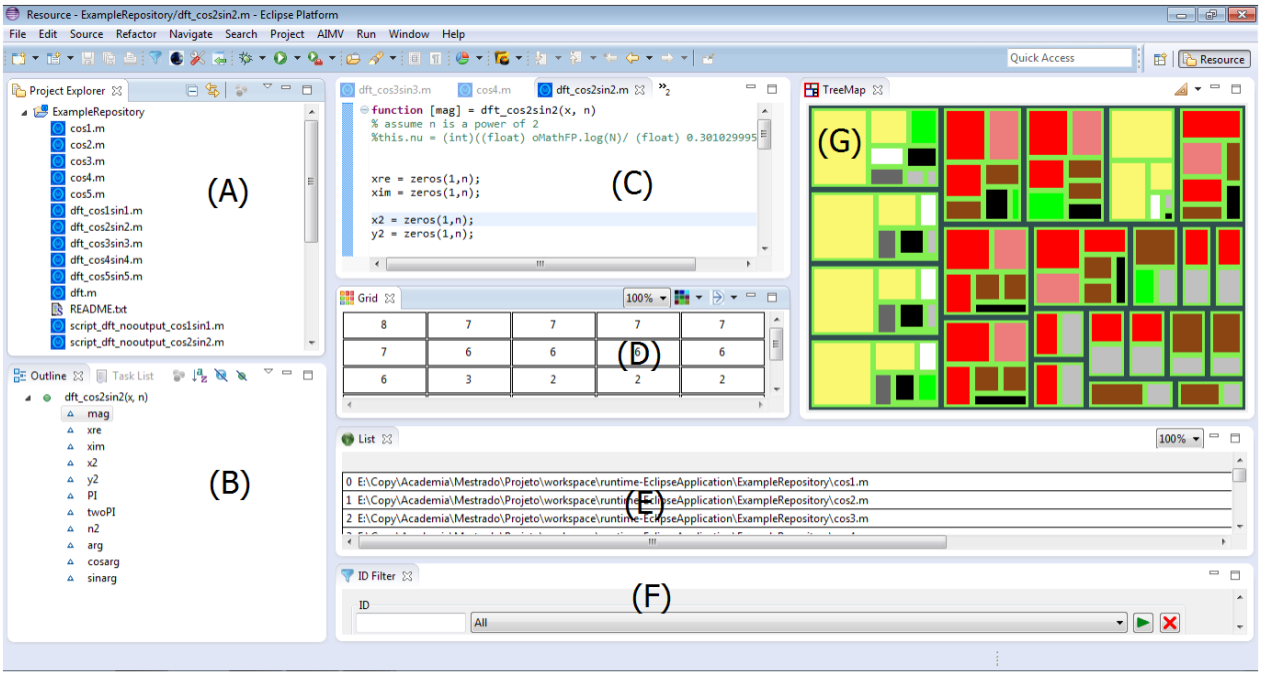

Figure 4. A Typical Scenario of *OctMiner* Use in the Eclipse IDE [7]

used in the repository along with several different metrics, e.g., number of occurrence of each token in each file or in the whole repository. This view can also be presented in several orderings, depending on what is convenient. Full details on the visualizations are provided in our ITNG paper [7].

III. Comprehension Activities with OctMiner

This section presents a case study to characterize the use of *OctMiner* in comprehension activities. In it, we investigate the following question: to which extent *OctMiner* provides effective support to identify potential symptoms of crosscutting concerns in MATLAB programs? In the study, we analyze 22 MATLAB image processing routines. The goal is the identification of the dual symptoms of scattering and tangling in the routines, as supported by *OctMiner*. *Scattering* [12] is the degree to which a concern is spread over different modules or other units of decomposition. *Tangling* [16] is the degree to which concerns are intertwined to each other in the same routines. Both scattering and tangling are indicators of the presence of crosscuting concerns in program code.

The case study explores the potential of tokens to be indicators of the scattering and tangling symptoms. The approach is as follows: sets of tokens can be associated to a given concern, which ideally would be modularized into its own file, with no additional concerns. When the concern is not modularized, its code is scattered across multiple files and its associated tokens are found in such files – an indicator of scattering. Often, such files also betray the presence of tokens categorized under multiple concerns – an indicator of tangling.

To explore the aforementioned approach, participants performed the following activities: i) Identify tokens most commonly used in the 22 routines; ii) Characterize the localization among files of the most commonly used tokens to assess the symptoms of scattering; iii) Characterize the relationship between the most commonly used tokens and other tokens in the files to assess the symptoms of tangling; iv) Determine the category (concern) to which the most commonly used tokens belong; v) Using the category of each token, identify the main functionalities (concerns) of the program. Using this approach, it was possible to identify the top most commonly used tokens in the analyzed routines and that this same tokens presented evidences of scattering. This study was the starting point for the use of *OctMiner* in comprehension activities.

We identified the following limitations in this study: considering that the routines were already analyzed by *OctMiner*, any new modification in the original routines will not be reflected in the views until a new analysis is performed to obtain these modification from the repository. In addition, the user can only select the predefined color in *OctMiner*. It is also not possible to define new colors in this version of *OctMiner*. The need to configure the XML file with the tokens is also a limitation. To overcome it, we intend to provide a XML file with a large number of MATLAB and Octave functions and their respective categories.

We recognize that *OctMiner* may not be able to provide support for all kinds of comprehension needs. To better characterize and validate its range of applicability, we plan additional studies (see section V). Another potential threat to validity is that both design and execution of the study were performed by the same person. To overcome this issue, further independent experiments will be carried out to compare results more thoroughly.

IV. PRELIMINARY STRATEGY BASED ON *OCTMINER*

Results from this case study enable us to propose a preliminary usage strategy based on *OctMiner* for comprehension purposes. The strategy includes a comprehension question as its starting point, which drives subsequent steps. The question is related to tangling and scattering, using a set of tokens from programs of a repository as a basis. Table 3 presents the steps proposed from evidences collected from this case study.

Table 3. A Proposed Set of Usage Strategies

Suggested Steps
1 - **Select a question:** the programmer needs to identify an issue relevant for his daily activities. Answers to the question should be available considering that the routines used in the code should be registered in the *OctMiner* configuration file.
2 – Identify a target routine: it should be the routine that plays a relevant role in the code of the primary solution to the selected question.
3 – Locate repositories that use the target routine: since *OctMiner* aims at assisting the comprehension of a given target routine, it is desirable that routines using the target routine provide good examples and be the subject of analysis.
4 – Identify the routines and their respective categories available in the official documentation: alternative routines used in the repository selected in Item 3 must also be identified. MATLAB and Octave routines are categorized in the official language sites of MATLAB and Octave.
5 – Register the target routine as well as other routines from the repository in the *OctMiner* configuration file: the routines should be registered in *OctMiner* configuration file using their specific group, identified according to Item 4.
6 – Create a To-Do list for identification through visualization: activities that the user must perform should be described so that the study is conducted as well as possible within *OctMiner*.
7 – Implementation of the proposed activities: the user must run *OctMiner* according to the activities set out in Item 6.
8 - Answer the original question: to prove the effectiveness of the tool, the user should be able to answer the question that started the process in Item 1.

V. CONCLUSIONS AND FUTURE WORK

This paper presents the following contributions: a) the provision of an environment called *OctMiner* for the comprehension of MATLAB/Octave routines supported by multiple views; b) Evidences of the effectiveness of OctMiner to support the identification of symptoms of code tangling and code scattering as discussed in the study presented at section III; c) the initial version of a sequence of steps for a strategy for the usage of *OctMiner* for comprehension purposes.

A previous paper by the same authors describing the architecture of *OctMiner* along with an illustrative example of its main functionalities in a real scenario of program comprehension, was presented at ITNG'2015 [7]. An extended version of the present paper, where the validation case studies are described in detail and additional information on the proposal is provided, will appear in the proceedings of ICCSA'2015 in Canada.

We will soon conduct a new version of a more detailed study, based on answers posted at popular question-and-answers sites (e.g., *StackOverflow*). We are planning research questions to assess the extent to which *OctMiner* provides effective support to clarify programmer´s issues. We believe *OctMiner* can help programmers in understanding the context of use of a routine through *OctMiner*'s visualizations. Our goal is to gather evidence of the effectiveness of *OctMiner* in supporting acquisition of insights by means of the visualization of target routines. We will base the next study on routines referred in posts from question-and-answers sites.

REFERENCES

[1] Card, S. K., Mackinlay, J. and Shneiderman, B. Readings in Information Visualization Using Vision to Think. San Francisco, CA, Morgan Kaufmann, 1999.

[2] Cardoso, J.; Fernandes, J; Monteiro, M.; Carvalho, T; Nobre, R. Enriching MATLAB with aspect-oriented features for developing embedded systems. Journal of Systems Architecture 59 (2013) p. 412–428.

[3] Carneiro, G.; Mendonça, M.. SourceMiner: Towards an Extensible Multi-perspective Software Visualization Environment. In: Slimane Hammoudi;José Cordeiro;Leszek A. Maciaszek;Joaquim Filipe. (Org.). Enterprise Information Systems. 1ed.: Springer International Publishing, 2014, v. 190, p. 242-263.

[4] Carneiro, G., Silva, M., Mara, L., Figueiredo, E., Sant'Anna, C., Garcia, A., Mendonc¸a, M., 2010. Identifying code smells with multiple concern views. In: XXIV BrazilianSymp. on Software Engineering (SBES 2010), IEEE Comp. Soc., Washington, DC, USA, pp. 128–137.

[5] Chaves, J.; Nehrbass, J.; Guilfoos, B.; Gardiner, J.; Ahalt, S.; Krishnamurthy, A.; Unpingco, J., Chalker, A.; Warnock, A.; Samsi, S. Octave and Python: High-Level Scripting Languages Productivity and Performance Evaluation. In Proc. of the HPCMP Users Group Conference (HPCMP-UGC '06).

[6] Data Explorer - StackExchange. Available at http://data.stackexchange.com/.

[7] Lessa, I.; Carneiro, G.; Monteiro, M.; Abreu, F. A Multiple View Interactive Environment to Support MATLAB and GNU/Octave Program Comprehension. In: International Conference on Information Technology:New Generations (ITNG), 2015, Las Vegas/EUA.

[8] MATLAB Programming Language. Available at www.mathworks.com/products/matlab.

[9] Monteiro, M.; Cardoso, J.; Posea, S. Identification and characterization of crosscutting concerns in MATLAB systems. In Conference on Compilers, Programming Languages, Related Technologies and Applications (CoRTA 2010), Braga, Portugal (pp. 9-10).

[10] Nunes, A.; Carneiro, G.; David, J. Towards the Development of a Framework for Multiple View Interactive Enviironments. In: International Conference on Information Technology:New Generations (ITNG), 2014, Las Vegas/EUA. p. 23-30.

[11] Octave Programming Language. Available at www.gnu.org/software/octave/.

[12] Robillard, M; Murphy, G. Representing Concerns in Source Code. ACM TOSEM, 2007.

[13] Spence, R. Information Visualization: Design for Interaction (2nd Edition). 2. ed.Prentice Hall, 2007.

[14] SourceMiner Website. Available at www.sourceminer.org/octminer

[15] Stenroos, M.; Mäntynen, V.; Nenonen, J. A MATLAB library for solving quasi-static volume conduction problems using the boundary element method. - Computer methods and programs in biomedicine, 2007.

[16] Tarr, P.; Ossher, H.; Harrison, W.; Jr., N. Degrees of Separation: Multi-Dimensional Separation of Concerns. ICSE, 1999.