An Annotated Repository for MATLAB Code

António Relvas
NOVA LINCS
DI-NOVA/FCT
Portugal

Nuno C. Marques
NOVA LINCS
DI-NOVA/FCT
Portugal
Email: nmm@fct.unl.pt

Miguel P. Monteiro
NOVA LINCS
DI-NOVA/FCT
Portugal
Email: mtpm@fct.unl.pt

Glauco Carneiro
Universidade Salvador
UNIFACS
Brazil
Email: glauco.carneiro@unifacs.br

***Abstract*—Currently, there is the need for systems to manage repositories of MATLAB code bases capable of supporting global queries and feed their results to analyses components. Such features are not directly supported in current platforms. This paper presents a repository management system that supports queries over semi-automatically annotated code files and are able to associate them to higher level concepts. To meet this need, this paper proposes an approach that equips the repository with support for sophisticated queries over its stored code base and allows patterns to emerge from such queries, namely for visualisation and further analysis. This is achieved through the synergistic combination of a token-based metrics extraction component and a relational model fed by an ubiquitous data mining process. The code base is represented by means of relational knowledge, enabling intelligent queries that can be extended with new code metrics. Presently, query results are being used for the detection of concerns, including those whose code is scattered over multiple modular units. This paper outlines the proposed system's architecture and presents a proof-of-concept implementation developed for MATLAB programs. It is evaluated by means of a set of illustrative queries over a seed repository of MATLAB systems.**

***Index Terms*—MATLAB, Concern, Self Organizing Map, UbiSOM, Advanced Data Exploration, Software Repository Management System**

I. Introduction

The MATLAB language is known to lack support for fully fledged modules capable of enclosing most concerns typically present in MATLAB systems [1] [2]. Its basic units of modularity are *m-files* (MATLAB code files) and *toolboxes*, comprising folders of m-files and optionally sub-folders. The latter often correspond to standalone programs or libraries made available for the user community. Lack of modularity makes it hard to obtain well-organised code that is easy to read and reuse [1] [3]–[5]. These shortcomings motivated ongoing research to study the symptoms induced by the lack of modularity [3] and use of that knowledge for the detection of unmodularised concerns in existing systems [6] [5].

The above research on concern detection techniques is centred on the idea of decomposing m-files into its low-level elements [3] and derive a number of metrics based on that information [6] [5]. A number of analysis components can subsequently be plugged into the system to derive higher-level information. Further details are given in section II.

Developing and maturing the concern detection techniques entailed the assembling of repositories of many MATLAB systems in order to exercise and test functionalities. A number of metrics were derived, which were extracted from the code base by means of a tool that includes a lexical analyser for MATLAB [3] [6] [5]. Activities on the code repository included the performing of many kinds of search, e.g., to find new patterns, perform studies, check results. Initially, it was carried out by the metrics extraction tool. As search patterns tended to become increasingly elaborate and structured, the motivation for mounting the entire repository in a more intelligent system arose. This paper presents the outcome of that effort.

This paper presents a repository management system that exposes low-level data and allows the plugging of new analysis and visualisation components, which can be added over time. The repository management system could compute many of the queries previously supported by the lexical analyser tool and much else besides. It supports queries over automatically added annotations received from a data mining process and related annotations made by humans. Among other things, the system is able to expose associations between m-files based on higher level concepts.

Past work illustrated one potential use of the proposed repository through the implementation of a high level knowledge from a *Self Organizing Map* (SOM) data mining model that derives patterns from the code to enable the semi-automatic detection of (possibly unmodularised) concerns [6] [5]. By *semi-automatic*, we mean a process that starts with a machine learning classification phase, whose output can be corrected by a human or enriched by new annotations. We implemented a relational database and an associated web interface through which query results produced by the UbiSOM algorithm [7] are made available for further processing.

The rest of this paper is organised as follows. Section II describes the token-based technique used to decompose m-files. Next, section III describes the system's architecture and the relational model that enables a wide variety of queries over the code. Section IV illustrates how an external concern mining model can be used to automatically provide concern relevant tagging for all m-files in the repository. Section V presents a number of queries that illustrate the repository's capabilities. Section VI provides a short discussion of this work and section VII concludes the paper.

DOI reference number: 10.18293/SEKE2019-137

II. Use Case: Concern Detection

This section relates to past research on the detection of concerns in MATLAB code bases [3] [5] and describes an use case for the annotated repository.

A *concern* is any abstraction, concept or cohesive set of functionalities that would ideally be enclosed in its own module, for the sake of comprehensibility and ease of maintenance and evolution [8]. Ideally, each individual concern would map to a different unit of modularity (e.g., an m-file or a function in MATLAB), with each unit having a single, *primary* concern. However, several factors contribute to this not being so in practice. The limited and incipient nature of MATLAB's modules and limited programming experience from many users, among other issues, contribute to many concerns remaining unmodularised. As a consequence, potentially useful and reusable pieces of code are left *scattered* throughout a system's m-files and functions, and *tangled* with conceptually unrelated code. Scattering and tangling are the two dual symptoms usually observed in the code when modularity support is deficient [9] [8]. Tangling is particularly harmful to the comprehensibility of all concerns found in the modular unit, including the primary concern [1] [4] [3].

The concern detection technique explored in this paper bases the representation of an entire code base of a repository, comprising all systems stored in it, on the decomposition of each and every code file into *tokens*, i.e., the lexical elements extracted by means of the lexical analyser. The subset of tokens that are *words* (keywords and identifiers) plays an important role in the concern detection approach. It is based on the the idea that specific groups of word tokens can be associated to specific concerns, with individual tokens being associated to one concern at most. Patterns of occurrence of such tokens can be used to identify the presence of the associated concern in the code unit.

Presently, this approach is focused on function names, particularly names of functions from standard MATLAB libraries, because they are deemed more intention-revealing and are common to many MATLAB systems. Such names provide a measure of guarantee that the technique will operate uniformly in most systems. Programmer defined variable names are not currently considered because they are more variable across a repository of systems developed by many different teams.

The examples shown in section V serve as an illustration of the technique. They are focused on concern *verification of function arguments*, which relates to the processing that many MATLAB functions must carry out at the beginning to determine in which "mode" they were called (e.g., by finding out how many arguments it received). It is associated to a group of tokens that include `nargin`, `varargin` and `varargout`.

III. The relational model

This section describes a relational model for a MATLAB seed code repository. It is important to note that although the system presented in this paper was developed as a proof of concept focused on MATLAB systems, its design – described here – was created in view of covering a broader range of systems and programming languages.

The annotated repository was developed with the purpose of accommodating in one place all the data needed to perform exploratory analyses on MATLAB code, by means of advanced analysis components. The diversity and quantity of data being generated called for a powerful solution for data storage and query support. For this reason, a relational data model was adopted (e.g., [10]). It represents all data as *tuples*, grouped into relations. Note that *all* tokens are stored into the model: not just word tokens but also symbolic tokens, literals, etc. Such a model can easily be implemented in a relational database management system. A MySQL solution proved to be sufficient for the Web component of the system, while data analysis is based on snapshots of the systems in the repository, taken at the time of analysis of relevant information stored in a SQLite database.

The relational model represents the toolboxes, m-files and their complete contents (including comments, though presently they are not used). New systems, toolboxes and m-files can be added and similarly represented, including new versions of existing elements. Two main entities represent the organisation of all the systems into toolboxes and m-files and taking into account the possibility of multiple versions of these elements. Each m-file is in turn decomposed into code lines containing MATLAB code and thus stored. To reconstruct the original file (e.g., for visualisation or for some other subsequent processing), the model provides a separate entity to represent lines with comments only.

Tokens are the main subject of the analysis in this paper. Figure 1 depicts the main entities and relations for modeling the relation of `MATLAB` tokens and annotations within the repository according to the notation used by Silberschatz et. al. [11]. *Blocks* are intermediate entities grouping one or several lines and that are part of an m-file. Relations between toolboxes, m-files, code blocks, lines and tokens are simple one-to-many relations. For instance, a given line is always part of a block, which is always part of some m-file. Though an m-file is not necessarily organized into functions, we are mainly interested in that set of m-files since our focus is on MATLAB modularity. Under that view, blocks will always belong to some function, which in turn always belongs to some m-file. Decomposition of the entire code base along these lines requires just some basic parsing functionality added to the lexical analyser tool. Each block ends either with the `end` keyword or when another clearly defined block (e.g., a new function or control structure) begins. During design, we chose to also model each line of code in the repository as a tuple of the entity `Lines_mfiles` identified by its unique `line_id` identification code. The line number within the m-file and the `block` unique id are also unique identifiers for each line in the repository.

A token *instance* refers to individual occurrences of a given token. For instance, the various occurrences in the code of the `while` keyword can be said to be instances of the `while` token. The present token-based approach requires the

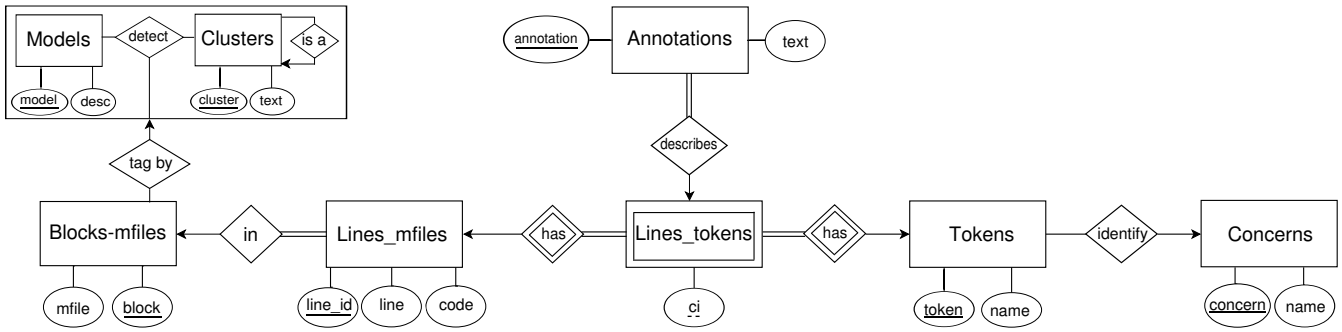

Fig. 1. Part of the system ER diagram for token related entities.

identification of each token instance, which is done by the lexical analyser tool. Each line comprises a *sequence* of token instances (order and position are important). Each non-empty line comprises at least one token instance or is a pure comment line in Fig. 1. Since a line can contain more than one instance of a given token, a weak entity (`Lines_tokens` in Fig. 1) is used to represent it. That weak entity is identified by the `line_id` of the `Lines_mfiles` entity and token position within the line (attribute `ic`). So, it is possible to know the line containing the token instance (attribute `line` from entity `Lines_mfiles`). This way, an indirect relation is made where each token instance is associated to its code or name. A token can also be associated to a given concern and a concern can be associated to several tokens — represented by the `identify` relation in Fig. 1. The design takes into account that future analyses may approach the code repository based on many different selection criteria. Annotations are also easily added using the `Lines_tokens` weak entity. This way a new entity `Annotations` is used so that users can add annotations when relevant `Lines_tokens` are found in Fig. 1. A simple annotation revision process is already supported: only accepted annotations will be shown in the final user interface.

During tool tests there was often a need to look for different combinations of two or more tokens or token-concern combinations. For instance, it was useful to find and visualize in what context some combinations occur in different m-files. The query can be done directly with a regular expression on the `code` field of entity `Lines_mfiles`, but this search is slow and slow to write. Besides, it will always be ineffective because limited to a single code line. As an alternative, the web interface performs a quick generation of queries in SQL (over the relational model). To obtain results quickly, the search is done on a first token after which an inner join is added for each additional token, of the resulting relation with itself. This way, it is possible to identify a sequence of elements (tokens or concerns) that are associated in the same m-file (through entity `Lines_tokens`). If they are relevant, extra constraints can be added to the query. For example, it is possible to specify a limit on the number of rows (or tokens) between the occurrences of both search tokens (the `ic` field of entity `Lines_tokens` is essential for a correct result).

The code base used as testing material for the present research originates from a repository of MATLAB programs and toolboxes originally assembled to test a compiler for MATLAB [12]. It comprises 35 193 m-files organised by toolboxes and covering various application domains, downloaded from *Sourceforge* and *GitHub* [12]. The repository was already used in previous work for concern mining [6] [5].

The output of a fully automatic concern mining component can be easily related with this system. The relational model aims to be generic and should support hierarchical unsupervised machine learning methods. The simultaneous reference to several concern mining models is supported by means of an aggregation between entities `Models` and its detected `Clusters`. Such aggregation can be used for tagging `Blocks-mfiles` entries with the cluster assigned by a model, resulting from a data mining process over the repository. Also some unsupervised learning methods discover models where clusters can be related with other clusters (e.g. in hierarchical clustering methods). This way, each block in the repository can be assigned to a cluster derived from the concern mining model and the resulting cluster can be a subcluster of another related cluster. Moreover, no restriction is made regarding sharing of clusters among models (this could be useful in situations where related models identify the same kind of clusters). Finally, `Block-mfiles` tagging can be continuously updated by a ubiquitous data mining process (such as illustrated in section IV) or can be directly assigned/revised by means of a human made model (which probably entails laborious manual m-file cluster identification and correction tasks). The concern mining model presented in previous work [5] is used in the illustrative results presented in section V.

IV. Validation using UbiSOM Output Models for Concern Mining

The system's design should provide for a continuous incoming stream and storage of MATLAB code files. Analysis and mining of its contents can be done based on ubiquitous data mining algorithms [13]. The UbiSOM algorithm was selected as an illustrative validation of this approach [5]. Appropriate support was developed for the continuous analysis

of data, approached as a data stream in which new m-files and toolboxes can be continuously added. The use of metrics to characterize the relevant blocks of code in the various m-files makes it possible to represent those blocks as a set of measures for different concerns, i.e., a set of feature-value pairs that is also stored in the database in a `patterns` entity. Each new set of such value pairs can be analyzed by means of the UbiSOM algorithm [7]. Note that each concern gives rise to its own specific value for each metric considered.

The UbiSOM component performs a continuous analysis of the data stream and maintains a SOM summarising all m-files in the repository and its contents, updating it whenever new contents are added. The relational model can deal with multiple SOM instances, all of which are represented in the database. This opens the way for (suitably trained) users to specify queries over the repository (using SQL) that also use SOM information to perform selections based of higher-level concepts. Note that each resulting SOM model and related query results can immediately become internally accessible to the system for subsequent processing.

The SOM model used consists of a fixed rectangular grid of units. Each unit can be seen as a generalisation of representations of sets of m-files with similar metric values – also called a *prototype* [14]. In the relational model, the `patterns` entity can also represent the various units of the SOM – again as sets of feature-value pairs. This way, for a given SOM model it is possible to associate each m-file to the SOM unit whose vector of metric values is closer to it (Euclidean distance is used for this purpose). The SOM community would call this unit the *best matching unit* (BMU) for that m-file. Various sub-sets of units in contiguous areas of the SOM, are also aggregated into *regions* of similar units in the SOM, whose information is stored as tuples in the database. Regions and units are represented in the entity `Clusters`.

V. Illustrative Queries

This section presents results that can be derived from mining the seed code base for higher level concepts. We call these *intelligent queries*. To facilitate comparison of results, we use an already published set of metrics and corresponding SOM model as an illustrative example [5] and which is copied into the database.

The set of concerns and related metric values are used as different dimensions or positions in the pattern vector fed to UbiSOM. The study refers to two disjoint clusters of m-files that were labelled as regions A_1 and A_3 in the original dataset [6] [5]. We retain the A_1 and A_3 labels in the database mainly to facilitate the task of readers wanting to make the connection with previous work [6] [5]. The description that follows does not depend on details from the other study. Though no m-file can belong to two clusters simultaneously, the previous analysis revealed the simultaneous presence of two or more concerns in those clusters [5]. This is a clear indicator of code tangling, which in turn is a clear indicator of deficient modularity [3] [6] [5]. From this, it can be concluded that the situation in which multiple concerns are found in the same m-file arises often. Several such cases are reported in that study [6] [5]. In it, A_1 and A_3 correspond to two SOM regions that represent two disjoint sets of m-files.

Fig. 2. Code view in the web system.

One of the concerns detected in m-files from the A_1 and A_3 clusters is *verification of function arguments*. It relates to functions that were prepared to be called in several different "modes", which are selected on the basis of the number of arguments that were passed upon its call. The previous study calls them *schizophrenic functions* [6] [5]. Typically, such functions use the `nargin` function from the MATLAB standard library and/or related functions. `nargin` returns the number of input arguments given in the call. A glimpse of code pattern based on calls to `nargin` is provided in Fig. 2 and Table II. In many cases, these `nargin` calls are made in a considerable number of points at the start of the code's schizophrenic function.

To illustrate the use of queries that join the relational representation of the SOM model with the data in the repository, a query is next shown, which returns the number of tokens associated to a given concern for each m-file covered any of the two regions A_1 and A_3, i.e., A_1 `or` A_3 [6] [5]. These regions represent the set of m-files that also give rise to high metric values relative to the concern *verification of function arguments*. Restricting the query to the set of m-files from regions A_1 `or` A_3, facilitates the analysis. Note that the restriction could be specified on the basis of *other* concerns (also restricted to A_1 `or` A_3 in this case). These are an examples of high-level restrictions that would be hard to express without the intelligent assistant for an annotated repository presented in the current paper.

TABLE I
Content (top 5 counts) higher level concepts query

Cluster	Sub-cluster	m-file	linesCount per m-file	VFACount per m-file
A_3	19	31424	779	202
A_3	19	12311	480	67
A_3	79	9252	950	49
A_1	739	30854	452	45
A_1	799	24100	162	40

Table I shows an output example of this query. It represents the dataset restricted to clusters A_3 or A_1 (column `Cluster`), SOM unit identifications (column `Sub-cluster`), the respective m-file id (column `m-file`), the number of lines of code of each m-file (column `linesCount`) and the number of tokens associated to concern *verification of function arguments* (column `VFACount`). After the query, we learn that the m-file with $id = 31424$ has 202 tokens associated to the concern under analysis (a significantly high value, with 26 occurrences per 100 lines of code). Table II shows a few code lines where token `nargin` occurs, in some cases complemented with the following lines for clarity. It is used in lines 185-189 and again in 195-200. In the second example, `nargin` checks whether the function receives zero arguments. In the third example, an error is issued in case the number of arguments equals 1.

TABLE II
SQL CODE PATTERN '%ARGIN%' IN M-FILE=31324

line	code
1	`function vargout = spm_sp(varargin)`

185–189	`if nargin == 0`
	`  error('Do what? no arguments given...')`
	`else`
	`  action = varargin{1};`
	`end`

195–200	`otherwise,`
	`  if nargin==1, error('No space : can''t do much!'), end`
	`  [ok,str] = spm_sp('issetspc',varargin{2});`
	`  if ~ok, error(str), else, sX = varargin{2}; end;`
	`end;`

The system's current implementation is a web prototype that pays special attention to code visualization. The code view enables token highlighting and provides pop-overs for various kinds of information as illustrated in Fig. 2. A frame is also available for searching *non-contiguous token sequences* in repositories as illustrated in Fig. 3. It also provides a *TreeMap* view (not shown) suitable for the visualisation of hierarchical data, e.g., *toolboxes* > *m-files* > *concern* [15], [16]. A visualization of the SOM model is also provided, which allows the selection of elements to derive a database relation of m-files belonging to specific SOM units and their regions.

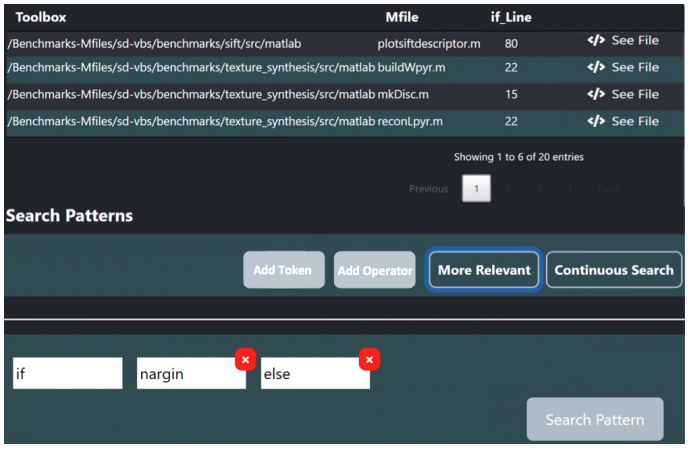

Fig. 3. Example of the web interface frame for token sequences.

VI. DISCUSSION

One may ask why a relational model was used instead of say a `noSQL`-type model. Granted, `noSQL`-type and graph databases may offer specific advantages in some cases. However, we opted for a relational model because token-based data comprises a significantly structured domain. The relational model facilitates integration with additional components and transaction support to cope with the addition of new annotations on the part of different users. It also facilitates efficiency gains for some queries, which are left for future work.

`Mathworks`[1] maintains the `MATLAB Central` website, which aims to provide the best support for MATLAB. It has a forum that is claimed to contain $110,000$ answered questions and a *File Exchange toolbox repository* for sharing code, where developers can easily import their toolboxes from `GitHub` [17]. As in every other software repository that we are aware of, the focus is on the toolbox. All toolboxes can be tagged and some of those toolboxes have online tutorials or even Webinars. There is also an area for MATLAB code examples, conveniently indexed by main topics in the language (e.g. matrixes and arrays) and highlighting to toolsets with example code. Each function also points to code examples. Unfortunately, MATLAB developers do not seem to have direct query access to that huge repository of code. We searched many repositories in several other major programming languages but failed to find a system managing a repository of MATLAB programs and toolboxes that supports global analyses and enables extraction of higher-level concepts. By contrast, this paper proposes a software repository enabling searches down to the level of tokens and which are still able to link results to the enclosing toolbox.

Tokens are usually defined as the smallest individual elements of any program. Queries over tokens allow the search of all the information in the repository. As token usages are also very diverse and case specific, there is a need for higher level searches, namely over concerns. However, there are too many possible types of tokens and token combinations. The use of concerns in queries allow for a more direct representation of the concepts involved in the reasoning of software developers when working on the code, thus bridging an important conceptual gap. To search for concerns, we need to resort to their manifestation in the code, by means of metrics on code patterns [5]. The present work is based on *concern metrics* and also relates to past research work on that category of metrics [18] [19].

It should be noted that the higher level concepts used in the illustrative queries are the combinations of concerns used in section V, which refers to our recent work on this front [5]. Presently, the database contains just the clusters and regions explained in that model [5]. Searches over the higher

[1]The leading company proprietary and developing MATLAB language.

level concepts with combinations of concerns proved very promising for several kinds of search regarding the concerns present in the code. However, the database and supporting site are more general. It is a relevant subject for this research to find other UbiSOM models (possibly with new metrics) that may answer different questions. The system is general enough to frame and annotate results from new knowledge discovery models and we are available to collaborate with the community to add new models to the system.

Past work used the UbiSOM algorithm to explore the related set of concern metrics [5], through which manual analyses of SOM results led to the detection of many cases of the joint occurrence of multiple concerns in a single m-file. However that method was still based on a generic data mining tool and unable to query a software repository. The repository here described uses the SOM as a data-mining tool to identify clusters of m-files having similar patterns regarding possible distinct sets of concern metrics. The SQL queries that can be devised over such clusters provide the advanced user with higher level concepts. Such queries result in sets of m-files available in the repository. A visual web interface allows end-users (namely MATLAB programmers) to search over relevant higher level concepts, such as the illustrative *schizophrenic functions* concept described in V.

VII. Conclusion

This paper proposes a software repository management system supporting intelligent queries over MATLAB code files and able to associate them to higher level concepts. This is achieved by the synergistic combination of a token extraction tool, a relational database and the advanced exploratory capabilities of a Self-Organizing Map. A web interface supports queries over the resulting knowledge stored in a relational model. The latter supports a token-based advanced exploration of the repository. Higher level concepts from SOMs – based on software concerns – can be used by programmers, by means of the web interface. A demonstration web site with full illustrative examples of such higher level concepts and supplementary material is available [2]. In addition, direct access to the repository database is freely available for research purposes.

Regarding future work, SOMs can be used for tackling problems other than those covered in this paper. SOM models are already available in our database, which opens the way for extending the present annotation with additional data mining processes. For instance, WEBSOM [20] is a classical use of SOM to document clustering. Line comments provide another interesting opportunity: if we approach them as documents, the joint inclusion of such a SOM in our database comprises a promising topic for future work on mining.

References

[1] J. M. Cardoso, J. M. Fernandes, and M. P. Monteiro, "Adding aspect-oriented features to matlab," in Fifth International Conference on Aspect-Oriented Software Development (AOSD 2016), 2006.

[2] T. Aslam, J. Doherty, A. Dubrau, and L. Hendren, "Aspectmatlab: An aspect-oriented scientific programming language," in Proceedings of the 9th International Conference on Aspect-Oriented Software Development, pp. 181–192, ACM, 2010.

[3] M. Monteiro, J. Cardoso, and S. Posea, "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, 2010.

[4] J. M. Cardoso, J. M. Fernandes, M. P. Monteiro, T. Carvalho, and R. Nobre, "Enriching matlab with aspect-oriented features for developing embedded systems," Journal of Systems Architecture, vol. 59, no. 7, pp. 412–428, 2013.

[5] N. Cavalheiro Marques, M. Monteiro, and B. Silva, "Analysis of a token density metric for concern detection in matlab sources using ubisom," Expert Systems, vol. 35, no. 4, 2018.

[6] M. P. Monteiro, N. C. Marques, B. Silva, B. Palma, and J. Cardoso, "Toward a token-based approach to concern detection in matlab sources," in proceedings of the 18th Portuguese Conference on Artificial Intelligence, pp. 573–584, Springer, 2017.

[7] B. Silva, Exploratory Cluster Analysis from Ubiquitous Data Streams using Self-Organizing Maps. PhD thesis, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, 12 2016. Manuscipt available at: http://hdl.handle.net/10362/19974.

[8] P. Tarr, H. Ossher, W. Harrison, and S. M. Sutton Jr, "N degrees of separation: multi-dimensional separation of concerns," in Proceedings of the 21st international conference on Software engineering, pp. 107–119, ACM, 1999.

[9] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin, "Aspect-oriented programming," in Proceedings of 11th European Conference on Object-Oriented Programming, pp. 220–242, Springer, 1997.

[10] E. F. Codd, "A relational model of data for large shared data banks," Communications of the ACM, vol. 13, pp. 377–387, June 1970.

[11] A. Silberschatz, H. F. Korth, and S. Sudarshan, Database system concepts. New York: McGraw-Hill, 6 ed., 2010.

[12] J. Bispo and J. M. P. Cardoso, "A matlab subset to c compiler targeting embedded systems," Software: Practice and Experience, vol. 47, no. 2, pp. 249–272, 2017.

[13] J. Gama, Knowledge Discovery from Data Streams. Chapman & Hall/CRC, 1st ed., 2010.

[14] T. Kohonen, "Self-organized formation of topologically correct feature maps," Biological cybernetics, vol. 43, no. 1, pp. 59–69, 1982.

[15] G. Langelier, H. Sahraoui, and P. Poulin, "Visualization-based analysis of quality for large-scale software systems," in Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering, pp. 214–223, ACM, 2005.

[16] B. Shneiderman, "Tree visualization with tree-maps: 2-d space-filling approach," ACM Transactions on graphics (TOG), vol. 11, no. 1, pp. 92–99, 1992.

[17] "Math Works matlab central website." https://www.mathworks.com/matlabcentral/. Accessed: 2019-01-22.

[18] E. Figueiredo, C. Sant'Anna, A. Garcia, T. T. Bartolomei, W. Cazzola, and A. Marchetto, "On the maintainability of aspect-oriented software: A concern-oriented measurement framework," in Proceedings of the 12th European Conference on Software Maintenance and Reengineering, pp. 183–192, IEEE, 2008.

[19] E. Figueiredo, C. Sant'Anna, A. Garcia, and C. Lucena, "Applying and evaluating concern-sensitive design heuristics," Journal of Systems and Software, vol. 85, no. 2, pp. 227–243, 2012.

[20] S. Kaski, T. Honkela, K. Lagus, and T. Kohonen, "Websom–self-organizing maps of document collections1," Neurocomputing, vol. 21, no. 1-3, pp. 101–117, 1998.

[2] http://bit.ly/MatlabAnnotatedRepository