

Analysis of a Token Density Metric for Concern Detection in Matlab Sources using UbiSOM

Nuno C. Marques^{1*} | Miguel P. Monteiro^{1*} | Bruno Silva^{2*}

¹NOVA Laboratory for Computer Science and Informatics, DI-NOVA-FCT, Portugal

²DSI, ESTSetúbal — Instituto Politécnico de Setúbal, Portugal

Correspondence

Miguel P. Monteiro, NOVA Laboratory for Computer Science and Informatics, DI-NOVA-FCT, Portugal
Email: mptm@fct.unl.pt

Present address

[†]Departamento Informática, NOVA-FCT, Quinta da Torre, 2829 -516 Caparica, Portugal

Funding information

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.
conflicts of interest: none

Matrix and data manipulation programming languages are an essential tool for data analysts. However, these languages are often unstructured and lack modularity mechanisms. This article presents a knowledge discovery approach for studying manifestations of the lack of modularity support in that sort 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 and their exploration in a code base. The Ubiquitous Self Organising Map (UbiSOM) is used to perform exploratory data analysis over concerns detected in a, possibly changing, repository of Matlab files. The UbiSOM is quite effective in detecting patterns of co-occurrence of multiple concerns. To illustrate the technique, a repository comprising over 35 000 Matlab files is analyzed. The results show that the use of *Token Density* metrics in conjunction with UbiSOM enables the detection of patterns of co-occurrence of multiple concerns in m-files.

KEYWORDS

business intelligence, concern metrics, concern mining, Matlab, token-based technique, self-organising maps, modularity

Abbreviations: UbiSOM, Ubiquitous Self-Organising Map; CCC, Crosscutting Concern; m-file, Matlab file; m-function, Matlab function; LoC, Lines of Code; U-Matrix, Unified Distance Matrix; CP, Component Plane

* Equally contributing authors.

1 | INTRODUCTION

This article presents a Software Engineering Business Intelligence and Knowledge Discovery approach for advanced exploratory data analysis using the Ubiquitous Self-Organising Map (UbiSOM) (Silva and Marques, 2015). Human-computer interaction is used for tuning UbiSOM to the analysis of a source code base, resulting in an expert systems technique for applying *concern mining* to study manifestations of lack of modularity support in Matlab code bases (m-files). The developed technique is based on a *concern metric* that is used as an indicator of the intensity of the presence of a given concern in each source code file.

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 concern metric provides the foundation on which exploratory analyses of source code bases can be performed – in this case, using the UbiSOM. UbiSOM’s underlying algorithm was initially proposed in Kohonen (1982) and became an established data mining algorithm with hundreds of applications in many scientific domains (Kohonen, 2001). The technique is scalable to large code bases. The UbiSOM model analyses proved to be particularly effective in detecting and representing patterns of co-occurrence of multiple concerns in the same source file.

The remaining sections of this article are structured as follows: Section 2 provides the background and motivation for this work. Section 3 describes the proposed concern mining technique and presents a repository of Matlab code. Section 4 presents the UbiSOM and its interactive interface, MultiSOM. Next, an illustrating analysis using the MultiSOM tool over the Matlab repository is described in section 5. Section 7 provides a discussion and outlines related work. Section 9 concludes the article 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 (Kiczales et al., 1997). The root cause is that each programming paradigm provides a *single* criterion to decompose a software system (Tarr et al., 1999).

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. Such non-aligning concerns are known as *crosscutting concerns* (CCCs) (Kiczales et al., 1997; Tarr et al., 1999). In modern software such as object-oriented systems, the usual symptoms of the presence of a CCC in source code are *scattering* and *tangling*. Scattering usually takes the form of code fragments scattered across multiple source files, often corresponding to repeated instances of “boiler plate code”. Tangling is found in the modules that the CCCs overlap: code pertaining to the primary concern appears intertwined with code pertaining to other concerns. Tangling is particularly harmful to the comprehensibility of all concerns found in the modular unit, including the primary concern.

Past research on techniques to detect unmodularised concerns in program code were carried out mainly under the umbrella name of *aspect mining* (Kellens et al., 2007), which studies tools and techniques for the automatic or semi-automatic detection of unmodularised concerns (as latent aspects) in existing systems. Though such techniques were the subject of much research in the context of object-oriented systems and systems coded in C, they were less so in the case of Matrix and data manipulation programming languages, of which Matlab is a prominent example. Matlab’s support for modularity is less sophisticated than that of object-oriented languages. Modules are mostly Matlab files (m-files) and Matlab functions (m-functions).

An m-file that encloses zero m-functions is a Matlab *script*. Our study is focused on m-files with at least one m-function, as we presently do not consider scripts. Whenever more than one concern is found in a given m-file, it contains a *core concern* plus one or several *secondary concerns*, which are the equivalent of CCCs in Matlab. Such "additional concerns" give rise to the Matlab versions of scattering and tangling, which can take extreme forms in some cases (Cardoso et al., 2006). The UbiSOM analysis is particularly interesting in m-files with two or more concerns, which we approach as cases of a deficient support for modularity.

The approach to concern detection proposed in this article 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 modularised concerns enclosed in a single source code file or unmodularised concerns, which are therefore scattered across multiple files.

Many instances of the past work on aspect mining were carried out with the aim of subsequently migrate the identified CCCs into aspect modules. The present work is not related to any existing aspect-oriented extension of the Matlab language, though such extensions have been proposed in the past (Cardoso et al., 2006; Aslam et al., 2010; Cardoso et al., 2013). In fact, the approach proposed here can be used to detect *any* kind of concern, crosscutting or not. Work is also presently being carried out to develop the technique into a general approach for concern detection that can be used equally well to detect concerns enclosed in a single source code file (e.g., an m-file) or CCCs scattered across multiple files.

It should be noted that the idea of aspect mining can serve purposes other than concern mining, as the information obtained through such techniques is useful and interesting in its own right and it can form the basis for a number of other tasks, namely assessments of the target system in terms of e.g., architectural soundness, quality of design and good programming style. For instance, certain aspect mining techniques are based on well-known metrics, which retain the usefulness associated to their original purpose (Marin et al., 2004).

3 | CONCERN MINING IN MATLAB SYSTEMS

Concern detection builds on previous work by *Monteiro et al.* (Monteiro et al., 2010; Cardoso et al., 2006) 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* (Figueiredo et al., 2008), which capture information about concerns, whether they are modularised or not. This is in contrast to traditional modularity metrics (Chidamber and Kemerer, 1994), which capture information on the modules themselves. 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 analyzer 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. In this approach, individual tokens must be associated to one concern at most.

The present work is based on a tool that includes a lexical analyzer for Matlab, plus components for extracting metrics from repositories of m-files, as well as an analysis component that uses Ubiquitous Self-Organising Maps for assisting in its multi-dimensional exploration and analysis of the extracted data. The tool materialises the above approach for concern mining by performing a tokenisation 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. All non-word tokens (e.g., symbols and literals, including strings) are discarded. The treatment of keywords varies according to the specific aims, though as a general rule they are discarded.

For the current study, the tool is also set to filter out all names that are not function names (mostly variable names

in practice). In addition to a lexical analyser, the tool was also provided with basic parsing functionality to identify the *lvalue* part of assignments, which comprise the standard way to create variables in Matlab. At present, all names tagged as *lvalues* are discarded and the remaining names are assumed to be function names. Function names, particularly from standard Matlab libraries, comprise the most useful tokens because they are common to most or all Matlab systems, thus providing a measure of guarantee that the technique will operate uniformly in most Matlab systems. Variable names are discarded because they look less promising: a given variable name found in different systems, created by different teams, is likely to mean different things.

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 computed by the tool and used in this work are: (i) *Lines of Code* (LoC), which counts all non-comment and non-blank LoCs for each m-file; and (ii) *Token Density*, which computes, for each m-file, the total count of occurrences of each instance 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. Note this metric parameterises 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)).

The *Token Density* metric 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, though it can be used as a direct selection criteria on m-files based on a single concern. This simple method already proved effective in tasks such as e.g., spotting examples such as those shown in Figure 1. However, on a macro-level and just by itself, a technique based on *Token Density* is not well suited to provide a broad view of a large repository and provide a panorama of all concerns from Figure 1 and its various co-occurrences.

```
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 specialisation;  – Memory allocation/ deallocation;

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

4 | ADVANCED DATA EXPLORATION WITH THE UBIQUITOUS SELF-ORGANISING MAP

The Ubiquitous Self Organising Map (UbiSOM) is a variant of the well-known Self-Organising Map algorithm (SOM) (Kohonen, 2001). While the later was conceived for *static* data, the former is tailored for real-time analysis of streaming data (Silva and Marques, 2015).

The UbiSOM reported in this article is used to obtain models of *Token Density* data. UbiSOM also ensures detection

Concern	Tokens
Verif. func. args & return values	nargchk, nargin, nargsout, nargsoutchk, varargin, varargout
Data type specialisation	double, fi, fimath, int8, int16, int32, int64, int, quantize, quantizer, sfi, single, ufi, uint16, uint32, uint64, uint
Data type verification	cast, class, intmax, intmin, isa, isboolean, iscell, ischar, iscolumn, isempty, isfi, isfield, isfimath, isfixed, isfloat, isinf, isinfinite, isinteger, islogical, isnan, isnumeric, isobject, isquantizer, isreal, isrow, isscalar, isstr, isstruct, isvector, length, ndims, numel, range, realmax, realmin, size, typecast, wordlength
Dynamic properties	eval, evalc, evalin, inline, feval
Console messages	annotation, assert, disp, display, error, last, lastwarn
Visualisation	aaxes, axis, box, cla, clabel, clf, close, datacursormode, datetick, errorbar, figure, figurepalette, fplot, gca, gcbf, gcbo, gco, getframe, gplot, grid, gtext, hist, histogram, hold, imfinfo, ishold, legend, line, loglog, mesh, meshgrid, newplot, pan, plot, plot3, plotbrowser, plottedit, plottools, plotyy, polar, propertyeditor, rectangle, reset, rgbplot, scatter, semilogx, semilogy, showplottool, subplot, surf, texlabel, text, title, xlabel, ylabel, zlabel, zoom, set, rotate, rotate3d, imformats, imread, imwrite, movie, image, frame2im, im2frame, VideoReader, VideoWriter
File I/O	diary, fgetl, fgets, fileformats, fopen, fprintf, fread, fscanf, fwrite, hgload, hgsave, load, save, saveas, uisave
System	ans, echo, exist, inmem, input, inputname, inputParser, isglobal, iskeyword, isvarname, mexext, mfilename, namelengthmax, pcode, symvar, who, whos, systems, slist, where, loadlibrary, mex, calllib, libisloaded, unloadlibrary, libfunctionsview, onCleanup, clearvars, rehash, pack, memory, clear, addtodate, now, weekday, date, calendar, tic, toc, dbcont, dbquit, dbstop, dbmex, ba, bafter, break, ebreak, nanbreak, rbreak, tbreak, run, xbreak, zcbreak, wait, stop, batch, spmd, pause, step, next, mlock, munlock, mislocked, clock, cputime, etime, start, startat, timerfind, timerfindall, profile, profsave
Mem. alloc./dealloc.	delete, global, ones, persistent, zeros
Parallelisation	gplus, gather, distributed, poolStartup, mpiSettings, promote, psave, gpuArray, gpuDevice, gpuDeviceCount, importParallelConfig, isreplicated, jobStartup, labBarrier, labBroadcast, labindex, labProbe, matlabpool, mpiLibConf, mpiprofile, numlabs, parfor, pctconfig, pctRunDeployedCleanup, pctRunOnAll, pload, pmode, ...

TABLE 1 Illustrative mapping between concerns and tokens.

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.

4.1 | The Ubiquitous Self-Organising Map

The UbiSOM is used as a tool for projecting high-dimensional input data onto a 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.

The UbiSOM retains all the properties of the original SOM algorithm, and, therefore, its visualisation capabilities for exploratory data analysis. It is an unsupervised learning artificial neural network algorithm, whose model consists of a fixed rectangular grid of data prototypes — the *map*, that form the projection layer for input data. Starting from a set of random initialised data prototypes, the algorithm continuously processes input patterns: for each input pattern a competitive learning rule selects the "closest" prototype — the "winner" or best matching unit (BMU), and both the BMU and its neighbouring prototypes are adjusted toward the input pattern by a gradient descent mechanism. Upon presentation of a sufficient number of diverse input patterns, a topological feature map is formed by the algorithm.

What distinguishes the UbiSOM from its original counterpart is its capability to adjust the model to new input patterns without the need to rebuild the entire model. Although it still uses the original *online* SOM update rule, the annealing learning parameters used in the gradient descent mechanism are estimated based on the error of the current model against the new input patterns, as opposed to simply monotonically decreasing them. This allows for a more "natural" and rapid convergence to *stationary* streams, while incorporating a form of neural *plasticity* to deal with new information, i.e., *non-stationarity*.

Comparison to K-means Clustering

The competitive learning scheme and gradient descent should resonate with those familiar with the *k*-means algorithm, although, and this is a critical difference, *k*-means only adjusts the *centroid*. Despite this close relation, the best way of using both algorithms in data mining is different.

In the *k*-means clustering algorithm the number of *K* clusters should be chosen according to the number of clusters present in the data. However, *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., data groupings that may have little relation.

On the other hand, in the UbiSOM (and in the SOM, in general) the number of data prototypes should be chosen to be much larger, irrespective of the number of clusters (Ultsch, 1995). Doing so, the cluster structures and descriptions can become visible by means of special powerful visualisation techniques to perform exploratory data analysis (Ultsch and Herrmann, 2005).

Hence, these capabilities for data clustering and pattern analysis without any prior assumptions on data form the reason why we deem the UbiSOM more suitable for our exploratory data analysis.

4.2 | Data Visualisations

UbiSOM visualisations are only possible due to the topological ordering of the prototypes, input density matching and the fixed-sized lattice of the UbiSOM. UbiSOM visualisations use colours as a visual representation of specific values — different colour scales can be used. This makes them available not only to experts, but laymen, when analysing them and can be easily understood if one is familiar with the representation. The basic visualisations that can be derived from an UbiSOM model are the following:

Component Planes. By *component plane* representation, we can visualise the relative component distributions of the input data. Component plane representation can be thought as a sliced version of the UbiSOM model. Each component plane (CP) has the relative distribution of the values of one feature. In this representation, and using a temperature-like colour scale, “cooler” colours represent relatively small values while “warmer” colours represent relatively large values. Using the analogy with the JET colour map of Matlab¹, “cooler” colours are encoded as blue and “warmer” colours are encoded as red. Correlations between features (components) can be inferred by visual inspection, e.g., if the outlook of two CPs is similar, the corresponding features correlate; if they seem like the “negative” of each other, then they are inversely correlated. Several examples of this method are given in section 5.

Unified Distance Matrix. The *unified distance matrix*, or simply *U-Matrix*, presents distances between data prototypes. The distances between adjacent data prototypes are calculated and presented with different colourings between the adjacent data prototypes. Following the same temperature-like colour scale, a warmer colouring between data prototypes corresponds to a large distance and thus a gap between the prototype values in the projected space; a cooler colouring means prototypes are close to each other in the input space. Cooler areas can be thought as clusters and warmer areas as cluster separators. This is specially powerful to understand the underlying structure of data, i.e., when one tries to find clusters in the input data without having any a priori information about the clusters. Consequently, the detection of complex clusters is achieved not by regarding single units, but by regarding the topological structure map.

Hence, while CPs allow discovery of relationships among features, the *U-Matrix* allows the visual discovery of clusters of data. Used in conjunction, one may infer which features best describe the detected clusters, i.e., a description of the clusters.

It must be emphasised that this knowledge discovery through visual exploration is the main motivating factor for using the UbiSOM in this article, as these visualisations are leveraged to perform data analysis of the cluster structure of detected concerns and the co-occurrence of concerns.

5 | METHODOLOGY FOR DATA ANALYSIS AND EXPLORATION

The *token density* metric was calculated for all files in an available m-file code repository and then values for distinct sets of concerns were prepared for UbiSOM analysis. Several possible relevant combinations of the concerns from Table 1 were repeatedly tested with the available repository.

5.1 | Matlab Code repository under analysis

The repository used for this work is the one used to test the Matlab compiler by Bispo and Cardoso (Bispo and Cardoso, 2017), comprising 35 193 files organised by toolboxes and covering various application domains. 28 000 m-files were downloaded from *Sourceforge* and 2 000 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*. Such small m-files are usually of limited interest but even one or two tokens make them yield a relatively high value for the metric. For this reason, the tool was set to discard all m-files with $LoC < 5$, leaving a little over 30 000 m-files.

¹<https://www.mathworks.com/help/matlab/ref/jet.html>

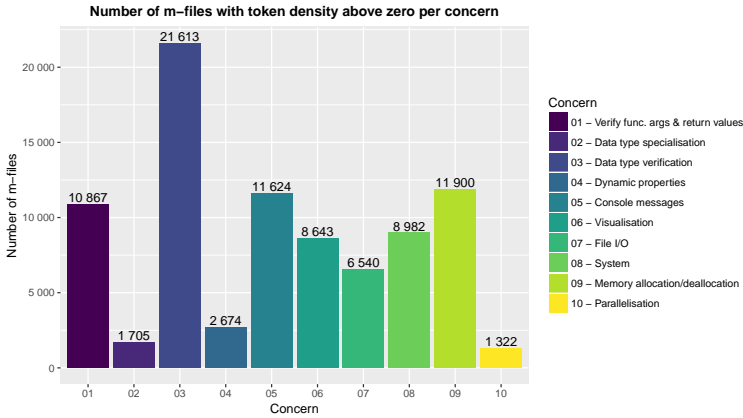


FIGURE 2 Number of *m*-files per concern.

Only about 9% of the *m*-files from the curated repository does not contain any of the concerns from Table 1. Out of the *m*-files with at least one such concern, the biggest chunk is for *m*-files with two concerns, followed by *m*-files with a single concern. Figure 2 compares the number of *m*-files with *Token Density* greater than zero. Each bar represents the number of *m*-files having each concern. Note that three concerns (*Data type specialisation*, *Dynamic properties* and *Parallelisation*) are representative of less than 3000 *m*-files (i.e., less than 10% of the *m*-files under analysis).

5.2 | Experimental Setting

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. So, there is one input pattern per different *m*-file. The data fed to the UbiSOM comprises a sequence of lines, with each line (or pattern) being an instance of the vector defined as $X = [x_{1k}, x_{2k}, \dots, x_{Nk}]$. Each component of this vector is obtained by computing Equation 1 for every x_{ik} , which is the component for concern i and *m*-file k .

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

Each component (feature) 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. $TkDensity_{y_{ik}}$ is the corresponding *Token Density* for concern i and *m*-file k and \max_c is the maximum of previous values of concern i for all *m*-files. Intuitively, the use of the logarithm of a $[0; 1]$ normalised density, i.e., the division in the numerator of Equation 1, is needed since the *Token Density* values should be made relevant for small values of the metric and zero for zero values. The multiplication of this value by a factor N is used to ensure that low-frequency values of *Token Density* are distinct enough from zero. We have set $N = 6\,000$ after empirically testing several alternatives. The histogram with frequency analysis of distinct *Token Densities* for $N = 6\,000$ is shown in Figure 3. Division by the maximum value ensures that each UbiSOM component value is always normalised between zero and one (as required by UbiSOM).

We used the baseline parameterisation for the UbiSOM algorithm provided in Silva (2016), namely map size of 20×40 , $\eta_i = 0.1$, $\eta_f = 0.08$, $\sigma_i = 0.6$, $\sigma_f = 0.2$ and $(T = 2\,000, \beta = 0.7)$.

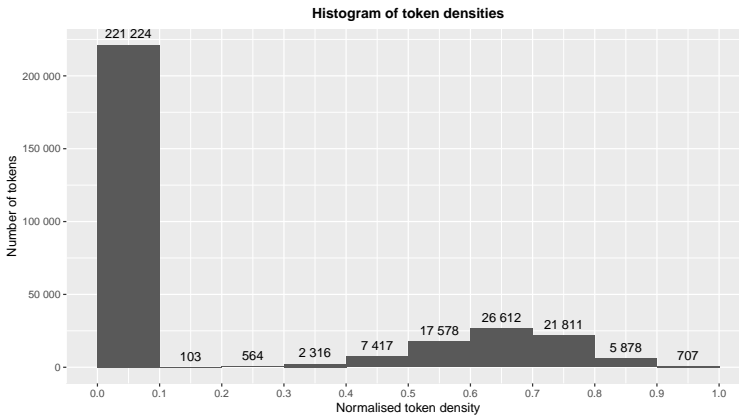


FIGURE 3 Full repository normalized *Token Density*, when $N = 6\,000$.

As the UbiSOM algorithm processes these input patterns of token densities per m-file, it generates a model consisting of a topologically ordered map of prototype patterns. Hence, each prototype (map unit) is a vector of averaged *Token Density* values obtained across all processed patterns. This topological ordering of prototype patterns – within a sufficiently large map, enables exploratory data analysis through the type of model visualisations described in Section 4.2, i.e., the detection of direct and inverse correlations between token densities (by the *Component planes* visualisation), and therefore concerns, as well as the detection of clusters of similar patterns (by the *U-Matrix* visualisation).

6 | RESULTS

Due to the stream based nature of UbiSOM, after proper model convergence, each newly presented m-file will result in a new valid UbiSOM model. By processing the available repository a valid stable model is obtained. The results of our work include both the newly developed visual tools for concern analysis and the observed stable model representing the selected m-file repository.

6.1 | An Advanced Data Analysis Tool based on the UbiSOM

We previously developed **multiSOM** (Marques et al., 2016), an interactive visualisation tool that uses the UbiSOM as the underlying data mining algorithm. For the purposes of the analysis reported in this article, the previously available tool received a few improvements, including commands providing new visualisation functionalities. All are meant to allow better control and exploration of the training process by an expert data analyst, who should be able to better tune the UbiSOM algorithm's parameters to a given particular analysis purpose. The relevant details are given next.

Multidimensional projection of an UbiSOM

The first improvement is the capability of either simulate the data stream as a random presentation of the m-files from the code repository, or to directly feed the map with fresh data received from the input data stream, e.g., if new files are added or updated in a Matlab toolbox (a set of m-files) or even if a new Matlab toolbox is inserted. This addition comprises an effective way of seeing the effect of a particular datastream input change, for example by considering

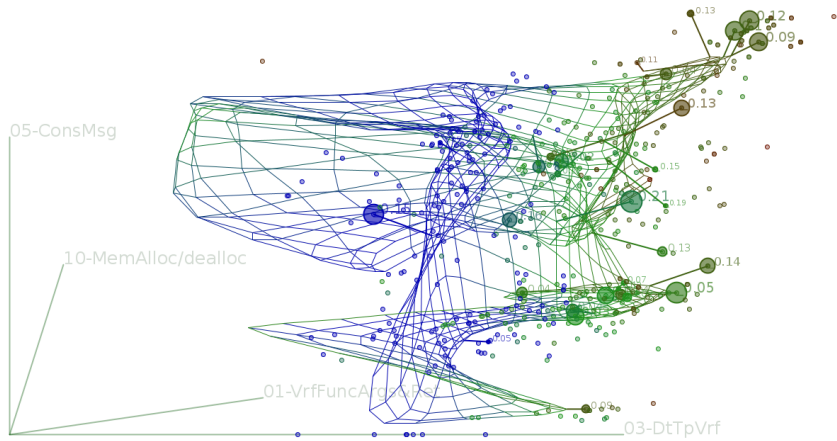


FIGURE 4 MultiSOM visualisation of the UbiSOM grid with primary projection on *Console messages vs. Data type verification concerns*. *Memory allocation& deallocation* and *Verification of function arguments* are used with weaker relevance.

the toolbox order of m-files in the training process. Also, the system developed for the present case study is capable of dynamically include or exclude different model concerns (features) through an explicit request from the data analyst. When a given concern is excluded from the model, all future m-file values for the corresponding concern metric are returned with a value that is equal to the average value of that metric in all units of the current UbiSOM model. This way, the data analyst can easily explore different configurations and see how training is influenced by the various concern metrics in the target code repository (by excluding or re-including features).

The newly added interactive commands that influence the UbiSOM training are feasible, because the UbiSOM can dynamically adapt its model to shifting concepts. Thus, the current interface for *multiSOM* is able to dynamically change the training mode upon each data analyst request. This way, it paves the way for an easier assessment of the relevance of each training feature (namely the concern metrics) during UbiSOM training. Moreover, since the dynamic change of the model being learned is possible, it is also particularly important to continuously monitor the learning process of the model. For that purpose, the data analyst can continuously see a n-Dimensional projection of all patterns (or m-files) in the feature space to the 2D output screen. For the present case study, the n-dimensional projection system proposed in (Marques et al., 2016) was further improved to present the input stream most recent m-file names and their BMU projections in the UbiSOM map. The data analyst can continuously change the relevance of each projection axis by visually dragging the projection weight of each concern. Also important is the continuous monitoring of two UbiSOM error measures: the average *topological error* and average *quantisation error* presented in (Silva and Marques, 2015; Silva, 2016). Both are continuously updated and presented to the data analyst regarding the last 5000 UbiSOM iterations by using two standard error graphs.

Figure 4 shows an illustrative value of the current training UbiSOM's interactive monitoring dashboard. Most recent patterns (m-files) are projected by small circles from n-dimensions to the two-dimensions of on-screen visualisation where the larger circles represent the most recent points. Then UbiSOM grid is visualised as connections among all prototype values, also projected into the output screen.

This allows an immediate relation with the U-Matrix: nearby neurons (in the projection) naturally correspond to near-by neurons (or similar points) in the U-Matrix, so the areas where there are more sample points are usually

corresponding to clusters. Online rotation of the multi-dimensional projection (by dragging the projection axis) allows an interactive zoom to such specific areas. The way the UbiSOM model (i.e., the projected grid) models the global dataset enables a synergy between the trained data analyst and the UbiSOM algorithm. Finally, each input pattern can also be associated with each filename and the corresponding concern metric values. At any time, the user can see current values of both error rates on a second screen (not shown).

Tensor Visualisation with an U-Matrix and Component Planes

Two additional visual outputs for a given UbiSOM model are considered in this article and comprise the two types of model visualisations described in Section 4.2: the U-Matrix and the various CPs. Both of these visual tools are extensively applied in the analysis of the concerns under study and are later presented both using U-Matrix and CPs (Sections 6.2 and 6.3).

At any given instant, a *tensor visualisation* can be used to inspect the UbiSOM model (Figure 4), which can be seen as composed of several CPs, i.e., a three dimensional matrix where lines and columns correspond to the traditional 2D SOM and depth corresponds to the several features. So a slice of this tensor represents a CP: an individual map for each input feature.

6.2 | Analysis of All Concerns

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 size 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 UbiSOM whose analysis is presented next, to illustrate the technique and the insights it provides. Final iterations of UbiSOM training used random resampling over the full repository. The UbiSOM quantisation error was convergent with reduction of quantisation error during model training (final quantisation error of 40%). Since no big variations were detected on the final resampling step, the current repository seems stable enough for model convergence.

The U-Matrix illustrated in Figure 5 and its various CPs illustrated in Figure 6 will be used to illustrate the UbiSOM results for this experiment. As usual, throughout the rest of the article, the regions from Figure 5 denoted A_1 to E_2 also refer to the equivalent regions of the various planes from Figure 6.

In our case study, the features are the concerns, so in the CPs, each position represents the *Token Density* for its respective concern. In addition, we can also measure the average distance among each map position to its immediate neighbours. That measure is represented in the U-Matrix.

In the U-Matrix, each unit measures the average Euclidean distance of the correspondent UbiSOM unit vector to the vectors of the eight UbiSOM unit vectors that are its immediate neighbours. Thus, 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. 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.

As previously explained, the average values in both the U-Matrix and the CPs are represented as a JET colour map (Marques et al., 2016), with red representing high values, blue representing low values (down to zero) and green

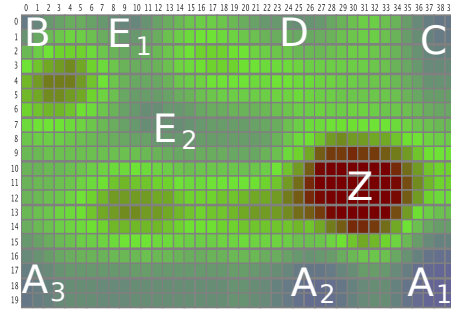


FIGURE 5 U-Matrix representing the relations among concerns in the Matlab repository. The various letter annotations highlight relevant regions.

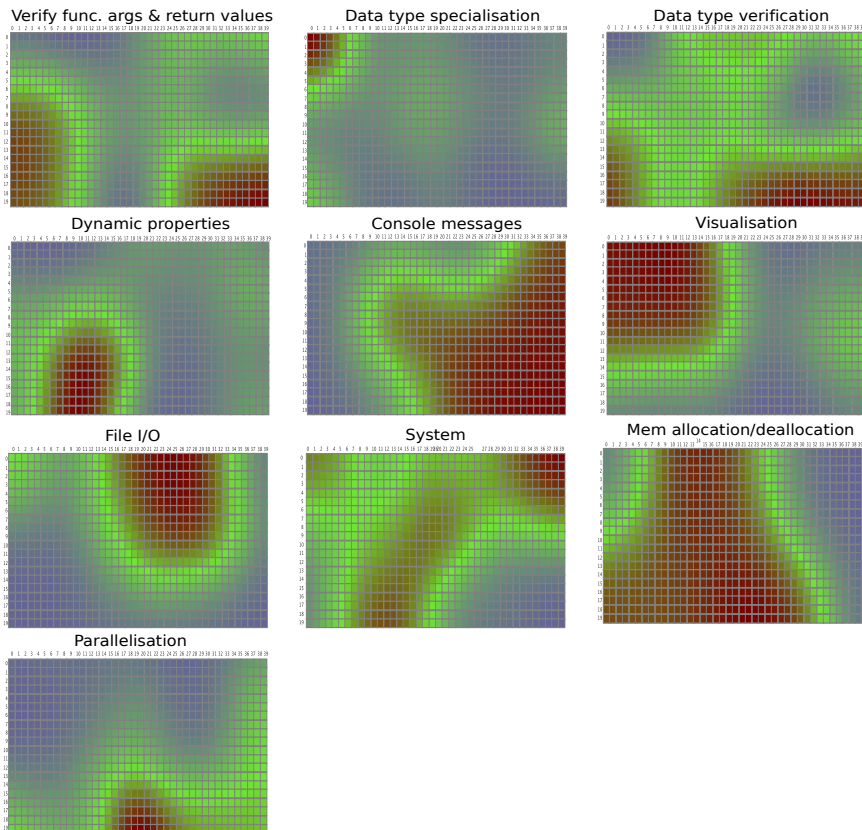


FIGURE 6 Component planes showing the colour maps of *Token Density* for concerns from Table 1.

representing 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.

Figure 5 shows an U-Matrix for the metrics data fed to the UbiSOM, from the example used in Section 6.2 to illustrate and summarise the analyses. In general, well-separated regions (i.e. the blue “lakes” of Marques et al. (2016)) in the U-Matrix represent clearly separated patterns in the data fed to the UbiSOM. Individual lakes are semi-automatically labeled using a derivation of the method described in Matos et al. (2015). During analysis, the user can select a given neuron on the U-Matrix, in which case all the units that are similar enough (up to a given user predefined threshold) will be automatically selected for the same region. The regions labeled A_1 , A_2 , A_3 and C from Figure 5 in Section 6.2 are good examples.

In the U-Matrix, blue regions represent homogeneous regions in the UbiSOM. Blue regions in a slight green hue such as B, E_1 , E_2 and D from Figure 5 also represent relatively homogeneous regions. Red regions (e.g., region Z from Figure 5) denote high variability among units. Green regions represent intermediate values for distance among UbiSOM units. Several well-defined blue and light green “lakes” can be discerned by their colour uniformity regarding neighbourhood units. These various data regions expose several well-defined concern patterns.

To analyze the data obtained for the various concerns, we look at their respective CPs (Figure 6). 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 particularly significant, because those regions are blue in the U-Matrix, meaning that the sharing is particularly uniform across the analyzed 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 A_1 and A_2 .

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). In such cases, an overlap would probably not be representative of the data as a whole. If such region existed, it would represent some diffuse concept, possibly meaningless. It is also significant that no “blue lake” from Figure 5 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.

The red region in the CP for concern *Data type specialisation* completely overlaps with that for concern *Visualisation* (region B) though the latter’s region is wider. This could suggest that in the majority of m-files where *Data type specialisation* has a significant presence, *Visualisation*’s presence is significant as well. There is also a relevant presence of this concern in area A_2 , jointly with concerns *Data type verification* and *Verification of function arguments and return value*. However, *Data type specialisation* does not have enough above zero values to enable a reliable characterisation 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 corresponding concerns. One example is m-file `merge_options.m` (shown in Figure 9 without comment lines) - one the m-files indicated by the common blue region A_1 of the U-Matrix (Figure 5) 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 A_1 , 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.

6.3 | Analysis of Selected Concerns

In previous results all concern metrics were approached as equally interesting features and therefore no feature selection was carried out. However, further analysis revealed that some concerns appear in very distinctive patterns, meaning that their presence was potentially dividing the whole data set into two big chunks as well as hiding less distinctive, but also potentially more interesting patterns. For example, in Figure 4, the UbiSOM grid is clearly divided in two by the *Console messages* concern. *De per si*, such division is not particularly interesting and the high-low values division for the metric seems to hide several high density areas of points that are similar on both low values and high values of the *Console messages* metric. A similar behaviour was also found on the *Visualisation*, *File I/O* and *System* metrics. So, although the relations among some of these metrics may be interesting for future work, we used the *exclude feature* command of our multiSOM tool (Section 6.1) to look for relations among the remaining concerns under study².

The various experiments used the multiSOM on the available repository and yielded a subset of concerns from Table 1. However, for better comparison, the results presented in this article were acquired starting the UbiSOM training from a randomly initialised map using the subset of concerns since the first iterations of UbiSOM. that corresponds to the component planes in Figure 8 added with a (meta-)measure called *norm Count₀*. This new (meta-)measure was inserted since the number of zeros was a potentially relevant measure of how many concerns are being used in the same Matlab file. So, *norm Count₀* is a normalised counter of how many concern metrics are different from zero in a given Matlab file. Since m-files with a concern metric smaller than 2 had already been excluded from our dataset, a zero in this normalised metric corresponds to three concern metrics. M-files with up to 8 of the concerns from Table 1 can be found in the repository under study and so, have a value of *norm Count₀* = 1.0 (this includes the concerns that were excluded for this experiment).

Once again, the UbiSOM algorithm used a sequential presentation of all available input samples for 7 000 iterations, and a random sampling over available repository during 36 000 iterations. Similarly to the previous experiment, both the measures and visual inspection of the map grid were stable from iteration 10 000 up to iteration 43 230 (when the training of UbiSOM was stopped). We have selected the snapshot at iteration 43 230 as a representative of the full Matlab code repository. The UbiSOM selected snapshot has a quantisation error of 8.3% and a topological error of 4.6%.

All UbiSOM component planes (Figure 8) and the corresponding U-Matrix (Figure 7) are presented for the UbiSOM snapshot. We can immediately notice four main clusters in the U-Matrix, corresponding to blue regions *M*, *N*, *O*, *P* e *Q* of Figure 7. Those regions mostly correspond to clear representative and non-representative values for *Token Density* of the concerns under analysis. The characterisation of each cluster is described in Table 2.

From the analysis of Table 2, it is noticeable that region *N* has a very well defined behaviour (i.e., it forms a clear cluster). In this cluster, the joint co-occurrence of high values for *Token Density* of concerns *Verification of function arguments and return values*, *Memory allocation/ deallocation* and *Data type verification* are observable. Moreover, this region has some distinct low values for the normalised count for zeros metric (*norm Count₀* CP of Figure 8) in the top right of this region (i.e. there is a small overlap of the blue/green area in *norm Count₀* CP with the U-Matrix top left of blue area *N*). So, depending on the region *N* sub-area, there are both m-files with few concerns on the top right part of the region (probably just the identified constraints) and m-files with many simultaneous concerns (remaining part of the region).

²The manual feature selection of multiSOM could have been done a priori to training and directly on the input data, but with an exponential number of possible combinations for excluding features, the graphical interface was indeed a valuable tool for testing different feature combinations.

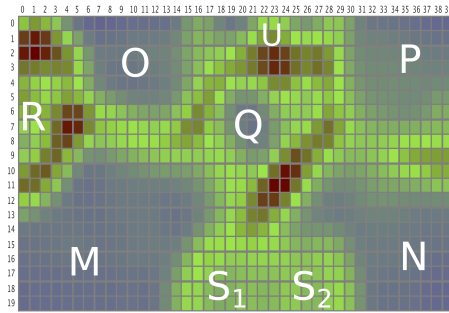


FIGURE 7 U-Matrix representing the selected concerns components. The various letter annotations highlight relevant regions.

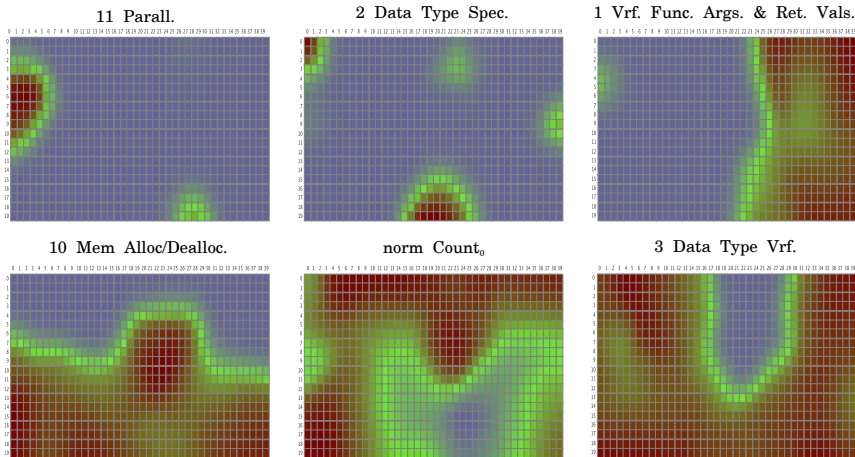


FIGURE 8 Component planes showing the colour maps of the *Token Density* metric for selected concerns.

Concern	M	N	O	P	Q	R	S ₁	S ₂	U
Parallelisation	N	N	N	N	N	R	N	R	N
Data type specialisation	N	N	N	N	N	N	R	S	N
Verify func. args & return values	N	R	N	R	N	S	S	R	S
Memory allocation/deallocation	R	R	N	N	R	S	R	R	N
Data type verification	R	R	R	R	N	R	R	R	N

TABLE 2 Relation among representative values of *Token Density* (R), partially representative (P) and non-representative values (N) by U-Matrix region for the concerns under analysis.

Regions M , P , Q and O are also well defined clusters representing similar phenomena to cluster N , but with the lack of one of the other concerns. For instance, region O presents an area where, among the concerns in analysis in this experiment, only the concern *Data type verification* has a representative enough value for *Token Density*. Regions R , S_1 , S_2 and U have less marked patterns in the U-Matrix. For example, looking at the component plane area representing the region S_2 , the concern metric density for *Data type specialisation* is found in only part of the region (and even there with not so representative values). Region U seems to have a somehow complementary nature for the junction of regions S_1 and S_2 , but the only meaningful pattern found was a neuron representing the lack of other patterns under study. In another case, region R also presents a less well marked co-occurrence of representative values in the U-Matrix, but, in its CPs, has of the *Parallelisation* and *Memory allocation/ deallocation* with high values for *Token Density* in the whole region. Nevertheless, R occurs jointly (though, just partially) with other concerns.

7 | DISCUSSION

The usefulness of SOM-based analysis becomes obvious when more than two concerns are considered. In such cases, the SOM model replaces the simplistic use of sorting by *Token Density*. In the first experiment described in Section 6.2, the UbiSOM model presents large distances between many units, which jointly with the presence of some quantisation error, suggests a good enough convergence of the model. Nevertheless it does not yet represent some patterns in the data, i.e., the boundaries separating clusters in Figure 5 could be “crisper”. Even so, the observation that most CPs have clear continuous regions in several trained UbiSOMs, points us to a clear and concise division of the concern patterns under analysis. In the second experiment, after the use of multiSOM (Section 6.1), for the selection of the most relevant features, a very representative analysis with a very good convergence was achieved (as described in the second experiment – Section 6.3). So, this focus on more representative features not only solved the slight instability observed in the first experiment, but also confirms its results and further relates it with the *Memory allocation/ deallocation* concern.

Overlapping areas in the CPs (Figures 6 and 8) 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 sets of functional responsibilities (i.e., concerns).

Use of UbiSOM is marred by high numbers 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 it in terms of the UbiSOM, be it in terms of not reducing the quantisation error.

The feature selection used in the second experiment (Section 6.3) solves most observed problems with the high number of zeros of *Token Density* for the global dataset. Nevertheless, such high number of zeros of *Token Density* seems to be an instance of an exponential law on the token occurrence. A promising alternative is to use a method such as Word2Vec (Mikolov et al., 2013), a natural language method that can build high-quality distributed vector representations that capture precise syntactic and semantic relationships. However, for a more rigorous and replicable evaluation in these experiments we choose to keep just the information provided by an expert (cf. Table 1).

The $N = 6\,000$ parameter used in Equation 1 is an efficient way to give proper relevance to low *Token Density* values. M-files with one token occurrence per 60 LoC have a normalised value of 0.5, while one token occurrence per 200 LoC (normalised value of 0.3). Even values corresponding to less than one token per 1,000 LoC (normalised value lower than 0.2) are still detectable. Preliminary experiments done without the exponential normalisation (cf. Equation 1) showed a large uniform region and few co-occurring concerns. Indeed, UbiSOM training (and SOM, in general) is known to be sensible to too many zeros. This problem is observed for concerns with a high number of zeros such as, e.g., *Data type*


```

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('matlab_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;

FIGURE 9 m-file with high *Token Density* values relative to three of the analyzed concerns.

specialisation, 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 6) 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 UbiSOM analysis is the high correlation of *Token Density* values for concerns *Verification of function arguments* and *Data type verification* (see Figures 6 and 8). An illustrating example of an m-file with these concerns is shown in Figure 9. The high correlation between both these parameters is common in many analyzed 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 (Cardoso et al., 2006).

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.

8 | RELATED WORK

The present work is based on *concern metrics* and therefore is related to past research work on that category of metrics (Figueiredo et al., 2008, 2012). These range from uses specifically meant as a preamble for subsequent refactoring to an AOP language (Ceccato and Tonella, 2004; Lopez-Herrejon and Apel, 2007), metrics with broader applicability but whose computation is manual rather than automated (Eaddy et al., 2007; Sant'Anna et al., 2007) and approaches that are both automated and broadly applicable (Ducasse et al., 2006; Baldi et al., 2008). The work presented here develops the idea of using data derived from token-based metrics, as initially proposed by (Monteiro et al., 2010). In the intervening years, a separate development of this work was presented Lessa et al. (2015b,a), which proposes a software visualisation tool whose visual metaphors are also based on token metrics. The metrics covered are counts of tokens of various kinds, as proposed by (Monteiro et al., 2010).

A work with several similarities to ours, is that by *Maisikely and Mitropoulos* (Maisikeli and Mitropoulos, 2010), 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 – an aspect-oriented extension of Java, is good at modularising. 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 a previous work (Monteiro et al., 2010).

Future Work

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

- **Automatise the development of the concern-token mapping.** We can use a distributional representation of tokens (Mikolov et al., 2013) 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: (i) automatise or semi-automatise the discovery of new, distinct patterns of occurrence that lead to new concerns and new entries for Table 1, and; (ii) automatically or semi-automatically refine existing entries of this 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 UbiSOM 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.
- **Derive further concern metrics.** The metrics used in the present work only scratch the surface of the metric space that can be used to feed the UbiSOM analysis. Even without going beyond the token level, further concern metrics can be derived, which will enable more precise identification of concerns, as well as broaden the scope of concerns that can be (semi-) automatically discovered. The results achieved with the proposed method are promising. The interactive and exploratory nature of multiSOM 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.

9 | CONCLUSION

This article makes the following contributions:

1. Proposes the *Token Density* metric as a direct development of some ideas proposed by Monteiro et al. (Monteiro et al., 2010).
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 arbitrarily large repositories of Matlab files.
3. Presents the first representative case-study of a multi-modal interactive interface for the UbiSOM – the multiSOM. The improvements reported in the experiment of Section 6.3 where possible due to the usage of that interface. This paper presents a first, hands-on use case of the multiSOM user interface.
4. The analysis described in this paper serves as a first validation of the *Token Density* metric.

This article shows how the UbiSOM's output can provide viewers with a panorama of the concern's co-occurrence patterns in the Matlab file input repository. It provides a first reported use of several tools relevant for an expert systems method using the UbiSOM in a sound and systematic way for analyses of Matlab code repositories. The reported analyses provide the first step for a tool enabling a systematic approach for analysing code repositories.

This work also provides the basis for a framework that is planned to integrate and support evolution tasks (development and re-engineering) of Matrix-oriented code bases. Since Matlab is a well established member of the family of Matrix oriented programming languages, more recent languages such as R or the *numPy* Python library are likely to directly benefit from this research.

ACKNOWLEDGEMENTS

We thank João Bispo for making available the repository of Matlab code used in this work.

REFERENCES

- Aslam, T., Doherty, J., Dubrau, A. and Hendren, L. (2010) Aspectmatlab: An aspect-oriented scientific programming language. In *Proceedings of the 9th International Conference on Aspect-Oriented Software Development*, 181–192. ACM.
- Baldi, P. F., Lopes, C. V., Linstead, E. J. and Bajracharya, S. K. (2008) A theory of aspects as latent topics. In *ACM Sigplan Notices*, vol. 43, 543–562. ACM.
- Bispo, J. and Cardoso, J. M. P. (2017) A matlab subset to c compiler targeting embedded systems. *Software: Practice and Experience*, 47, 249–272. URL: <http://dx.doi.org/10.1002/spe.2408>. SPE-15-0162.R2.
- Cardoso, J. M., Fernandes, J. M. and Monteiro, M. P. (2006) Adding aspect-oriented features to matlab. In *Fifth International Conference on Aspect-Oriented Software Development (AOSD 2016)*.
- Cardoso, J. M., Fernandes, J. M., Monteiro, M. P., Carvalho, T. and Nobre, R. (2013) Enriching matlab with aspect-oriented features for developing embedded systems. *Journal of Systems Architecture*, 59, 412–428.
- Ceccato, M. and Tonella, P. (2004) Measuring the effects of software aspectization. In *1st Workshop on Aspect Reverse Engineering*, vol. 12. Citeseer.

- Chidamber, S. R. and Kemerer, C. F. (1994) A metrics suite for object oriented design. *IEEE Transactions on software engineering*, **20**, 476–493.
- Ducasse, S., Girba, T. and Kuhn, A. (2006) Distribution map. In *Software Maintenance, 2006. ICSM'06. 22nd IEEE International Conference on*, 203–212. IEEE.
- Eaddy, M., Aho, A. and Murphy, G. C. (2007) Identifying, assigning, and quantifying crosscutting concerns. In *Proceedings of the First International Workshop on Assessment of Contemporary Modularization Techniques*, 2. IEEE Computer Society.
- Figueiredo, E., Sant'Anna, C., Garcia, A., Bartolomei, T. T., Cazzola, W. and Marchetto, A. (2008) On the maintainability of aspect-oriented software: A concern-oriented measurement framework. In *Software Maintenance and Reengineering, 2008. CSMR 2008. 12th European Conference on*, 183–192. IEEE.
- Figueiredo, E., Sant'Anna, C., Garcia, A. and Lucena, C. (2012) Applying and evaluating concern-sensitive design heuristics. *Journal of Systems and Software*, **85**, 227–243.
- Kellens, A., Mens, K. and Tonella, P. (2007) A survey of automated code-level aspect mining techniques. In *Transactions on aspect-oriented software development IV*, 143–162. Springer.
- Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M. and Irwin, J. (1997) Aspect-oriented programming. *ECOOP'97—Object-oriented programming*, 220–242.
- Kohonen, T. (1982) Self-organized formation of topologically correct feature maps. *Biological cybernetics*, **43**, 59–69.
- (2001) *Self-Organizing Maps*. Springer-Verlag Berlin.
- Lessa, I. d. M., Carneiro, G. d. F., Monteiro, M. J. T. and e Abreu, F. B. (2015a) A multiple view interactive environment to support matlab and gnu/octave program comprehension. In *Information Technology-New Generations (ITNG), 2015 12th International Conference on*, 552–557. IEEE.
- Lessa, I. M., de Figueiredo Carneiro, G., Monteiro, M. P. and e Abreu, F. B. (2015b) Scaffolding matlab and octave software comprehension through visualization. In *SEKE*, 290–293.
- Lopez-Herrejon, R. E. and Apel, S. (2007) Measuring and characterizing crosscutting in aspect-based programs: Basic metrics and case studies. In *International Conference on Fundamental Approaches to Software Engineering*, 423–437. Springer.
- Maisikeli, S. G. and Mitropoulos, F. J. (2010) 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, V1–212. IEEE.
- Marin, M., Van Deursen, A. and Moonen, L. (2004) Identifying aspects using fan-in analysis. In *Reverse Engineering, 2004. Proceedings. 11th Working Conference on*, 132–141. IEEE.
- Marques, N. C., Silva, B. and Santos, H. (2016) An interactive interface for multi-dimensional data stream analysis. In *Information Visualisation (IV), 2016 20th International Conference*, 223–229. IEEE.
- Matos, D., Marques, N. C. and Cardoso, M. G. (2015) Stock market series analysis using self-organizing maps. *Revista de Ciências da Computação*, **9**.
- Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S. and Dean, J. (2013) Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, 3111–3119.
- Monteiro, M., Cardoso, J. and Posea, S. (2010) Identification and characterization of crosscutting concerns in matlab systems. In *Conference on Compilers, Programming Languages, Related Technologies and Applications (CoRTA 2010), Braga, Portugal*, 9–10. Citeseer.

- Sant'Anna, C., Figueiredo, E., Garcia, A. and Lucena, C. J. (2007) On the modularity of software architectures: A concern-driven measurement framework. In *European Conference on Software Architecture*, 207–224. Springer.
- Silva, B. (2016) *Exploratory Cluster Analysis from Ubiquitous Data Streams using Self-Organizing Maps*. Ph.D. thesis, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa. Manuscript available at: <http://hdl.handle.net/10362/19974>.
- Silva, B. and Marques, N. C. (2015) The ubiquitous self-organizing map for non-stationary data streams. *Journal of Big Data*, 2, 1–22.
- Tarr, P., Ossher, H., Harrison, W. and Sutton Jr, S. M. (1999) N degrees of separation: multi-dimensional separation of concerns. In *Proceedings of the 21st international conference on Software engineering*, 107–119. ACM.
- Utsch, A. (1995) Self organizing neural networks perform different from statistical k-means clustering. In *In Proceedings of GfKI '95*.
- Utsch, A. and Herrmann (2005) The architecture of emergent self-organizing maps to reduce projection errors. In *Proceedings of the European Symposium on Artificial Neural Networks (ESANN 2005)*, 1–6. Verleysen M. (Eds).



NUNO CAVALHEIRO MARQUES was born in Lisboa, Portugal and is tenured Assistant Professor at Computer Science Department of Faculdade de Ciências e Tecnologia from Universidade NOVA de Lisboa. He is working as a researcher since 1991 and as a lecturer since 1997. His PhD work has pioneered Text Mining for the Portuguese language. His current research interests include artificial neural networks and deep learning used for knowledge extraction from text and data.



MIGUEL PESSOA MONTEIRO was born in Braga, Portugal and is tenured Assistant Professor at Computer Science Department of Faculdade de Ciências e Tecnologia from Universidade NOVA de Lisboa. He received in B.E. degree in computer science engineering in 1989, his Master's degree in 1996 and his Ph.D. in 2005 - all from Minho University. His research interests include software modularity, reengineering and evolution.



BRUNO SILVA Bruno Silva was born in the Azores, Portugal, in 1981. He received the B.E. degree in computer science engineering from Escola Superior de Tecnologia de Setúbal do Instituto Politécnico de Setúbal (ESTSetúbal), Setúbal, Portugal, in 2005, and the M.Eng. and Ph.D. degrees in computer science from the Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, Monte da Caparica, Portugal, in 2008 and 2017, respectively. Since February 2006, he has been with the Department Systems and Informatics, ESTSetúbal, where he is an Associate Professor since 2018. His current research interests include data mining, machine learning and neural networks.