Repositório ISCTE-IUL

On the Use of a Multiple View Interactive Environment for MATLAB and Octave Program Comprehension

Ivan M.Lessa[1], Glauco de F.Carneiro[1], Miguel P.Monteiro[2], Fernando Brito e Abreu[3]

[1]Universidade Salvador (UNIFACS), Salvador/Bahia, Brazil
ivan.lessa@gmail.com, glauco.carneiro@unifacs.br
[2]Universidade Nova de Lisboa (UNL), NOVA LINCS, Lisbon, Portugal
mtpm@fct.unl.pt
[3]Instituto Universitário de Lisboa (ISCTE-IUL), ISTAR-IUL, Lisbon, Portugal
fba@iscte-iul.pt

Abstract. MATLAB or GNU/Octave programs can become very large and complex and therefore difficult to understand and maintain. The objective of this paper is presenting an approach to mitigate this problem, based upon a multiple view interactive environment (MVIE) called *OctMiner*. The latter provides visual resources to support program comprehension, namely the selection and configuration of several views to meet developers' needs. For validation purposes, the authors conducted two case studies to characterize the use of *OctMiner* in the context of software comprehension activities. The results provided initial evidences of its effectiveness to support the comprehension of programs written in the aforementioned languages.

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

1 Introduction

MATLAB [1] and its open source "clone" Octave [2] are high-level programming languages and development environments that are widely used for rapid prototyping and simulation of scientific applications. As those applications grow in size and complexity, they face the usual maintenance challenges that are common in the so-called "legacy systems" [1]. Maintainability depends on our ability to understand programs, what lead to the creation of a *Program Comprehension* scientific community[3].

By reviewing the available literature, we found evidence of a lack of support for the comprehension of programs coded in MATLAB and Octave, as described in a following section. We tackled this research opportunity by implementing a multiple view interactive environment (MVIE) named *OctMiner*. MVIEs provide resources to

---

[1] a registered trademark of *The MathWorks© (http://www.mathworks.com/products/matlab)*

[2] see *http://www.gnu.org/software/octave/*

[3] see *http://www.program-comprehension.org/*

support data analyses and unveiling information that otherwise would remain unnoticed [2] [3]. To validate *OctMiner* effectiveness, we conducted two case studies using the tool to support the comprehension of MATLAB/Octave programs. The first study aimed at characterizing the MVIE support to identify crosscutting concerns following previous research on this issue [4] [5]. The second study focused on analyzing to which extent *OctMiner* can help programmers to understand the solutions proposed in the *StackOverflow* community [4], a popular question-and-answer site for professional programmers, regarding MATLAB and Octave problems.

This paper is structured as follows: section 2 summarizes the main concepts of the MATLAB/Octave programming languages and describes the key functionalities of *OctMiner* and its architecture; section 3 presents two case studies to exemplify how *OctMiner* can support MATLAB/Octave program comprehension; section 4 proposes a set of usage strategies to be performed with *OctMiner* for comprehension purposes; section 5 reviews related work; finally, section 6 presents the final considerations and outlines opportunities for future work.

2   Multiple View Interactive Environments

Visualization provides perceivable cues to several aspects of the data under analysis to reveal patterns and behaviors that would otherwise remain "under the radar" [6]. Card et al. [2] 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 [7] extended this model to adapt it to the context of MVIEs. The extended model is portrayed in Fig. 1 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, which is then organized into data structures suitable for information exploration. This process is called *data transformation* [3]. Next, the aforementioned 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 rendered. The final step, presented in Fig. 1, is the *visual 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].

Nunes et al. [8] proposed a toolkit implemented as a Java Eclipse plugin from which MVIEs could be developed. The plugin provides a basic structure that allows

---

[4]   see *http://www.stackoverflow.com*

the creation and inclusion of new resources and functionalities to develop MVIEs. Fig. 2 presents the way the toolkit was used and extended by other plugins to reify the *SourceMiner* MVIE. This MVIE was originally developed to support the comprehension of Java source code. As can be seen in Fig. 2, the extension points of the *toolkit.aimv* plugin enable the inclusion of new plugins to the MVIE. Each of the conveyed extension points 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 visual data structure. According to the extended reference model for MVIEs, this is a requirement to feed the views. Fig. 2 presents a set of plugins that comprise the *SourceMiner* MVIE.
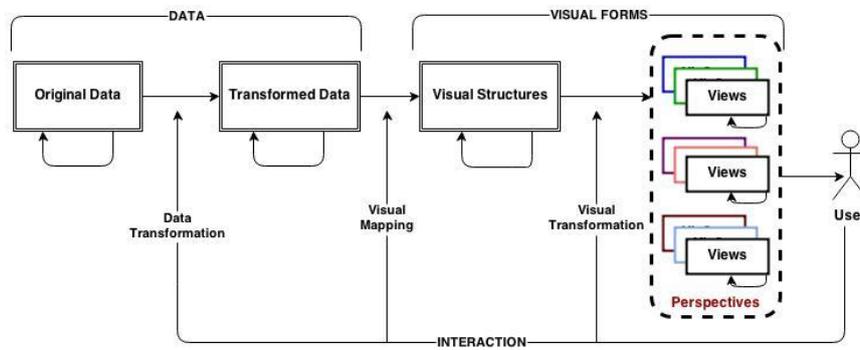

Fig. 1. An extended reference model for MVIEs [3]
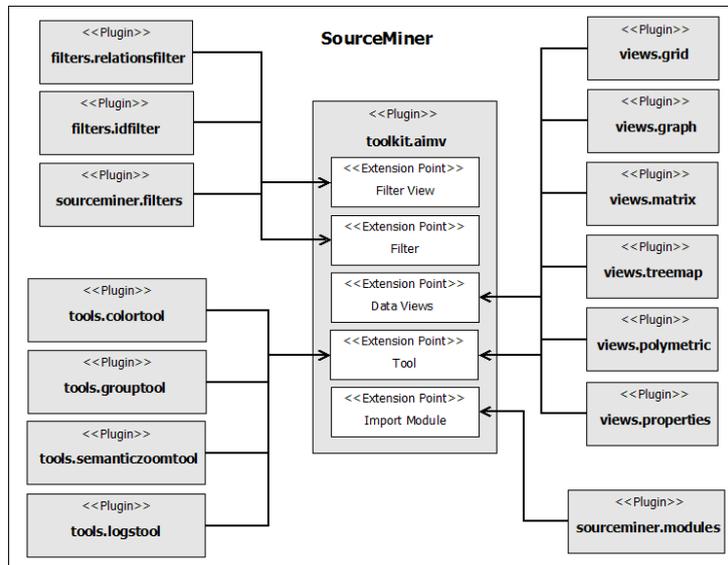

Fig. 2. The *SourceMiner* MVIE [8]

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. The application of the aforementioned toolkit to this domain was reified through *OctMiner*, whose architectural overview is depicted in Fig. 3. The *Grid* and *Treemap* views were provided by the MVIE. On the other hand, the *List* view, the *Filters* and the *Analyzer* were extended from the MVIE specifically for *OctMiner* usage.
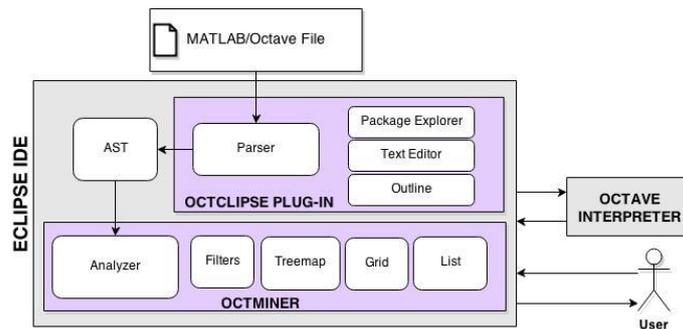
Fig. 3. *OctMiner* architectural overview [9]

2.1 The MATLAB and OCTAVE Programming 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 with Fortran [10] [11].

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 between the two languages are the following:

i) some similar functions 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 *"end-function"* 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.

2.2 The *OctMiner* MVIE

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 Fig. 2. The main difference is that in this case the focus is on MATLAB/Octave rather than Java. Fig. 3 depicts the main four elements of *OctMiner*: the Eclipse IDE RAP/RCP (Remote Application Platform / Rich Client Platform) , the *Octclipse* plugin[5], the Octave interpreter and the MVIE toolkit proposed in [8]. 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 for the analysis of data from different domains, e.g., the data gathered from MATLAB/Octave programs. We implemented an *Analyzer* module, as conveyed in Fig. 3, which is analogous to *sourceminer.modules* in Fig. 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*[6]. This environment enables programmers to create Octave scripts (**.m* files), edit them in a multi featured text editor, run the Octave interpreter and see the results displayed in the IDE's console. *OctMiner* is freely available for download[7].

3 Comprehension Activities with *OctMiner*

This section presents two studies to characterize the use of *OctMiner* in software comprehension activities.

3.1 The First Study

The first study investigated the following question: "to which extent *OctMiner* provides effective support to identify potential symptoms of crosscutting concerns in MATLAB programs?" To answer this question we analyzed 22 MATLAB image-processing routines with *OctMiner*, to identify the presence of scattering and tangling. Scattering [12] is the degree to which a concern is spread over different modules or other units of decomposition. Tangling [13] is the degree to which concerns are intertwined to each other in the same module. Both scattering and tangling are indicators of the presence of crosscutting concerns. The basic units of decomposition (modules) in MATLAB or Octave are *functions* and *files*. For cohesion sake, a file usually contains a set of related functions.

The term "token", to be used hereinafter, represents a function name from the MATLAB/Octave systems. This study considers that the distribution of the occurrence of these tokens can be used as an indicator of scattering and tangling

---

[5] see *http://sourceforge.net/projects/octclipse/*

[6] see *http://eclipse.org/dltk/*

[7] see *http://www.sourceminer.org/octminer*

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 above 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.

This approach allowed identifying the top most commonly used tokens in the analyzed routines. These tokens presented evidences of scattering. This study was just a pilot-test in using *OctMiner* in comprehension activities and it allowed us to identify a set of improvements, which were added to *OctMiner* before the next study took place.

3.2   The Second Study

The second study had the following research question, based on answers posted at *StackOverflow*: "to which extent *OctMiner* provides effective support to clarify programmer´s doubts"? The main goal of this study was then to show *OctMiner's* effectiveness in supporting the visualization of target functions as the ones reported at *StackOverflow*. In other words, we hypothesized that *OctMiner* can help programmers to understand the context of use of a function in routines that serves as examples supported by the available views. The authors searched for the top questions about the two selected programming languages and their corresponding best answers. For this purpose, the search used the *StackExchange Data Explorer* tool[8]. Applying the following query, using the mentioned tool, we obtained as a result the top 200 questions related with the keywords "MATLAB" and "Octave":

```
SELECT TOP 200 a.creationdate, q.owneruserid, q.title
FROM users u, posts a, (SELECT id, owneruserid, title, tags,
creationdate FROM posts WHERE tags LIKE '%<KEY_WORD>%') q
WHERE q.id = a.parentid and a.owneruserid=u.id
ORDER BY a.creationdate desc
```

We classified the questions in the following categories:

(a) programming language basic issues – 146 questions;

(b) common mistakes in MATLAB and Octave – 51 questions;

(c) using functions to perform specific work such as numerical calculation and image processing – 98 questions;

(d) using functions to plot data on the screen – 69 questions;

(e) questions that do not fit into any of the previous categories – 56 questions.

As can be seen, category (a) has the greater number of questions, which indicates a lack of basic knowledge of the two languages. We considered this fact as the start point to select the following question: "I want to create a vector without the number 1". The answer with most votes was "I would use *setdiff*". The answer was illustrated as follows "*setdiff(-5:5,1)*".

Configuring *OctMiner* to Answer the Question. The participant configured *OctMiner* according to the goal of the second study. The configuration consists of editing a XML file as follows. *<GroupName>* defines the group to which the function belongs to, whereas *<function>* contains the list of functions to be represented in the views.

```
<group>
  <GroupName title="GroupName" color="color">
    <function>;Function1;Function2;</function>
  </GroupName>
</group>
```

Table 1. Categories and their Colors in *OctMiner*

Category	Color Name	Color
Array and Matrix Creation and Concatenations	Concrete	
Set Operations	Green	
Indexing	MethodBorder	
Parse Strings	Size	
Logical Operations	Blue	
Advanced Software Development	Class	
Mathematics	Abstract	

We selected 22 MATLAB routines to illustrate the use of the *setdiff* function, the target function of the selected doubt. The authors selected these routines searching the *StackOverFlow* repository using the string "MATLAB *setdiff*". The authors also registered *settdiff* function and all the other functions identified in the 22 routines in the *OctMiner* configuration XML file. More details regarding the XML file can be obtained at *OctMiner* page. Table 1 conveys the categories and their respective colors to be presented by *OctMiner*.

Focusing *OctMiner* on the setdiff Function. Based on our experience, we proposed a set of steps in Table 2 focusing on the comprehension of *setdiff* function supported by *OctMiner* to clarify a real doubt registered by a programmer at *StackOverFlow*. In the next section, we explain how these steps were executed.

Table 2. Proposed Steps in *OctMiner*

STEPS
Select a question: to clarify a doubt.
Identify the *setdiff* function in the repository: the programmer should configure *OctMiner* to visually identify occurrences of the *setdiff* function in the repository routines and the way they are used.
Identify the category that the function belongs to: the programmer should configure *OctMiner* to spot other functions that belong to the same category of *setdiff* to help in the comprehension tasks.
Identify similar functions from the repository that can replace *setdiff*: configure *OctMiner* to support the identification of similar functions that can replace the target function.
Verify if the gathered information was enough to clarify the doubt: the user can now be more confident and can agree why the answer was the one with most votes.

Executing the Steps. In this section, we describe how *OctMiner* can help programmers to clarify their doubts about MATLAB and Octave. To this end, we use Fig. 4 through Fig. 7, which apply one of the following two types of configuration. Type I, presented in Fig. 4 and Fig. 5, focuses on files and their respective functions. In these figures, each rectangle from the *Grid view* (part D) represents a file together with the number of function categories found there one. Each rectangle from the *List view* (part E) represents the complete name and path of each file. In the case of the *TreeMap view* – part G of Fig. 4 – all rectangles together convey a panoramic visual representation of the files. In fact, the *TreeMap view* conveys a 2D visualization that maps a hierarchical structure into rectangles with each rectangle representing a file. In that case, files and functions are represented as nested rectangles, where the innermost rectangles are functions and the outermost rectangles are files. In the configuration type I, the size of each innermost rectangle corresponds to the number of functions implemented in each file and the color is associated to the category of the function.

Configuration type II (Fig. 6 and Fig. 7) is focused on the functions. Each rectangle from the *Grid view* (part D) represents a function together with their number of occurrences in the repository, in which multiple occurrences in the same file are counted. Each rectangle from the *List view* (part E) represents the complete name of function from the repository. The rectangles from the *TreeMap view* represent functions and the size of each rectangle is proportional to the number of times a function appears in the repository. The colors represent the category of each function.

When the user executes *OctMiner* and selects the option "*Visualize with OctMiner*", the tool conveys a typical visual scenario like the one presented in Fig. 4. This scenario uses the configuration type I, which focus on the analyzed repository files and their respective functions. In a first step, the user can understand the way the functions are distributed in the repository based on the information provided by the views as described in the following sentences. The *Grid view* (part D of Fig. 4) provides an overview of how many types of functions are implemented in each file (rectangle) of the repository (all rectangles). The *List view* (E) lists each files name and location in the repository. The *TreeMap view* (G) provides an overview of the files from the repository. Using this visual metaphor, functions found in a file are represented in the rectangle representing that file. A single screen shot can show all functions and files in accordance with its position in the file structure. We adapted the *TreeMap* visual paradigm to use colors to represent categories to which each specific function belongs. The difference of Fig. 4 and Fig. 5, is that in Fig. 5 we apply a filter (indicated by a red ellipse in part F) to highlight the files that implement *setdiff*. As a result of the filter, part G of the same figure highlights the functions where *setdiff* occurs by painting the rectangles in green.

The next step is the configuration of *OctMiner* to present the visual scenario of Fig. 6 that applies the configuration type II focusing on functions. The *List view* (part E of Fig. 6) enables checking the exact name of the routine, as well as the category to which the function belongs by looking at the color of the rectangles. The green color indicates that *setdiff* belongs to category "*Set Operations*". The user can access and read the code of specific routines (Part C) and analyze the various ways in which the function is used.

Using the type II configuration, focusing on functions, it is possible to spot the *Treemap view* displaying the largest green rectangle to represent the *setdiff* function. On the other hand, the *Grid view* complements this information by reporting that there are eleven occurrences of this function in the analyzed repository. The second largest rectangle from the *Treemap view* belongs to the same category and refers to function *isMember*. Interestingly, that function can replace *setdiff*: to find out if a vector is a subset of another, we can use *isempty(setdiff(a, b))* where a and b are arrays – but we will get the same result using *all(isMember(a, b))*. Based on information provided by *OctMiner*, we could identify an alternative function to *setdiff* if necessary. Different functions, like those already mentioned from the same category, are indications of the dominant category of the repository under analysis. The function *setdiff* belongs to the group *"set Operations"* but the category with more distinct features is *"Array and Matrix Creation and Concatenations"*, which includes a large number of functions to deal with arrays and matrices. In Fig. 7, this category is highlighted to emphasize that

a high number of distinct functions in a given category is a possible indicator of auxiliary functions, which may be of interest for the user.

The aforementioned conclusions can be confirmed from the replies registered at *StackOverFlow*. The user now can be more confident to understand the answers provided by the repository, considering both the target and similar functions, their utility, as well as the way they can be used to solve the stated problem.
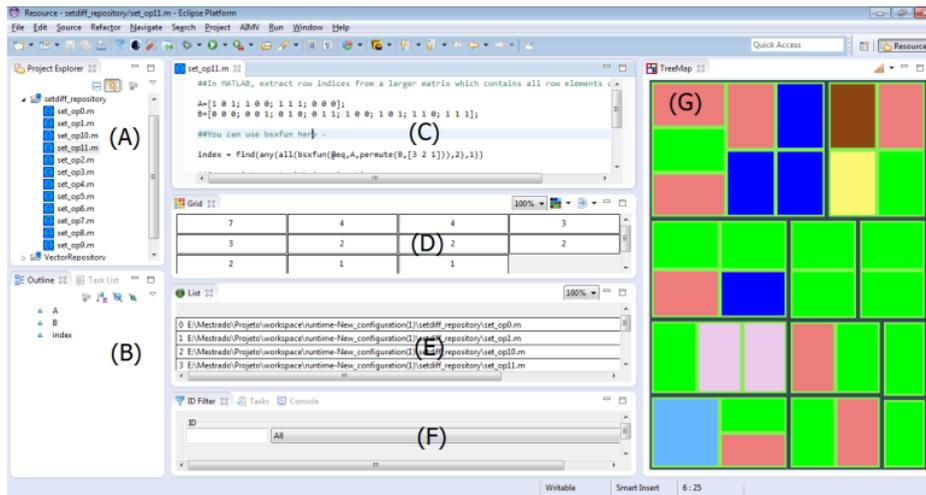

Fig. 4. *OctMiner* panoramic views for the initial analysis
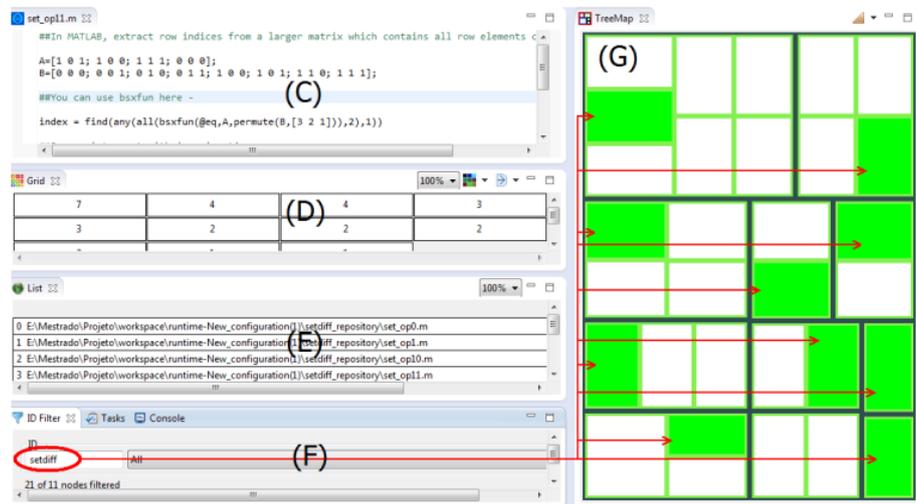
Fig. 5. Using filters to identify *setdiff* function occurrences
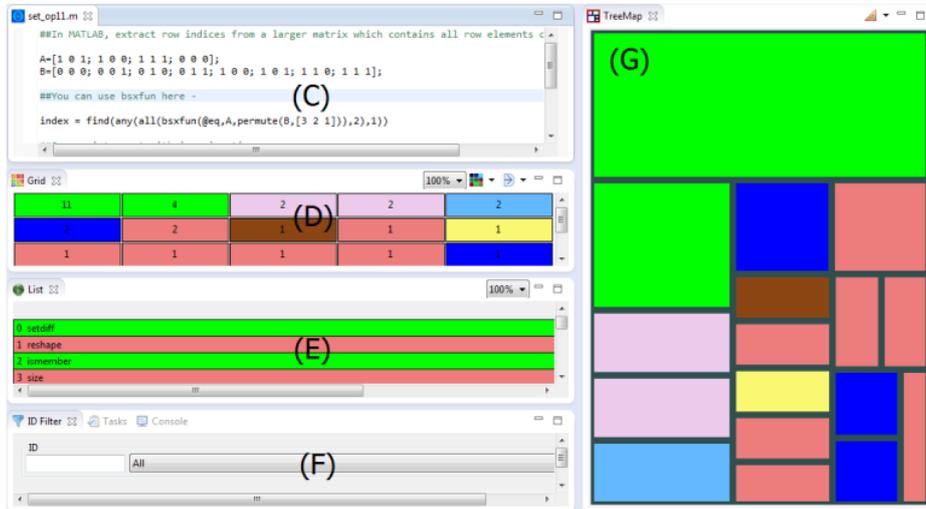
Fig. 6. Visual representation of functions from the repository
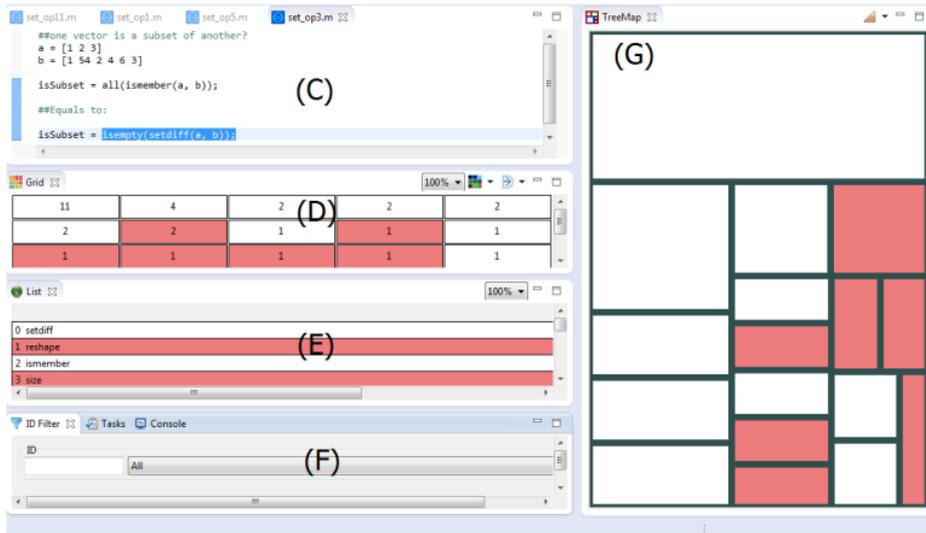
Fig. 7. Visual representation of functions in *OctMiner*

Even though this example is simple, it illustrates the benefits from using *OctMiner* to support comprehension. The combined use of the configuration types I and II can be an effective way for the comprehension of particularities of MATLAB and Octave programming that would be difficult to notice through non-visual, non-interactive approaches.

In the second study, one of the potential threats related to external validity (to which extent results can be generalized) is that just one question from *StackOverflow* was evaluated in *OctMiner*. The environment might not easily fit other issues registered at *StackOverflow*. However, we do not expect that the case studies presented in this paper should to be generalizable to all types of issues and questions from *StackOverflow*. The purpose of both studies was to provide insights about the potential of *OctMiner* as support for the comprehension of MATLAB/Octave programs. The first study had the goal to use *OctMiner* to support the detection and characterization of crosscutting concerns [5], as well as to characterize the use of *OctMiner* and improvement opportunities of its use. The second study explored two configuration types to use *OctMiner* for supporting comprehension of issues posted at *StackOverflow*.

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. Another potential threat to validity is that both the design and the execution of the study were performed by the same person. To overcome this issue, further independent experiments should be carried out to better compare results.

4 A Comprenhension Strategy based on *OctMiner*

The results from the two studies enabled us to propose a set of usage strategies based on *OctMiner* for comprehension purposes. The set has a comprehension question that drives the strategy steps as a starting point. The question of the first study was related to tangling and scattering, using a set of tokens from programs of a repository as a basis. The second study focused on questions posted at *StackOverflow* by programmers. Table 3 presents the steps proposed from evidences collected from the two case studies presented in this paper.

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 functions used in the code should be registered in the *OctMiner* configuration file. A repository of questions and answers, such as StackOverFlow, may be used for this purpose, as illustrated in the second study.
2 – **Identify a target function:** it should be the function that plays a relevant role in the code of the primary solution to the selected question. In repositories such as the StackOverFlow, the best ranked answers usually indicate the relevant function to solve the problem.
3 – **Locate repositories that use the target function:** since *OctMiner* aims at assisting the comprehension of a given target function, it is desirable that routines using the target function provide good examples and be the subject of analysis.

4 – Identify the functions and their respective categories available in the official documentation: alternative functions used in the repository selected in Item 3 must also be identified. MATLAB and Octave functions are categorized in the official language site of MATLAB and Octave.

5 – Register the target function as well as other function from the repository in the *OctMiner* configuration file: the functions 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*. In the example from the second preliminary study, the user is directed through four comprehension tasks centred on the *setdiff* function.

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.

5   Related Work

Research on MATLAB and Octave program comprehension is in its infancy. A simple proof of this claim can be obtained with *Google Scholar*. While the search string *"Java program comprehension"* returns a considerable number of hits[9], at the time of writing this paper, similar searches with MATLAB[10] or Octave[11] did not match any articles. Therefore, we enlarged our search to include related aspects such as static analysis, code refactoring, reverse engineering or program transformation and optimization.

The oldest reference found was from V. Menon and K. Pingali, where the authors proposed three kinds of source-to-source transformations for optimizing MATLAB programs and show their effectiveness [14]. They claim that transformations yield performance benefits additional to those obtained by (optimizing) compilation, and may be useful for other DSLs that are high-level, untyped, and interpreted.

In spite of MATLAB's popularity, and the need for static analysis (e.g. for program optimization, code smells detection, refactoring), Jesse Doherty claimed that there was no publicly available framework for creating static analyses for that language, until he created the *McLAB Static Analysis Framework (McSAF)* [15] [16]. The goal of this framework was to make new analyses easy to write and to extend to new language features.

Soroush Radpour, also at McGill University [17], developed a tool named *McBench[12]* that is claimed to help compiler writers understand the language better, by giving some insight about how programmers use MATLAB. He also proposed a suite

---

[9]   *https://scholar.google.pt/scholar?q="Java+program+comprehension"*

[10]   *https://scholar.google.pt/scholar?q="MATLAB+program+comprehension"*

[11]   *https://scholar.google.pt/scholar?q="Octave+program+comprehension"*

[12]   see *https://github.com/isbadawi/mcbench*

of semantic-preserving refactoring for MATLAB functions and scripts including: function and script inlining, converting scripts to functions, extracting new functions, and converting dynamic *feval* (function evaluation) calls to static function calls.

Last, but not the least, Anton Dubrau and Laurie Hendren, again from the *McLab Project*[13] team at McGill University, claim that MATLAB users often want to convert their programs to a static language such as Fortran [18]. They developed an object-oriented open source toolkit, called *Matlab Tamer*[14], for supporting the generation of static programs from dynamic MALTAB programs.

6   Conclusions and Future Work

MATLAB and Octave are popular languages for numerical computations used by scientists, engineers and students worldwide. As their programs grow in size and complexity, they face the usual maintenance challenges that originated the emergence of the program comprehension domain in Computer Science. Software visualization techniques can mitigate those maintenance challenges, but as far as we could devise, their use has not yet been adopted by the MATLAB and Octave communities.

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 first study; c) Evidences of the effectiveness of *OctMiner* to understand the solutions proposed in a popular question-and-answer site for professional programmers, regarding MATLAB and Octave languages as discussed in the second study; d) a set of usage strategies of *OctMiner* for comprehension purposes.

A preliminary version of this paper, describing *OctMiner* architecture, along with an illustrative example of its main functionalities in a real scenario of program comprehension, was presented at ITNG'2015 [9]. A short paper, including a first case study for validating *OctMiner* feasibility, was presented at SEKE'2015 [19]. On the current version, additional details are provided on the validation case studies and additional information on the proposal is provided.

We now plan to conduct a controlled experiment where engineering undergraduate students will perform comprehension activities with and without the support of *OctMiner*. We also plan to include collaborative resources in *OctMiner* to enable programmers to communicate and cooperate among themselves to more effectively achieve software comprehension activities related to MATLAB and Octave software development.

---

[13]   see *http://www.sable.mcgill.ca/mclab/*
[14]   see *http://www.sable.mcgill.ca/mclab/projects/tamer/*

References

1. Seacord, R., Plakosh, D., Lewis, G.: Modernizing legacy systems: software technologies, engineering processes, and business practices. Addison-Wesley Professional (2003)

2. Card, S., Mackinlay, J., Shneiderman, B.: Readings in Information Visualization Using Vision to Think. Morgan Kaufmann, San Francisco, CA (1999)

3. Carneiro, G., Silva, M., Mara, L., Figueiredo, E., Sant'Anna, C., Garcia, A., Mendonça, M.: Identifying code smells with multiple concern views. In : proceedings of the XXIV Brazilian Symposium on Software Engineering (SBES'2010), pp.128–137 (2010)

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

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

6. Spence, R.: Information Visualization: Design for Interaction 2nd edn. Prentice Hall (2007)

7. Carneiro, G., Mendonça, M.: SourceMiner: Towards an Extensible Multi-perspective Software Visualization Environment. In Hammoudi, S., Cordeiro, J., Maciaszek, L., Filipe, J., eds. : Enterprise Information Systems 1st edn. Springer International Publishing (2014) 242-263

8. Nunes, A., Carneiro, G., David, J.: Towards the Development of a Framework for Multiple View Interactive Environments. In : proceedings of the International Conference on Information Technology: New Generations (ITNG'2014), Las Vegas, USA, pp.23-30 (2014)

9. Lessa, I., Carneiro, G., Monteiro, M., Brito e Abreu, F.: A Multiple View Interactive Environment to Support MATLAB and GNU/Octave Program Comprehension. In : proceedings of the International Conference on Information Technology: New Generations (ITNG), Las Vegas, USA (2015)

10. Chaves, J., Nehrbass, J., Guilfoos, B., Gardiner, J., Ahalt, S., Krishnamurthy, A., Unpingco, J. ., Warnock, A., Samsi, S.: Octave and Python: High-Level Scripting Languages Productivity and Performance Evaluation. In : proceedings of the HPCMP Users Group Conference'06 (2006)

11. 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)

12. Robillard, M., Murphy, G.: Representing Concerns in Source Code. ACM TOSEM (2007)

13. Tarr, P., Ossher, H., Harrison, W., Jr., N. I.: Degrees of Separation: Multi-Dimensional Separation of Concerns. In : proceedings of the ICSE'99 (1999)

14. Menon, V., Pingali, K.: A case for source-level transformations in MATLAB. In : proceedings of the DSL'99, pp.53–66 (1999)

15. Doherty, J.: McSAF: An extensible static analysis framework for the MATLAB language. MSc dissertation, McGill University, Montréal, Canada (September 2011)

16. Doherty, J., Hendren, L.: McSAF: A Static Analysis Framework for MATLAB. Sable Technical Report sable-2011-01, McGill University, Montréal, Canada (December 2011)

17. Radpour, S.: Understanding and refactoring the MATLAB language. MSc dissertation, McGill University, Montréal, Canada (August 2012)

18. Dubrau, A., Hendren, L.: Taming MATLAB. Sable Technical Report sable-2011-04, McGill University, Montréal, Canada (December 2011)

19. Lessa, I., Carneiro, G., Monteiro, M., Brito e Abreu, F.: Scaffolding MATLAB and Octave Software Comprehension Through Visualization. In : proceedings of the 27th International Conference on Software Engineering and Knowledge Engineering (SEKE'2015), Pittsburgh, USA (2015)