Toward a Token-Based Approach to Concern Detection in MATLAB Sources

Miguel P. Monteiro[1], Nuno C. Marques[1], Bruno Silva[2], Bruno Palma[1], and Joao Cardoso[3]

[1] NOVA Laboratory for Computer Science and Informatics, DI-NOVA-FCT, Portugal
nmm@fct.unl.pt, mtpm@fct.unl.pt
[2] IPSetubal - Escola Superior de Tecnologia de Setubal, Portugal
[3] FEUP, Universidade do Porto, INESC-TEC, Portugal
jmpc@acm.org

Abstract. Matrix and data manipulation programming languages are an essential tool for data analysts. However, these languages are often unstructured and lack modularity mechanisms. This paper presents a business intelligence approach for studying the manifestations of lack of modularity support in that kind of languages. The study is focused on MATLAB as a well established representative of those languages. We present a technique for the automatic detection and quantification of concerns in MATLAB, as well as their exploration in a code base. Ubiquitous Self Organizing Map (UbiSOM) is used based on direct usage of indicators representing different sets of tokens in the code. UbiSOM is quite effective to detect patterns of co-occurrence between multiple concerns. To illustrate, a repository comprising over $35,000$ MATLAB files is analyzed using the technique and relevant conclusions are drawn.

Keywords: business intelligence; concern metrics; concern mining; MATLAB; token-based technique; Self-Organizing Maps; Modularity

1 Introduction

This paper presents a Software Engineering Business Intelligence and Knowledge Discovery approach for advanced exploratory data analysis using Ubiquitous Self Organizing Maps (UbiSOM) [12]. The UbiSOM main algorithm was initially proposed in 1982 [7] and has become an established data mining algorithm with hundreds of applications in many scientific domains. Here, UbiSOM is used for studying the symptoms of lack of modularity in the source code under analysis. It is based on a *concern metric* [4] that is used as an indicator of the intensity of the presence of a given concern in a code file and provides the foundation for the exploratory analysis of source code bases. UbiSOM analysis is particularly effective in detecting and representing patterns of co-occurrence of multiple concerns in the same file. The technique is scalable to large code bases. The present study is focused on MATLAB since it is a classical language for matrix manipulation and of which large repositories are publicly available.

The rest of the paper is structured as follows. Section 2 provides the background for this work as well as its motivation. Section 3 describes the proposed concern mining technique and presents a repository of MATLAB code, for which an illustrating analysis is described in Section 4. Section 5 provides a discussion and outlines related work. Section 6 concludes the paper and mentions several opportunities for future work.

2 Background and Motivation

A *concern* is any abstraction, concept or cohesive set of functionalities that ideally is enclosed in its own module, for the sake of comprehensibility and ease of maintenance and evolution. It has long been accepted that existing programming paradigms have limitations on the ability to enclose all concerns in separate modules [6]. The root cause is that each programming paradigm provides a *single* criterion to decompose a software system. Concerns that do not align with the primary decomposition tend to cut across the system's modular structure, even when developers follow the best practices of design and programming style.

These limitations were the subject of much study in object-oriented (OO) languages, but less so in the case of Matrix and data manipulation programming languages, of which MATLAB is an important example. MATLAB's support for modularity is less sophisticated than that of OO languages. Modules are mostly MATLAB files (m-files) and MATLAB functions (m-functions). Our study is focused on m-files with at least one m-function (i.e., we presently do not consider MATLAB *scripts*). UbiSOM analysis is particularly interesting in m-files with two or more concerns, which we approach as cases of a deficient support for modularity. We consider that such files contain a *core concern* plus one or several "additional" concerns.

Past research on techniques to detect unmodularized concerns in program code were carried out mainly under the umbrella name of *aspect mining* [5], which studies tools and techniques for the automatic or semi- automatic detection of unmodularized concerns in existing systems. The approach to concern detection proposed in this paper is a pioneer approach that uses unsupervised knowledge extraction tools as part of an effort to develop a general approach for concern detection that can be used equally well to detect modularized concerns enclosed in a single source code file or unmodularized concerns, which are therefore scattered across multiple files.

3 Concern Mining in MATLAB Systems

Concern detection builds on previous work by Monteiro et al. [11][2] which, to our knowledge, is the sole previous work on concern mining specifically tailored for MATLAB systems. The approach is based on the analysis *concern metrics* [4], which capture information about concerns present in one of more modules. This is in contrast to traditional modularity metrics [3], which capture information

on individual modules. Concern metrics are particularly suitable for supporting concern mining tasks.

The main information unit are the *tokens*, i.e., the lexical elements extracted from a code file by means of some lexical analyser tool. Monteiro et al. base their work on the hypothesis that specific groups of tokens can be associated to specific concerns, in which case patterns of occurrence of such tokens can be used to detect the presence of the corresponding concerns. Individual tokens must also relate to one concern at most.

This work includes a component that performs a tokenization of all the non-comment code from each m-file from a given target repository and computes a number of metrics based on the word tokens obtained. Nonword tokens (e.g., symbols and literals, including strings) are not considered in most cases. Treatment of keywords varies according to the specific aims. A final filtering phase yields just the words that are function names, discarding the rest. Function names - particularly from standard MATLAB libraries - provide stronger guarantees of uniformity than, e.g., local variables. These metrics are then directly used by an Ubiquitous Self-Organizing Map [12] component for assisting in the multi-dimensional exploration and analysis of the extracted data.

The proposed technique uses the trial mapping between concerns and function names shown in Table 1, which was proposed by a domain expert. One of the aims of the work described here is to test and assess the technique, using this mapping. The metrics used in this work are (1) *Lines of Code* (LoC), which counts all non-comment and non-blank lines of code for each m-file; and (2) *Token Density*, which computes, for each m-file, the total count of occurrences of the tokens from a given set (e.g., 5 occurrences of 'double' count as 5), divided by LoC. This metric represents the average number of tokens per LoC, for a given concern. It parametrises the concern, i.e., the specific set of tokens considered. The mapping from Table 1 gives rise to 10 instantiations of *Token Density*. Figure 1 shows the contents of an m-file (minus blank and comment lines) whose metrics stand out for a few concerns. With just 9 LoC, it has 2 tokens indicative of *Verification of function arguments and return value* (*Token Density* 0.2(2)), 9 tokens indicative of *Data type specialisation* (*Token Density* 1.0) and 5 tokens indicative of *Memory allocation/ deallocation* (*Token Density* 0.5(5)). *Token Density* is sufficient to enable the automatic detection of concerns. However, on a macro-level and just by itself, this technique is not well suited to provide a broad view of a large repository and provide a panorama of all concerns and its various co-occurrences.

The Self-Organizing Map (SOM) is an unsupervised learning artificial neural network model, based on competitive learning [7]. Standard SOM model consists of a set of topologically ordered data prototypes arranged in a rectangular lattice (the map). The SOM is widely used as a tool for projecting high-dimensional data onto the two-dimensional representation map. This projection retains the relationship between input data as faithfully as possible, thus describing a topology-preserving projection of input similarities in terms of distances in the output space. It is this special projection that, when a large enough map is used (also

Verification of function arguments and return value	nargchk, nargin, nargout, nargoutchk, varargin, varargout
Data type specialization	double, fi, fimath, int16, int32, int64, int, quantize, quantizer, sfi, single, ufi, uint16, uint32, uint64, uint
Data type verification	cast, class, intmax, intmin, isa, isboolean, iscell, ischar, $\cdots$ isquantizer, isreal, isrow, typecast, wordlength
Dynamic properties	eval, evalc, evalin, inline, feval
Console messages	annotation, assert, disp, display, error, last, lastwarn
Visualization	aaxes, axis, box, clf, close, errorbar, Figure, $\cdots$ plot, plot3, plotedit, rectangle, title,ylabel, zlabel, zoom
File I/O	diary, fgetl, fgets, fileformats, fopen, fprintf,fread, fscanf, fwrite, hgload, hgsave, load, save, saveas, uisave
System	batch, break, clear, clock, cputime, date, $\cdots$ toc, unloadlibrary, wait, weekday, who, whos, xbreak
Memory allocation/ deallocation	delete, global, ones, persistent, zeros
Parallelization	cancel, codistributed, codistributor, createParallelJob, createTask, defaultParallelConfig, demote, $\cdots$ sparse, submit, subsasgn,subsref, taskFinish, taskStartup

Table 1. Illustrative mapping between concerns and tokens (some tokens removed due to space concerns).

known as Emergent SOM), establishes as a valuable data exploratory analysis tool with special visualizations [13]. This property also makes SOM distinct from other clustering algorithms. De per si, the traditional k-means clustering is unable to compare distinct clusters and a bad setting of the k parameter could result in too generic clusters, i.e, group sets of data that may have little relation. By contrast, SOM is less "eager" in clustering together the data it processes, i.e., it significantly reduces the risk that it will group data too much. That is one reason we deem SOM more suitable for exploratory data analysis.

The proposed method uses an extension of the SOM technique called Ubiquitous SOM [12] for training maps of *Token Density* data. This technique has an easier and better defined parameterization than standard SOM and allows the detection of meaningful variations among the data, which is approached as a stream. The classical *online* SOM decreases learning parameters monotonically throughout time and requires the same observations to be presented several times, consequently considering the underlying distribution *stationary*. UbiSOM [12] also uses the classical *online* SOM update rule but with different mechanisms for estimating SOM learning parameters for non-stationary datastreams. The UbiSOM algorithm switches between an *ordering* and *learning* state. The latter is only achieved when the current distribution of the data stream is already sufficiently modeled after a previous *ordering* phase. Preference for UbiSOM [12] is justified by its dynamic visualization and data exploratory

advantages over other traditional data mining tools. Moreover, UbiSOM ensures detection of distinct phenomena if new or too distinct repositories are used. This is particularly relevant if, as in this case, more code repositories are intended to be added later on.

The UbiSOM for exploring *Token Density* relations is composed of units where each position (or unit) is a vector of average values representing the *Token Density* metrics. So, each concern in a similar enough set of m-files (or micro-cluster) is associated to a particular unit / map position. This way, the method enables analyses of emergent regions of *Token Density*, enabling exploratory data analysis for concern detection and correlation in the map and dissimilar regions to detect uncommon but relevant correlation patterns.

```
function fired = nemoStep(fstim, istim_nidx, istim_current)
  if nargin < 1
    fired = nemo_mex(uint32(12), uint32(zeros(1,0)), uint32(zeros(1,0)), zeros(1, 0));
  elseif nargin < 2
    fired = nemo_mex(uint32(12), uint32(fstim), uint32(zeros(1, 0)), zeros(1, 0));
  else
    fired = nemo_mex(uint32(12), uint32(fstim), uint32(istim_nidx), istim_current);
  end
end
```

[ ] – *Verification of function arguments and return value;* [ ] – *Data type specialization;* [ ] – *Memory allocation/ deallocation;*

Fig. 1. m-file showing high values for *Token Density* in relation to 3 concerns.

The repository used for this work is one used to test the MATLAB compiler by Bispo and Cardoso [1], comprising 35,193 files organized by toolboxes and covering various application domains. 28k m-files were downloaded from Sourceforge and 2k m-files were downloaded from GitHub. Of these, 784 m-files were discarded due to the lack of useful MATLAB code inside them. For instance, we found a number of m-files with zero LoC, as they contained only comment text. After this curating, the repository comprised 34,409 m-files.

We found that in practice, m-files with too low a LoC value tend to hog results for *Token Density*. Since such small m-files are usually uninteresting, the tool was set to discard all m-files with $LoC < 5$, leaving a little over $30k$ m-files. According to the data gathered, only a small percentage of the m-files from the repository does not contain any of the concerns from Table 1. Out of the m-files with at least one concern, the biggest chunk is for m-files with *two* concerns, followed by m-files with a single concern.

4 Data Analysis and Exploration

The SOM component of our tool processes the *Token Density* data from each m-file from the repository. Our method is specifically focused on co-occurrence patterns among the various concerns from Table 1. Each pattern is formed as a vector $X = [x_{1k}, x_{2k}, \cdots, x_{Nk}]$, where each component of this vector is calculated by equation (1), where x_{ik} is the component for concern i for m-file k.

$$x_{ik} = \frac{log(\frac{TkDensity_{ik}}{max_c(TkDensity_{ic})} \times N + 1)}{log(N+1)} \quad (1)$$

Each component of the vector corresponds to a row number i in Table 1, so $i = 1$ to NC, where NC=10 is the total number of concerns considered for analysis. $TkDensity_{ik}$ is the corresponding *Token Density* for concern i and m-file k and $max_c\bullet$ is the maximum of previous values of concern i for all m-files. Intuitively, the use of the logarithm of a $[0; 1]$ normalized density (i.e., the division in the numerator of equation) is needed since the *Token Density* values should be made relevant for small values of the metric and zero for zero values of the metric. The multiplication of this quantity by a factor N is used to ensure that low-frequency values of *Token Density* are distinct enough from zero. We have set $N = 6000$ after empirically testing several alternatives. The histogram with frequency analysis of distinct *Token Densities* for $N = 6000$ is shown in Figure 2. Division by the maximum value ensures that each SOM component value is always normalized between zero and one (as required by SOM).

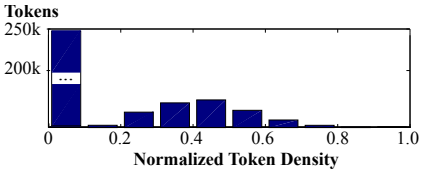

Fig. 2. Full repository normalized token density, when $N = 6000$.

Several distinct UbiSOMs were trained by filtering patterns with three or more concerns. Note that filtering improves the quality of the map, but reduces the sample for detecting patterns. Therefore, in a systematic approach, m-files with less than two simultaneous concerns with above-zero *Token Density* were filtered away and several UbiSOMs were trained with the resulting datasets. Patterns filtered with three simultaneous concerns, i.e., by selecting only the m-files with three or more above-zero *Token Density* values, were used to train a SOM that is used next, to illustrate the technique and the insights it provides. Final iterations of UbiSOM training used random resampling over the full repository. The SOM quantization error was convergent with a good reduction of quantization error during model training (final quantization error = 0.05 per component). Since no big variations were detected on the final resampling step, the current repository seems stable enough for model convergence.

The visual outputs from SOM considered in this paper comprise two components [12][9]: the *Unified Distance Matrix* (U-Matrix) and the various *Component Planes* (CP). In each CP and the U-Matrix, the same areas in the planes represent the same units, i.e., regions in the data that tend to have high or low values together. Throughout the rest of the paper, the regions from Figure 4 denoted A1 to E2 also refer to the equivalent regions of the various planes from Figure 4. However, it is important to note that the cells and colours of U-Matrix and CPs

are interpreted in different ways and that regional commonalities between CPs must be interpreted in light of the specific colour pattern found in the corresponding region of the U-Matrix. It is the combination of both views that yields a proper understanding of the full dataset and their underlying patterns. The U-Matrix provides a general view of the units or patterns found in the data, while the various CPs provide the details of the various units. Here, each CP relates to one specific concern.

The average values in both the U-Matrix and the CPs are represented as a JET colour map [9], meaning that red represents high values, blue represents low values (down to zero) and green represents intermediate values. In addition (and consistently), green areas with a blue hue represent values lower than pure green, while "dirty" green areas with a red hue represent values higher than pure green. In the U-Matrix, each unit measures the average Euclidean distance of the correspondent SOM unit vector to the vectors of the eight SOM unit vectors that are its immediate neighbours. In the CPs, each position represents the density parameters for its respective concern.

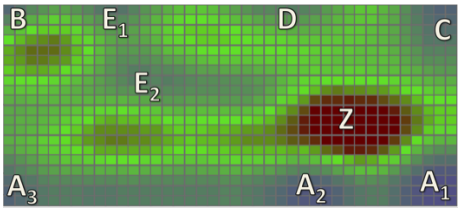

Fig. 3. U-Matrix representing the selected m-files in the MATLAB repository. The various letter annotations highlight relevant regions.

Figure 3 shows an U-Matrix for the metrics data fed to the SOM, from the example used in the rest of this paper to illustrate and summarize our analyses. In general, well-separated regions (i.e. the blue "lakes" [9]) in the U-Matrix represent clearly separated patterns in the data fed to the SOM. The regions labelled A1, A2, A3 and C from Figure 3 are good examples. In the U-Matrix, blue regions represent homogeneous regions in the SOM while red regions (e.g., region Z from Figure 4) denote high variability among units. Green regions represent intermediate values for distance among SOM units. Blue regions in a slight green hue such as B, E1, E2 and D from Figure 3 also represent relatively homogeneous regions. Several well-defined blue and light green "lakes" can be discerned by their colour uniformity regarding neighbourhood units. These various data regions betray several well-defined concern patterns.

To analyse the data obtained for the various concerns, we look at their respective CPs (Figure 4). For instance, we see that concerns *Data type verification* and *Verification of function arguments* share two common red regions in their respective CPs (at the bottom, to the left and right). This means that there is a significant co-occurrence between those two particular concerns. This sharing is

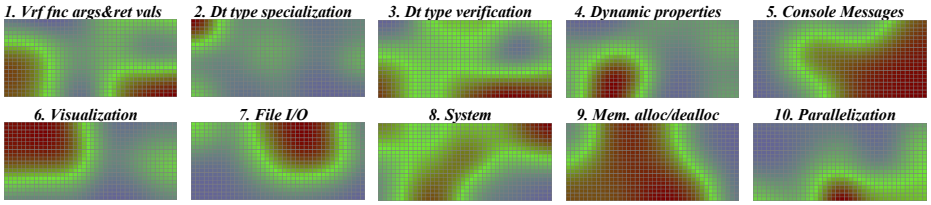

Fig. 4. Component planes showing the colour maps of the *Token Density* metric for concerns from Table 1.

particularly significant, because those regions are blue in the U-Matrix, meaning that the sharing is particularly uniform across the analysed data. The two concerns co-occur with high densities, though we can also see a significant overlap with concerns *Console messages* and *Memory allocation/deallocation* in regions A1 and A2.

A sharing of red regions taking place in a red rather than blue region of the U-Matrix (no such case in the present example) would depict more variable distances between units and would mean that patterns in such regions would be much more diverse (with high, low and intermediate densities). The overlap would probably not be representative of the data as a whole. It is also significant that no "blue lake" from Figure 4 is devoid of at least one CP with high values in the same region, i.e., all red zone in the CPs correspond to blue lakes in the U-Matrix. If such region existed, it would represent some diffuse concept, possibly meaningless.

The red region in the CP for concern *Data type specialization* completely overlaps with that for concern *Visualization* (region B) though the latter's region is wider. This could suggest that in the majority of m-files where *Data type specialization* has a significant presence, *Visualization*'s presence is significant as well. There is also a relevant presence of this concern in area A2, jointly with concerns *Data type verification* and *Verification of function arguments and return value*. However, *Data type specialization* does not have enough above zero values to enable a reliable characterization in the CP. For this reason, it was discarded from subsequent analyses. A similar overlap of red regions also occurs with *Console messages* and *System* (region), though the overlap is less complete in this case.

We performed several manual checks on several m-files to confirm that red regions from the various CPs do indeed relate consistently to high values of *Token Density* with respect to their respective concerns. One example is m-file `merge_options.m` (shown in Figure 5 without comment lines) - one the m-files indicated by the common blue region A1 of the U-Matrix (Figure 3) and selected as a small representative from a shortlist of m-files with high density values for more than one concern. As expected, those m-files show several tokens associated with the concerns found in region A1, particularly *Verification of*

function arguments and return value and *Data type verification* but also *Console messages*.

M-file `merge_options.m` is an interesting illustration of the attempt to separate concern *Verification of function arguments and return value* from other concerns. Its sole purpose seems to test the way the function is called and configure a call to another function that provides the functionality proper. However, that does not prevent `merge_options.m` being a clear case of code tangling in MATLAB. Throughout our work, we found several examples of this symptom, in functions that do not separate argument-checking from other concerns.

5 Discussion and Related Work

Token Density is sufficient to directly allow the selection of the top densities on any individual concern from Table 1. However, it cannot be scaled to provide an overview of a large repository spanning multiple concerns.

The SOM model presents large distances between many units, which jointly with the presence of some quantization error, suggests a good enough convergence of the model but does not yet represent some patterns in the data, i.e., the boundaries separating clusters in Figure 4 could be "crisper". Nevertheless, the observation that most CPs have clear continuous regions in several trained SOMs, points us to a clear and concise division of the studied concern patterns under analysis. Overlapping areas in planes from the CPs (Figure 4) are related primarily with co-occurring concerns. This is a clear indicator that the groups of tokens from Table 1 are effective in identifying the concerns to which they are associated and that those groups of tokens cover distinct programming considerations (i.e., concerns).

SOM use is marred by a high number of zeros in the *Token Density* data. The traditional technique for handling sparse data would be Principle Component Analysis (PCA). However, PCA is not discriminating enough for our purposes, since it aggregates the several concerns in different variables. Namely, the tests performed using PCA for our problem did not improve the outcome, be in terms of the SOM, be in not reducing the quantization error. The high number of zeros of *Token Density* seems to be an instance of an exponential law on the token occurrence. A better alternative would be to use a method such Word2Vec [10], a natural language method that can build high-quality distributed vector representations that capture precise syntactic and semantic relationships. However, for a more rigorous evaluation we chose to use information provided by an expert (cf. Table 1).

The $N = 6000$ parameter used in equation 1 is an efficient way to give proper relevance to low *Token Density* values. M-files with one concern occurrence per 60 LoC have a normalized value of 0.5, while one concern occurrence per 200 LoC (normalized value of 0.3). Even in more than 1000 LoC (normalized value lower than 0.2) are still detectable. Preliminary experiments done without the exponential normalization (cf. equation 1) showed a large uniform region and few co-occurring concerns. Indeed, SOM training is known to be sensible to too many

```
function options = merge_options(default_options, varargin)
  if ~isempty(varargin) && mod(length(varargin),2) == 0
    options = merge_structs(struct(varargin{:}),default_options);
  elseif length(varargin)==1 && isstruct(varargin{1})
    options = merge_structs(varargin{1},default_options);
  elseif ~isempty(varargin)
    error('matlag_bgl:optionsParsing',...
     'There were an odd number of key-value pairs specified');
  else
    options = default_options;
  end
end
```

[ ] – *Vrf. of func. Args. and return value;*

[ ] – *Data type verification;*

[ ] – *Console messages;*

Fig. 5. m-file with high *Token Density* values relative to three of the analysed concerns.

zeros. This problem is seen for concerns with a high number of zeros such as, e.g., *Data type specialization*, which tends to adjust with high error and therefore tends to appear opportunistically in several areas of the space. Manual analysis nevertheless shows that the red and green areas in this concern's CP (Figure 4) are related with m-files relevant for this concern. Restricting the selection to m-files with more co-occurring concerns reduced this problem.

In our illustrative example, a particularly relevant result from the SOM analysis is the high correlation of the densities for concerns *Verification of function arguments* and *Data type verification* (see Figure 4). An illustrating example of an m-file with these concerns is shown in Figure 5. The high correlation between both these parameters is common in many analysed m-files. This result suggests that MATLAB is somewhat problematic as regards validating and verifying function arguments or data types. The problem with arguments is caused by the practice of "schizophrenic functions" that need to test multiple variations of its parameters to learn in what "mode" they were actually called. In object-oriented languages, method overloading enables the tackling of the same problem with better separation of concerns. As regards data types, Cardoso et al. propose aspect-oriented extensions to MATLAB to enable a precise control of data types without incurring the symptom of code tangling [2].

Considering the large percentage of m-files from the analysed repository that betray the presence of multiple concerns, we can conclude that the code tangling symptom is quite common.

The present work is based on concern metrics and therefore is related to past research work on that category of metrics [4]. A work with several similarities to ours is that by Maisikely and Mitropoulos [8], whose purpose is to develop an aspect mining technique that uses a SOM component to process metrics data to find similarities. It differs from ours in that the target language is Java and the mining is specifically tailored to the kind of CCCs that AspectJ is good at modularizing. As it would be expected, the metrics used are also quite different. All of those metrics reside at method level and the group seems to be centered in two dynamic (i.e., captured at runtime) versions of *method Fan-In* and *method Fan-Out*. To complement, other method-level metrics are used, namely: *Information flow* (Fan-In*Fan-Out), *Method Signature* (method name, parameter types and their order, visibility modifiers and return type), *Method Internal Coupling* and *Method External Coupling*. Maisikely and Mitropoulos report positive results from an experiment with their technique, using as target systems

JHotDraw 5.41b and a small system comprising 6 classes. The present work uses token-based metrics because they seem more appropriate for a first experiment for aspect mining of MATLAB system, as argued in previous work [11].

6 Conclusions and Future Work

This paper makes the following contributions:

1. Proposes the *Token Density* metric as a direct development of some ideas proposed by Monteiro et al. [11].
2. Proposes a technique for exploratory analysis of MATLAB code, scalable to large repositories. The technique provides a precise way to identify patterns of occurrence and co-occurrence of concerns in (possibly large) repositories of MATLAB files.
3. The analysis described in this paper serves as a first validation of the *Token Density* metric.

We identify some of the opportunities for future work. Next, we highlight a few of them.

Automatize the development of the concern-token mapping. We can use a distributional representation of tokens [10] as a direct input to the UbiSOM. Finding out the best way to automatically learn this representation from MATLAB code bases is an avenue left open by this work. In future, we will further mature this technique in at least two directions: (1) automatize or semi-automatize the discovery of new, distinct patterns of occurrence that lead to new concerns and new entries for Table 1; and (2) automatically or semi-automatically refine existing entries of the Table, by applying a more fine-grained, token-level analysis that indicates the tokens that should be kept in the mapping – because patterns of occurrence confirm their usefulness – and those that should be discarded – because patterns of occurrence are too sparse and/or do not add meaningful information.

The results achieved with the proposed method are promising. The interactive and exploratory nature of SOM allows a focused search for specific concerns in code repositories. In future, this will be part of a tool for finding relevant code snippets in MATLAB repositories. There is an increasing trend in big data analytics using matrix based analysis (or N-dimensional matrices, i.e., tensors). This can be seen by the increasing number of packages and tools for matrix and data handling, e.g., Google TensorFlow, the numPy package in Python and the very relevant statistical analysis tools and languages such as SPSS and R. We plan to make a social collaborative web site with such a tool as a useful way for reusing and improving previously available code. This site will also be a forum for exchanging good practices on development of data analysis software. Empirical data collected from this site promises be a powerful means to develop this line of research.

References

1. Bispo, J., Cardoso, J.M.P.: A matlab subset to c compiler targeting embedded systems. Software: Practice and Experience 47(2), 249–272 (2017), http://dx.doi.org/10.1002/spe.2408, sPE-15-0162.R2
2. Cardoso, J.M., Fernandes, J.M., Monteiro, M.P.: Adding aspect-oriented features to matlab. In: Fifth International Conference on Aspect-Oriented Software Development (AOSD 2016) (2006)
3. Chidamber, S.R., Kemerer, C.F.: A metrics suite for object oriented design. IEEE Transactions on software engineering 20(6), 476–493 (1994)
4. Figueiredo, E., Sant'Anna, C., Garcia, A., Bartolomei, T.T., Cazzola, W., Marchetto, A.: On the maintainability of aspect-oriented software: A concern-oriented measurement framework. In: Software Maintenance and Reengineering, 2008. CSMR 2008. 12th European Conference on. pp. 183–192. IEEE (2008)
5. Kellens, A., Mens, K., Tonella, P.: A survey of automated code-level aspect mining techniques. In: Transactions on aspect-oriented software development IV, pp. 143–162. Springer (2007)
6. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.M., Irwin, J.: Aspect-oriented programming. ECOOP'97—Object-oriented programming pp. 220–242 (1997)
7. Kohonen, T.: Self-Organizing Maps. Springer-Verlag Berlin (2001)
8. Maisikeli, S.G., Mitropoulos, F.J.: Aspect mining using self-organizing maps with method level dynamic software metrics as input vectors. In: Software Technology and Engineering (ICSTE), 2010 2nd International Conference on. vol. 1, pp. V1–212. IEEE (2010)
9. Marques, N.C., Silva, B., Santos, H.: An interactive interface for multi-dimensional data stream analysis. In: Information Visualisation (IV), 2016 20th International Conference. pp. 223–229. IEEE (2016)
10. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Advances in neural information processing systems. pp. 3111–3119 (2013)
11. 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. Citeseer (2010)
12. Silva, B., Marques, N.C.: The ubiquitous self-organizing map for non-stationary data streams. Journal of Big Data 2(1), 1–22 (2015)
13. Ultsch, A., Herrmann: The architecture of emergent self-organizing maps to reduce projection errors. In: Proceedings of the European Symposium on Artificial Neural Networks (ESANN 2005). pp. 1–6. Verleysen M. (Eds) (2005)