A Concern Visualization Approach for Improving MATLAB and Octave Program Comprehension

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)
ISTAR-IUL
Lisbon, Portugal
fba@iscte-iul.pt

Abstract— **The literature has pointed out the need for focusing efforts to better support comprehension of MATLAB and Octave programs. Despite being largely used in the industry and academia in the engineering domain, programs and routines written in those languages still require efforts to propose approaches and tools for its understanding. Considering the use of crosscutting concerns (CCCs) to support the comprehension of object-oriented programs, there is room of its use in the context of MATLAB and Octave programs. The literature has purpose and examples in this direction. Considering this scenario, we propose the use of visualization enriched with CCCs representation to support the comprehension of such programs. This paper discusses the use of a multiple view interactive environment called *OctMiner* in the context of two case studies to characterize how collected information relating to crosscutting concerns can foster the comprehension of MATLAB and GNU/Octave programs. As a result of the conducted case studies, we propose strategies based on *OctMiner* and tailored to support different comprehension activities of programs written in MATLAB and Octave.**

Keywords- MATLAB/Octave; software comprehension; crosscutting concerns; software visualization.

I. INTRODUCTION

MATLAB is a popular interpreted programing language among students and researchers of physics, biomedical and telecom engineering, among other 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][18]. 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[1].

Our literature review reported in [9] indicates a lack of support for the comprehension of programs coded in these languages. We tackled this research opportunity by implementing a MVIE named *OctMiner* [8], a multiple view interactive environment (MVIE) that provides resources to support data analyses and unveiling information that otherwise would remain unnoticed [1][4].

Two case studies were conducted with the tool, to support the comprehension of MATLAB/Octave programs. The former aimed at identifying crosscutting concerns (CCCs) [7] following previous research on the issue [2][10]. The latter aimed at assessing to which extent a concern-oriented use of *OctMiner* could help programmers understand the solutions proposed in StackOverflow[2] , a popular question-and-answer site for professional programmers.

This paper consolidates the insights acquired during preliminary research, namely when we used *OctMiner* to identify symptoms of scattering and tangling [8]. In previous work, we outlined a strategy, comprising a sequence of steps to configure and use *OctMiner* [9]. In this paper, we propose a new strategy for the configuration and usage of *OctMiner* for the comprehension of routines whose main characteristics are not familiar to the user. This is a scenario in which the user does not possess prior knowledge on the repository of MATLAB/Octave routines to be analyzed. We figured out the importance of this scenario during our efforts to derive the first strategy [9].

The research questions (RQ) raised in this research are: **RQ1** – What are the concepts and/or entities in MATLAB/Octave programs whose comprehension can benefit from the support of *OctMiner*? **RQ2** – To which extent *OctMiner* can provide support for the comprehension of MATLAB/Octave programs? We implemented *OctMiner* to answer these research questions following an iterative and incremental approach, driven by the insights acquired from the conducted case studies and the strategies proposed for its usage.

This paper is organized as follows. Section II describes the context of this work (part i of **Figure 1**); Section III presents the visualization environment called *OctMiner* (part iii of **Figure 1**); Section IV characterizes the use of *OctMiner* through two case studies (part iv of **Figure 1**); Section V presents strategies to tailor *OctMiner* to best fit programmers' needs (part v of **Figure 1**); Section VI presents conclusions and future work along with the analysis and answers of the research questions (part vi of **Figure 1**).

[1] http://www.gnu.org/software/octave/

[2] www.stackoverflow.com

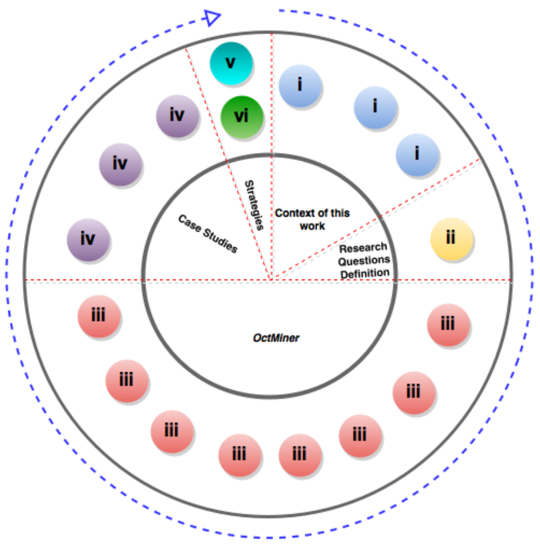


Figure 1. Steps of our Research

II. CONTEXT

In this section we present relevant concepts and approaches to better contextualize our work.

A. *Software Visualization and MVIEs*

Visualization is a means of providing perceivable cues to several aspects of the data under analysis to unveil patterns and behaviors that would otherwise remain "under the radar" [17]. Card et al. [1] proposed a well-known reference model for information visualization. Following this approach, the creation of views goes through a sequence of 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 and emphasizing the visualization process as highly interactive. Nunes et al. [13] 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. This MVIE was originally developed to support the comprehension of Java source code bases. The extension points of the MVIE Eclipse plugin enable the attachment of new plugins to the MVIE. Each of the extension points provides an interface with methods and their respective signatures. The goal of the toolkit is to provide an infrastructure to develop MVIEs for different domains.

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. It is worth noticing that due to the similarities between these two languages, it is possible to interpret MATLAB programs in the interpreter of the GNU/Octave with no major problems.

B. *Crosscutting Concerns*

The present work is based on the idea initially proposed by Monteiro et al [12], for which we next provide a short summary. A concern is anything a developer wants to consider as a conceptual unit, including domain-specific features, non-functional requirements, and design patterns [19][16]. Ideally, any decomposition unit of a software system –function, in the case of MATLAB – would enclose code relating to a single concern. However, in many cases, a single function encloses code conceptually related to multiple concerns, which means that some concerns are not modularized and cut across the modular boundaries of that software system.

It is known that a number of common concerns occurrences over the modular structure of software systems, namely persistence, transaction management, security and caching. These "unmodularized" concerns are known as *crosscutting concerns* [7].

The phenomenon of CCCs in object-oriented systems is well studied [16]. However, the nature, characteristics and revealing symptoms of CCCs in MATLAB and Octave systems did not yet receive equal attention. Monteiro et al. [12] considered the importance of CCCs in the context of MATLAB/Octave program.

Monteiro et al [12] propose an approach to automatically obtain indicators of the presence of CCCs in MATLAB/Octave code, by analysis of a few metrics relating to occurrence of function names and how those names – here, also referred as "tokens" – are distributed across the MATLAB/Octave files. For each separate MATLAB file, several metrics are computed, including count of occurrences of a function name in that file as well as the number of different functions. For an entire subject repository of functions – possibly comprising many toolboxes – a few aggregate statistics are computed from these counts. Monteiro et al [12] also proposed an illustrating list of CCCs in MATLAB/Octave systems. The CCCs are grouped in several categories. They also propose sets of tokens whose occurrence can be used as indicators of the presence of a given CCC in the MATLAB/Octave system – see **Table 1**. For more details, see [12].

III. THE *OCTMINER* MVIE

Based on the approach presented in the previous section, the *OctMiner* MVIE for MATLAB and Octave programs supports the identification of CCCs through the use of multiple views [8]. The main motivation for representing concerns manifested in MATLAB/Octave code in a MVIE is the enhancement of the comprehension activities. To this end, a novel mapping was derived from information obtainable from the AST, based on two primary dimensions: MATLAB/Octave files and function names.

We developed *OctMiner* using the Toolkit proposed in [13]. The toolkit can be extended through the use of plugins. The MVIE toolkit enables this to tailor the MVIE for the analysis of data from different domains, e.g., the data gathered from MATLAB/Octave programs. **Figure 2** 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 [13]. The *Octclipse* plugin provides a Matlab/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 [15].

Table 1. Examples of Tokens per Category of Concern

CCC Category	Examples of Tokens
Messages and monitoring	plottools, semilogx, semilogy, loglog, plotyy, plot3, grid, title, xlabel, ylabel, zlabel, axis, axes, hold, legend, subplot, scatter, ishold, newplot, figure, cla, clf, reset, close, plot, polar,**error**, display,zoom, assert, disp;
I/O data	imwrite, imread, imformats, hgsave, saveas, hgload,save, loade, diary, fwrite,fileformats, movie,image, fread, fscan;
Verification of function arguments and return values	nargchk, nargin, nargout, varargin, vararout;
Data type verification and specialization	int8, int16, int32, int64, quantize, quantizier, fi, isscalar, isstruct, isfield, iscell, isempty;
System	pause, print, printop, wait, last, mex, inmen, batch, pack, pcode, echo, input, syntax, run, tic, start;
Memory allocation/deallocation	clear, delete, zeros, persistent, global;
Parallelization	parfor,spmd, gpuDevice, feval, demote, taskStartup, cancel, submit, resume;
Dynamic properties	eval, evalc, evalin, inline.

We implemented an Analyzer module to import and convert data from the original data repository to be represented in the multiple views. To this end, we performed a mapping between the real and visual attributes as presented in the following paragraphs.

Different comprehension activities can be supported by the visual metaphors provided by *OctMiner*. They can be set and tailored according to the degree of detail in the comprehension task. For this purpose, *OctMiner* provides three types of semantic zoom to apply new representation (visual attributes) of real attributes [17]. The first type focuses on the file repository level. It has a result a high level visual representation – called "group All" – where the views portray information relating to the entire repository of MATLAB/Octave files under analysis. This level is indicated for a panoramic view of files. In this level the files can be ordered by the number of tokens, as well as the number of different groups of tokens manifested in each of them. The second type of semantic zoom provides the transition for the file level. It provides detailed information regarding a specific MATLAB/Octave file and their respective count and group of tokens. The third type focuses on the token – called "group Token". This level provides representation of more than one view for each token.

Figures 3, **4** and **5** illustrate the mapping between real and visual attributes in the design of visual metaphors in *OctMiner*. In the following paragraphs we will describe how we performed this mapping.

Figure 3 uses the treemap visual metaphor. Examples of real attributes used in these figures are MATLAB/Octave function (token) and how many times this function is manifested in a file or repository. These real attributes are represented with the following visual attributes: the name of each token and the size of each rectangle respectively. Examples of token names are "pi", "floor", "quantizier" and "x". In the same figure, these tokens are manifested in "file1.m", "file2.m" and "file3.m". The category of each CCC as described in **Table 1** is another real attribute that is mapped to the visual attribute color. In **Figure 3**, the groups "All" provides a visual representation of different tokens manifested in the three files that hypothetically comprises the repository, where the colors represent the category that each token belongs to and the size of each rectangle is related to how many times this token is used in the file. The group "Token" from the same figure focuses on the representation of how the tokens are manifested in the entire repository. The group "File" conveys the same real attributes now focusing on a specific file.

Figure 4 uses the grid as a visual metaphor. In this figure, it is possible to select the group "All" to see the list of all files from the repository under analysis, by order of quantity of tokens found in each file. In the group "Token", the view aggregates in a single block the counts of all tokens found in the files from the repository. When viewing single files, we can see which tokens manifested in a file, how many times they occur and to which category this token belongs to considering the color of each rectangle.

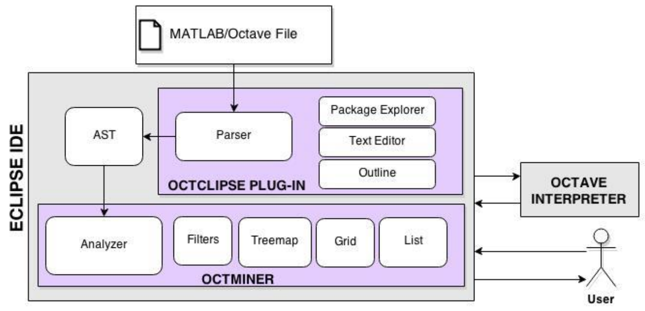


Figure 2. *OctMiner* Architectural Overview [8]

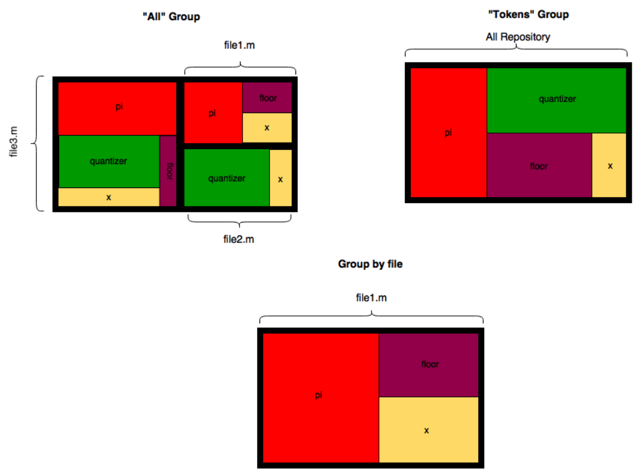


Figure 3. Details of the Design of the TreeMap view for *OctMiner*.

An illustration of the List view is the visual metaphor selected represented in **Figure 5**. In this example, this view is used to present a listing of all files from the repository under analysis, when group "All" is selected. When group "Token" is selected, a list of the tokens found is presented. These can either refer to counts from the entire repository (group "Token") or from a single, specific file (group "File"). The ordering of files and tokens depend on how they are actually organized in the system using *OctMiner*.

OctMiner can be configured to expose symptoms of CCCs in MATLAB/Octave routines. As can be seen in the following paragraph, 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. **Table 2** shows a mapping of categories of CCCs to colors, which result from the XML specifications. These colors and category names are used in the two case studies described next.

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

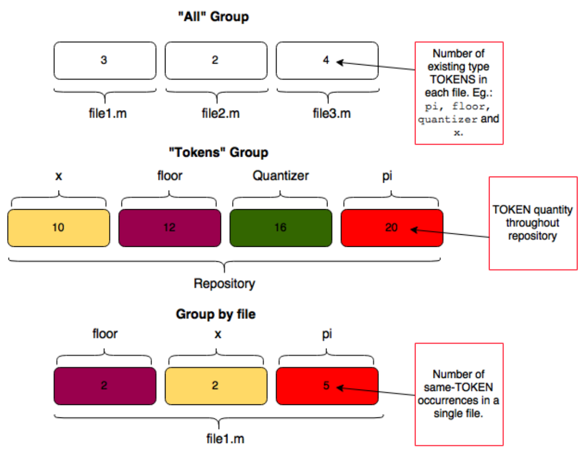

"All" Group
3
2
4
file1.m
file2.m
file3.m
Number of existing type TOKENS in each file. Eg.: pi, floor, quantizer and x.
"Tokens" Group
x
floor
Quantizer
pi
10
12
16
20
TOKEN quantity throughout repository
Repository
Group by file
floor
x
pi
2
2
5
Number of same-TOKEN occurrences in a single file.
file1.m

Figure 4. Details of the Design of the Grid view for *OctMiner*.

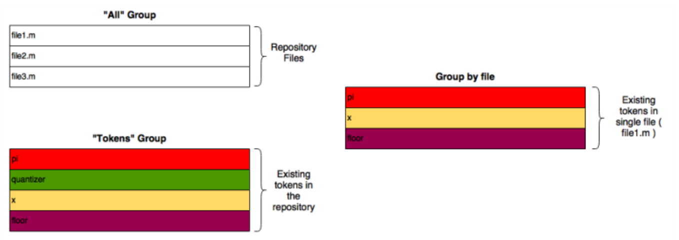


Figure 5. Details of the Design of the List View for *OctMiner*.

Table 2. Categories and their Colors in *OctMiner* [8]

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	🟨

IV. Case Studies

A. *The First*

The first study investigated the following questions: to which extent *OctMiner* provides effective support to identify potential symptoms of CCCs in Matlab programs? And to which extension these symptoms support the comprehension of the analyzed programs? The study aimed at analyzing 22 MATLAB image processing routines. The goal was the identification of scattering and tangling in these routines supported by *OctMiner*. Scattering [7][16] is the degree to which a concern is spread over different modules or other units of decomposition. Tangling [7] is the degree to which concerns are intertwined to each other in the same functions. Both scattering and tangling are indicators of the presence of CCCs in program code. The term token refers to function names found in MATLAB/Octave systems.

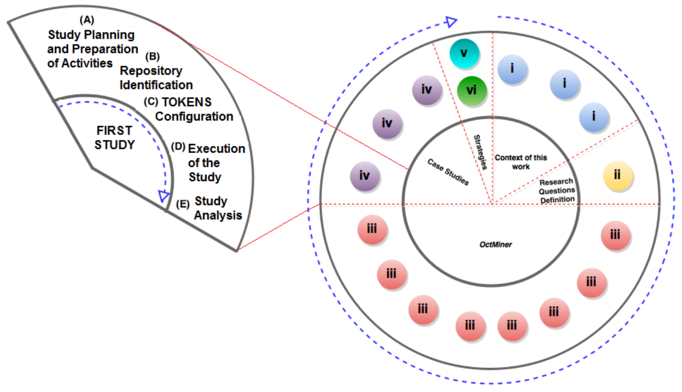


Figure 6. Steps of the First Study

The study explores the potential of these 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.

The steps followed the sequence represented in **Figure 6**. The study starts with the planning of the study and preparation of the activities to be carried out (**Figure 6** – A). Next, we select and identify the MATLAB/Octave repository (**Figure 6** – B) and the tokens are set (**Figure 6** – C) in the XML file. After this configuration, *OctMiner* is ready to be used (**Figure 6** – D) and planned activities should be performed. Finally, analysis of results is carried out (**Figure 6** – E) and conclusions related to the research questions are established. These steps are next described in detail.

To explore the above approach, participants performed the following activities: i) Identify tokens most commonly used in the 22 routines; ii) Characterize the location 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 tokens most commonly used in the routines under analysis and whether these tokens present 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 having as a source the routines recently modified; the user can only select the predefined color in *OctMiner*, Presently it is not possible in this version to define new colors: the need to configure the XML file with the tokens is therefore a limitation. To address it, in future we plan to provide a XML file with a large number of MATLAB and Octave functions and their respective categories. The results of this study are presented in [8].

B. The Second

The second study assessed OctMiner and brought insights on how to improve and mature the tool [9]. In the study, OctMiner was configured with the aim to address some of the issues brought by MATLAB/Octave programmers. To this end, an analysis of the most common issues raised by the StackOverflow community was performed (**Figure 7** – F). Questions posted about MATLAB/Octave were classified (**Figure 7** – G). A repository of StackOverflow posts was selected (**Figure 7** – H), according to which a number of functions (preset tokens) was configured in the XML file (**Figure 7** – I). Next, the study was carried out as planned (**Figure 7** – J). Details of the process are shown next.

The second study comprised the following research questions: a) To which extent *OctMiner* provides effective support to clarify programmers' issues, based on answers posted at StackOverflow? b) To which extent does the CCCs exposed through *OctMiner* support the clarification of programmers' issues? The primary goal of this study was to demonstrate the effectiveness of *OctMiner*'s visualization of target functions in exposing the kind of functions suggested in StackOverflow posts. The views provided by *OctMiner* help programmers to understand the context of use of a function in routines from the repository of MATLAB code.

We searched for the issues raised most often about MATLAB and Octave as well as the corresponding better ranked answers. We used the StackExchange Data Explorer tool [6] to perform a search. We applied the following query and as a result obtained the top 200 questions related with 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 0: (a) Programming language basic issues – 146 questions; (b) Common mistakes in MATLAB and Octave – 51 questions; (c) Using MATLAB and Octave functions to perform specific work such as numerical calculation and image processing – 98 questions; (d) Using MATLAB and Octave functions to plot data on the screen – 69 questions; (e) Questions that do not fit into any of the others above – 56 questions. Considering the top 200 questions/issues retrieved by the query, we verified that there were overlapping among them, so several questions/issues were classified in more than one category.

Category "a" has the highest count, which indicates a lack of basic knowledge of the two languages. We considered this fact as the starting 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)*".

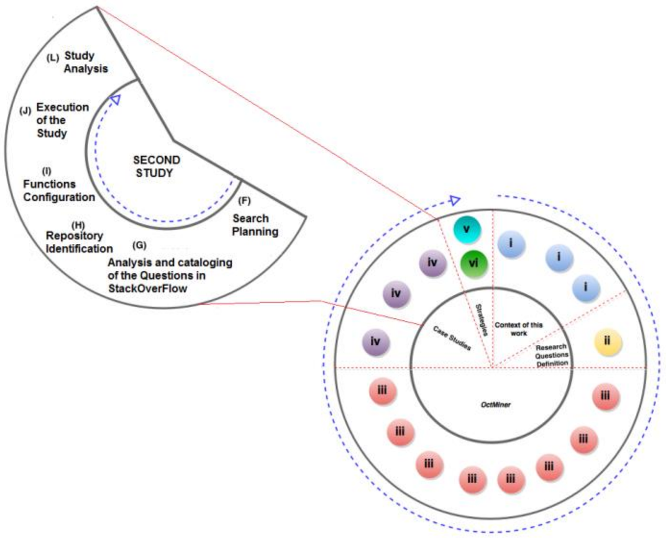

(L) Study Analysis
(J) Execution of the Study
SECOND STUDY
(I) Functions Configuration
(F) Search Planning
(H) Repository Identification
(G) Analysis and cataloging of the Questions in StackOverFlow
Strategies
Context of this work
Case Studies
Research Questions Definition
OctMiner
v
vi
i
iv
ii
iii

Figure 7. Steps of the Second Study

We selected 22 MATLAB routines to illustrate the use of the `setdiff` function, the target function of the selected issue raised. We selected these routines by searching through the StackOverFlow repository using string "MATLAB `setdiff`". We also registered `settdiff` function and all 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 shows the categories and their respective colors as used in *OctMiner*.

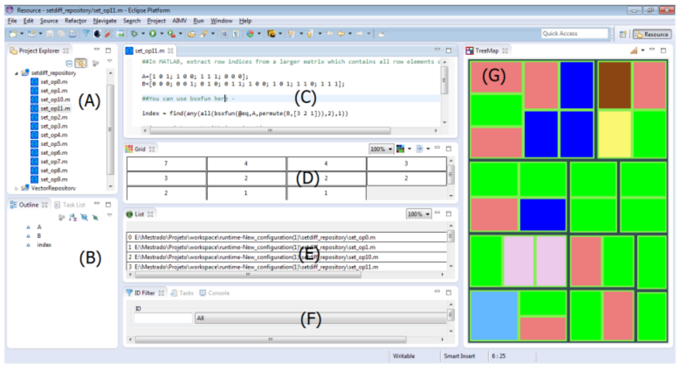
C. *Focusing OctMiner on the setdiff Function*

Based on the experience acquired, we proposed a set of steps – see **Table 3** – focusing on the comprehension of function `setdiff` supported by *OctMiner* to clarify the issue raised at StackOverFlow. Details on how the steps were executed can be obtained in [9].

The steps listed in **Table 3** are illustrated from **Figure 8** to **Figure 11** to illustrate one of the following two types of *OctMiner* configuration used in the studies. Type I, presented in **Figure 8** and **Figure 9**, focuses on the program file/function dimension. Each rectangle from the Grid view (part D of Figures 8-11) represents a file along with the number of function categories found there. Each rectangle from the List view (part E) represents the complete name and path of each program file. In the case of the TreeMap view (part G) all rectangles together convey a panoramic visual representation of the files. Configuration type II (**Figure 10** and **Figure 11**) 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 next step is the configuration of *OctMiner* to present the visual scenario of **Figure 10** that applies the configuration type II focusing on functions. The List view (part E of **Figure 10**) enables checking the exact name of the function and also 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.

Table 3. Proposed Steps in *OctMiner [9]*

Steps
Select a question: to clarify an issue raised.
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 answer the question: the user can now be more confident and can agree why the answer was the one with most votes.

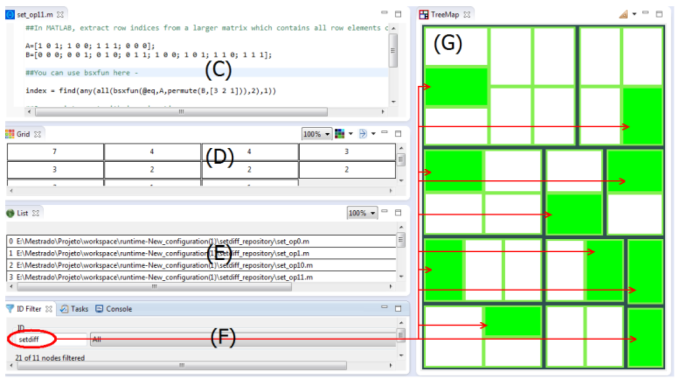


```
##In MATLAB, extract row indices from a larger matrix which contains all row elements c

A=[1 0 1; 1 0 0; 1 1 1; 0 0 0];
B=[0 0 0; 0 0 1; 0 1 0; 0 1 1; 1 0 0; 1 0 1; 1 1 0; 1 1 1];

##You can use bsxfun here -

index = find(any(all(bsxfun(@eq,A,permute(B,[3 2 1])),2),1))
```

7	4	4	3
3	2	2	2
2	1	1	

```
0 E:\Mestrado\Projeto\workspace\runtime-New_configuration(1)\setdiff_repository\set_op0.m
1 E:\Mestrado\Projeto\workspace\runtime-New_configuration(1)\setdiff_repository\set_op1.m
2 E:\Mestrado\Projeto\workspace\runtime-New_configuration(1)\setdiff_repository\set_op10.m
3 E:\Mestrado\Projeto\workspace\runtime-New_configuration(1)\setdiff_repository\set_op11.m
```

Figure 8. *OctMiner* Panoramic Views for the Initial Analysis [9]



```
##In MATLAB, extract row indices from a larger matrix which contains all row elements c

A=[1 0 1; 1 0 0; 1 1 1; 0 0 0];
B=[0 0 0; 0 0 1; 0 1 0; 0 1 1; 1 0 0; 1 0 1; 1 1 0; 1 1 1];

##You can use bsxfun here -

index = find(any(all(bsxfun(@eq,A,permute(B,[3 2 1])),2),1))
```

7	4	4	3
3	2	2	2

```
0 E:\Mestrado\Projeto\workspace\runtime-New_configuration(1)\setdiff_repository\set_op0.m
1 E:\Mestrado\Projeto\workspace\runtime-New_configuration(1)\setdiff_repository\set_op1.m
2 E:\Mestrado\Projeto\workspace\runtime-New_configuration(1)\setdiff_repository\set_op10.m
3 E:\Mestrado\Projeto\workspace\runtime-New_configuration(1)\setdiff_repository\set_op11.m
```

ID: setdiff

21 of 11 nodes filtered

Figure 9. Using Filters to Identify *setdiff* Function Occurrences [9]

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.

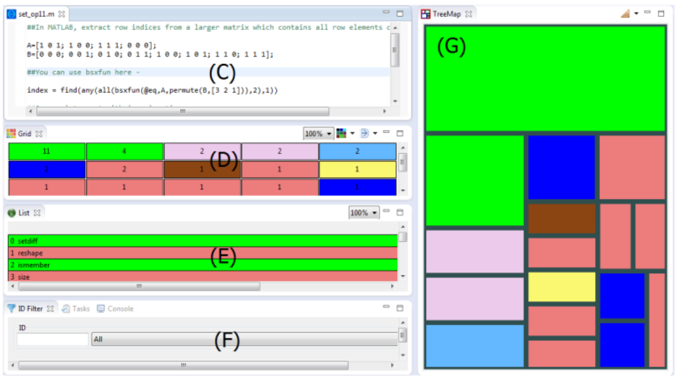

```
##In MATLAB, extract row indices from a larger matrix which contains all row elements c

A=[1 0 1; 1 0 0; 1 1 1; 0 0 0];
B=[0 0 0; 0 0 1; 0 1 0; 0 1 1; 1 0 0; 1 0 1; 1 1 0; 1 1 1];

##You can use bsxfun here -

index = find(any(all(bsxfun(@eq,A,permute(B,[3 2 1])),2),1))
```

Figure 10. Functions Visual Representation from the Repository [9]

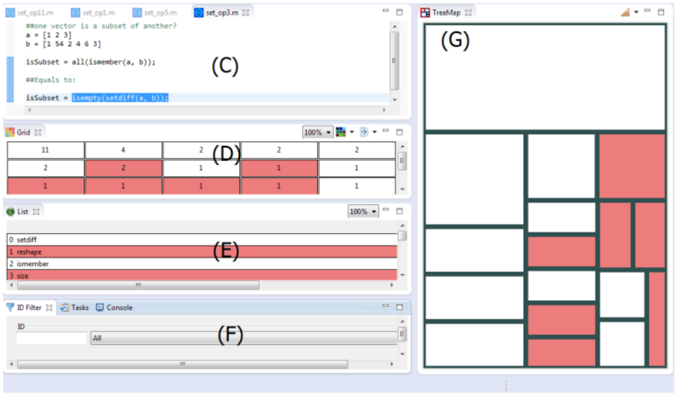

```
##one vector is a subset of another?
a = [1 2 3]
b = [1 54 2 4 6 3]

isSubset = all(ismember(a, b));

##Equals to:

isSubset = isempty(setdiff(a, b));
```

Figure 11. Visual Representation of Functions in *OctMiner* [9]

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 CCCs [12] 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.

V. COMPRENHENSION STRATEGIES BASED ON *OCTMINER*

The experience acquired 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 4 presents the steps proposed from evidences collected from the two conducted case studies.

A. An Alternative Strategy

We noticed that the strategy presented in Table 4 stems from a limitation. In comprehension tasks of MATLAB/Octave systems, we observed that it does not necessarily originate from a directly visible item or entity. For example, we may need to perform a task such as identifying the main functionalities of a repository without having previous information about it. This is an opportunity to use *OctMiner* for the support of different comprehension scenarios and to progress successfully from any of them.

The aim of this strategy presented in Table 5 is to guide the user having as a starting point a given MATLAB/Octave repository. Considering that the user is not familiar with the routines from the repository, the views provided by *OctMiner* should assist him/her by providing useful insights.

To validate the second strategy proposed, steps 1-6 also listed in Table 5 are carried out.

*1) Locate MATLAB/Octave repositories **for analysis:***

The repository used was randomly selected and comprises 17 MATLAB routines for image processing. They originate from a larger repository collected from MATLAB toolboxes freely available on the Web.

*2) **Identify functions from the repository and the categories to which they belong, according to information available in the official documentation***

After identifying the repository, the routines must be examined and each function classified. For the repository used, 41 functions were identified, on the basis of the official catalogue from MATLAB Functions [8].

*3) **Classify functions from the repository in OctMiner configuration file (XML)***

Using the XML file, each function is classified according to its respective category.

*4) **Propose list of environment exploration activities***

The activities proposed in this section are meant to guide the user of *OctMiner* through the discovery of relevant information, using the visualizations. These activities are sufficiently general to be suitable for any kind of MATLAB/Octave repository.

The activities are as follows

i. Find a MATLAB/Octave repository: following the second strategy, the first step to perform the analysis is to identify the repository we want to analyse visually;

ii. Identify the categories used most often, which may provide insights on the domain of application of the system under analysis.

iii. Locate the functions used most often: this information may assist the user in identifying the most important functionalities in the repository.

iv. Check whether the functions used most often belong to the same category: if yes, check whether they implement equivalent or complementary functionalities. A similar step was carried out in the second preliminary study (section IV.A), from which we noticed some equivalence of function;

v. Identify the largest count of different functions in a single category: at this point, the user should check whether distinct functions from the same category are auxiliary functions. A similar activity was also carried out in the second study, which yielded information on auxiliary functions.

vi. Analyse results: see what information can be derived from the visualizations used while conducting the activities.

Table 4 . A Proposed Set of Usage Strategies [9]

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 in Item 1.

Table 5. An Alternative Strategy

Suggested Steps
1. Identify MATLAB/Octave repositories for analysis: select a repository that includes a consistent domain of application. For instance, we can envision domains such as for signal processing, parallelization of a given application, benchmarking**.**
2. Identify functions in the repository as well as the categories to which they belong, following official documentation: identify the functions used in the selected repository. To this end, use the list of functions available in the MATLAB official documentation - MATLAB Functions (2014) and Octave Functions (2014)**.**
3. Classify the functions from the repository in the *OctMiner* configuration file (XML). For this file to work correctly, the various functions should be classified according to the concern which they are most closely associated, on the basis of the identification from step 2**.**
4. Propose list of exploration activities; some activities were defined to enable the user to discover relevant repository information.
5. Carry out exploration activities: the user runs *OctMiner* and performs the activities defined in step 4.
6. Analysis of activities: the user presents the results obtained by carrying out the activities set at step 5.

5) *Perform the activities for environment exploration*

OctMiner was executed in both first and second studies, taking advantage of available resources to identify the items mentioned in the activities. Some interesting points were noticed, as described next.

6) *Analysis of activities*

In activity 1, the MATLAB repository yielded a large number of functions to be catalogued. While consulting the official documentation [11], we followed the strategy of cataloguing each function according to its primary category, since subcategories varied too much to group them in the configuration file. No difficulties were felt, but several different ways to perform the configuration were apparent, e.g., use the various sub-categories in the XML file. This indicates how flexible the tool is.

Though the sub-categories were not used in activity ii, the visualizations provide a general idea of the kind of repository which is under analysis.

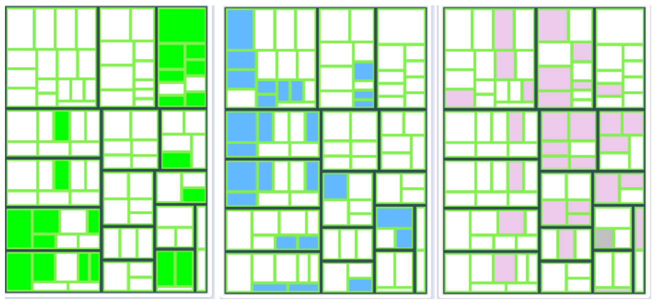
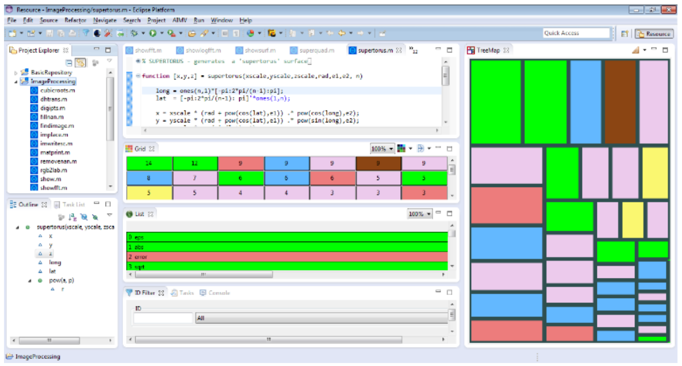


Figure 12. The Most Often Used Categories in TreeMap

In **Figure 12**, TreeMap highlights the three most frequent categories found in the system. This way, the user can infer that the repository separates three of its categories into groups of routines. This suggests a well modularized system structure.

To locate the functions used most often, activity iii uses the "by function" view of OctMiner. It revealed that the functions used most often are *cos* and *sin* (the two green rectangles at the top) as illustrated in **Figure 13**. Table 6 presents the most referenced functions from the repository.

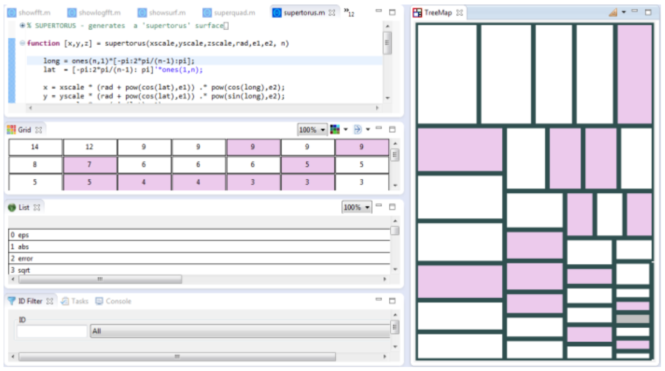


```
% SUPERTORUS - generates  a 'supertorus' surface

function [x,y,z] = supertorus(xscale,yscale,zscale,rad,e1,e2, n)

    long = ones(n,1)*[-pi:2*pi/(n-1):pi];
    lat  = [-pi:2*pi/(n-1): pi]'*ones(1,n);

    x = xscale * (rad + pow(cos(lat),e1)) .* pow(cos(long),e2);
    y = yscale * (rad + pow(cos(lat),e1)) .* pow(sin(long),e2);
```

14	12	9	9	9	9	9
8	7	6	6	6	5	5
5	5	4	4	3	3	3

```
0 eps
1 abs
2 error
3 sqrt
```

Figure 13. The Most Often Referenced Functions



```
% SUPERTORUS - generates  a 'supertorus' surface

function [x,y,z] = supertorus(xscale,yscale,zscale,rad,e1,e2, n)

    long = ones(n,1)*[-pi:2*pi/(n-1):pi];
    lat  = [-pi:2*pi/(n-1): pi]'*ones(1,n);

    x = xscale * (rad + pow(cos(lat),e1)) .* pow(cos(long),e2);
    y = yscale * (rad + pow(cos(lat),e1)) .* pow(sin(long),e2);
```

14	12	9	9	9	9	9
8	7	6	6	6	5	5
5	5	4	4	3	3	3

```
0 eps
1 abs
2 error
3 sqrt
```

Figure 14. Highlighting of Functions in Image Processing Repository

Table 6. Most referenced functions in the imaging repository

Number of Occurrences	Function	Category
14	cos	Mathematics
12	sin	Mathematics
9	error	Programming Scripts and Functions
9	axis	Graphics
9	size	Language Fundamentals
9	class	Advanced Software Development
9	strcmp	Language Fundamentals
8	figure	Graphics

In the Grid view from **Figure 13**, we learn that *cos* is used 14 times and that *sin* is used 12 times across the entire repository.

Activity *iv* aims to replicate the insight revealed in the second preliminary study. When the functions used most often belong to the same category, this may be an indicator of use of equivalent and/or complementary functions. In this case, we observed that functions *sin* and *cos* are complementary but can also be used together in a single expression. As expected, they are used to calculate sines and cosines.

Finally, activity *v* illustrated in provides indications that several distinct functions from the same category, spread throughout the repository, are auxiliary functions, e.g., function *max* (which computes the highest value from an array) or *min* (which returns the lowest), or plot (which presents values in a 2D perspective).

Regarding the research questions (**RQs**) for this work, we have the following considerations. **RQ1** – What are the concepts and/or entities in MATLAB/Octave programs whose comprehension can benefit from the support of *OctMiner*? The results of the conducted case studies and the strategies to use *OctMiner* provided initial evidences that performing comprehension activities based on crosscutting concerns (CCCs), tokens/functions, files and the repositories from which they belong can be fostered by the *OctMiner* usage. **RQ2** – To which extent *OctMiner* can provide support for the comprehension of MATLAB/Octave programs? The use of strategies such as those discussed in this paper can lead programmers to better contextualize their knowledge regarding the MATLAB/Octave programs as long as they use the concepts of CCCs and tokens/functions.

We recognize that *OctMiner* may not be able to provide support for all kinds of comprehension tasks. To better characterize and validate its range of applicability, we are planning new studies to have better knowledge regarding these limitations. 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.

VI. 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) Details of the design of the visual metaphors provided by *OctMiner* to convey real attributes of MATLAB/Octave routines; 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 different 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 [8] . A preliminary strategy based on *OctMiner* was presented at SEKE'2015 [10]. At ICCSA'2015 [9], we presented the validation case studies in detail. In the current paper, we consolidate the execution of all the steps planned in **Figure 1** and present a set of usage strategies with their respective examples of use.

We now plan to conduct a controlled experiment where undergraduate engineering students will perform comprehension activities with and without the support of *OctMiner*. We also plan to include new visual metaphors to explicitly represent the relationship and dependency among the tokens based on its usage in the routines. The version of *OctMiner* used in this paper is able to represent the intensity of use of a set of tokens, but not possible dependencies among themselves throughout the code.

The authors would like to thank the Brazilian Coordination for the Improvement of Higher Education Personnel (CAPES) for their financial support for this project.

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., Mendonça, 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] Kiczales, G.; Lamping, J.; Mendhekar, A.; Maeda; Lopes, C.; Longtier, J.M.; Irwin, J. Aspect-Oriented Programming. In: 11th European Conference on Object-Oriented Programming (ECOOP), 1997, Jyväskylä, Finland (pp.220-241).

[8] 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.

[9] Lessa, I.; Carneiro, G.; Monteiro, M.; Abreu, F. On the Use of a Multiple View Interactive Environment for MATLAB and Octave Program Comprehension. ICCSA (4) 2015: 640-654.

[10] Lessa, I.; Carneiro, G.; Monteiro, M.; Abreu, F. Scaffolding MATLAB and Octave Software Comprehension Through Visualization. SEKE 2015.

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

[12] 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).

[13] 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.

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

[15] *OctMiner* Website. Available at www.sourceminer.org/*OctMiner*

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

[17] Spence, R. Information Visualization: An Introduction. Springer; 3rd ed. 2014 edition.

[18] 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.

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