An incremental approach to Semantic Clustering designed for software visualization

Juraj Vincúr
Slovak University of Technology
Faculty of Informatics and Information Technologies
Bratislava, Slovakia
juraj.vincur@fiit.stuba.sk

Ivan Polášek
Slovak University of Technology
Faculty of Informatics and Information Technologies
Bratislava, Slovakia
polasek@fiit.stuba.sk

***Abstract*— In this paper, we introduce an incremental approach to semantic clustering, designed for software visualization, inspired by behavior of fire ant colony. Our technique focus on identification of equally sized but natural clusters that provides better hindsight of software system structure for development participants. We also address performance issues of existing approaches by maintaining similarities based on global weights incrementally, using subspaces and covariance matrix. Effectivity of visualization is improved by representing multiple documents with precise medoid approximation.**

***Keywords*— Clustering; Ant colony; Visualization**

I. Introduction and Related Work

Understanding the structure of large and complex software systems is difficult task that demands huge effort. A certain level of understanding by software engineers is necessary when they perform daily development and maintenance tasks. Many approaches have been proposed to minimize the cost of such a comprehension, but most of them ignore huge amount of developer knowledge hidden in identifier names since they only focus on program structure [1] or documentation [2].

To enrich software analysis by this valuable source of information, Semantic Clustering has been proposed [3]. According to its evaluation this technique provides a useful first impression of an unfamiliar software system by grouping source code artifacts (e.g. class types, functions, methods) to clusters. Main problem of this approach identified by authors themselves lies in tendency of identifying one oversized and dozens of small clusters. Such a distribution is not suitable for software visualization and it is caused by use of average-linkage clustering algorithm. The authors also do not take in mind frequent software system changes (e.g. by adding features, bug fixing or refactoring) that each require recalculation of whole vector representation. This behavior leads to huge performance over-head as identified in [4, 20]. Use of incremental techniques should be considered to minimize time required for obtaining up-to-date representation since decisions based on non-up-to-date visualization of software system may be wrong and lead to degradation of its structure. Different approaches [5, 2] try to increase quality of identified topics by use of probabilistic model called Latent Dirchlet Allocation [6]. However, this method is very sensitive to user pre-defined number of resulted topics which are also harder to visualize.

It is obvious that robustness and quality of visualization based on Semantic Clusters is highly related to choice of clustering algorithm. An exhausting review of clustering algorithms may be found in [7]. We will focus on 3 types of cluster analysis techniques since our method is partially related to them.

First set of methods are members of hierarchical cluster analysis which aims to build hierarchy of clusters by using generally two strategies. Agglomerative, also known as bottom-up approach, which in initial phase considers each observation as cluster. Pairs of clusters are iteratively merged until only one huge cluster remains or the other defined criterion is met. Divisive, top-down approach, treats data in opposite direction. An efficient and scalable representative of agglomerative methods is BIRCH (Balanced Iterative Reducing and Clustering) [9] which requires two input parameter: the distance threshold and the tree branching factor. With respect to these parameters the BIRCH algorithm build CF-tree consisted of summary information (clustering features) about candidate clusters. Then, instead of using full data set, another agglomerative clustering algorithm is applied on these features to refine clusters. This approach is suitable for very large data sets, however, since each node in CF-tree can contain only limited number of data items obtained clusters may not correspond to natural clusters. BIRCH also requires significant effort to tune its parameters [8]. Non spherical clusters trouble BIRCH as well since it uses diameter or radius to control boundaries of clusters.

Second set of algorithms are based on minimum or maximum spanning tree. Multiple non-incremental approaches are described in [9]. This type of methods may be considered as subset of hierarchical clustering. Analogy between them is well described in [10], where MST implementation of single-linkage clustering is presented. We are not aware of any incremental approach based on spanning tree even though their base implementations are relatively poor in performance with respect to BIRCH.

Third group consists of ant colony algorithms. These algorithms, members of swarm intelligence family, are inspired by multiple species and behavior models of ants. A brief survey may be found in [11].

With respect to identified limitations, this paper aims at providing incremental approach to semantic clustering that identifies equally sized but natural clusters. Section 2 explains concepts of our approach. Section 3 presents our adaption of circle packing layout to visualize obtained semantic clusters. Results and evaluation are presented and discussed in section 4. Section 5 summarizes this paper and its contribution.

II. The Fire Ant approach to clustering

Our clustering approach is inspired by behavior of fire ant colony during floods. Fire ants have unique ability to gather together and form rafts on the surface of water [13]. This formation in which they can retain for weeks, allows them to survive. We assume that each ant has limited vision range in which it can sense the other ants. We also assume that during gathering ants are dispersed in space so first clusters of ants are formed and then these clusters are merged to one big raft (hierarchy). Moving principles of these ants are often compared to fluids. The behavior we described is similar to oil spreading on the surface of water. In this paper we work with simplified model that considers only one possible way of grabbing another ant (instead of 7), by its jaw. Number of survivors then depends on number of connected ants (raft size). To maximize it, we add restriction to suppress cycle connections between ants, thus we get spanning tree representation, which is close to minimum.

In this section, we will first present our domain specific observations that were considered in design and implementation. Next, we will discuss main steps of our approach in following order: preprocessing (building corpus representation), spanning tree construction, identification of cluster-points, assigning documents to cluster-points and merging cluster-points to clusters.

A. Domain observations

Most approaches treat source code snippets as textual documents although they are produced differently. Source code projects are usually implemented in incremental manner by applying batches of changes. Dimensionality of space representing documents affected by single batch should be relatively low since these documents are usually related (e.g. classes affected by single commit). To support our hypothesis we conduct an experiment in which we simulate development of few open source projects handled by Git version control system. Description of chosen projects may be found in TABLE II. Results of experiments are shown in Fig. 1. Data are summarized in TABLE I. As we can see, average values of dimensionality of such a subspace lie between 172.07 and 297.14, which is relatively small compared to total number of dimensions (see TABLE II.). Moreover, it doesn't seem to be strongly related to project size. Trend lines in Fig. 1 even suggest that this dimensionality may be constant (in average). Another aspect we were interested in is the density of document-term matrix (DTM) which is comparable to domain of textual documents (see TABLE II.).

TABLE I. Average value[a] in covariance matrix of increment

	Titan	Elastic	Neo4j	Spring	Total
avg sim	0.35	0.48	0.34	0.39	0.39
med dim	146	132	200	196	199
avg dim	201.92	172.07	270.07	297.14	235.30

a. Diagonal values set to 0.

TABLE II. Properties of DTM of chosen projects

	Titan	Elastic	Neo4j	Spring
row	1238	5593	5753	9833
col	3226	5297	5257	6562
nnz	63009	249107	225114	414638
density	1.58%	0.84%	0.74%	0.64%
avg	50.90	44.54	39.13	42.17

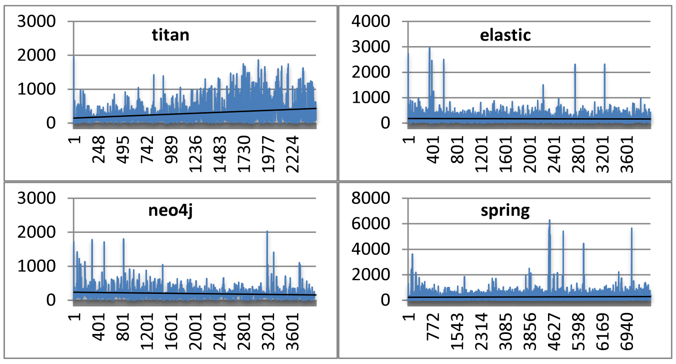


Fig. 1. Number of dimensions per commit subspace (with trend lines).

B. Corpus representation

Quality of clusters is highly dependent on precision of similarity measure obtained in preprocessing step. In our work we use cosine similarity [16] and corpus representation that exploits global weights and low density of document-term matrix. Primary corpus representation is constructed as document-term matrix with term frequency (TF) weighting. No normalization vector or global scale is applied because these may change in time by editing corpus. However, vector of document lengths and average document length are updated for further usage. Document lengths are used for calculation of IDF [14] and average document length for pivot normalization [18]. TF weights are used for serialization purposes only. In order to calculate document similarity, TF-IDF model has to be computed by applying IDF vector to normalized primary matrix. We consider global weight in TF-IDF essential due to its tendency to suppress false cluster identification caused by crosscutting concerns [15]. For example, using raw TF weights, term "map" widely spread across classes in source code of distributed database may cause identification of oversized cluster that will contain all these classes. Problem with TF-IDF model in terms of incremental approaches is that weights change by adding, removing or editing documents. To overcome this behavior or even employ it we compute covariance matrix. We assume that documents similar at some particular not too early point of project development should be relatively equally similar at later point of development as well. Moreover, we assume that earlier point similarity is more precise because it takes in account aspect of time. If two documents were highly similar before some words become widely spread (by adding some module or by merging branches) they are probably similar even if new TF-IDF weights say otherwise. To add new documents to covariance matrix, their similarities to all documents have to be calculated (see Fig. 2). However, since the subspace for these documents tends to be constant in number of dimensions (term dimension of DTM), only one dimension of matrix is growing (document dimension of DTM). Moreover, document dimension may be filtered as well because there is high probability that relatively low dimensional vector (rounded average of 44 dimensions for our sample) extracted from high-dimensional space (rounded average 5085) representing single class has no common dimensions with constructed subspace (rounded average 235).

Since both matrices, TF weights and TF-IDF model are sparse, we address memory efficiency problem by storing them using sparse matrix schema. More specifically, TF weights are stored using dictionary of keys format, which assigns to each row, column tuple corresponding value. This structure is efficient for constructing sparse matrices incrementally, for row and column slicing, but not efficient for arithmetic operations. On the other hand, TF-IDF model is represented by compressed sparse row matrix format. This format consists of indices (array of column indices), data (array of nonzero values) and pointer array whose items points to row starts in indices and data. It is an efficient representation for arithmetic operations and dot product, but very slow representation for column slicing or changing structure.

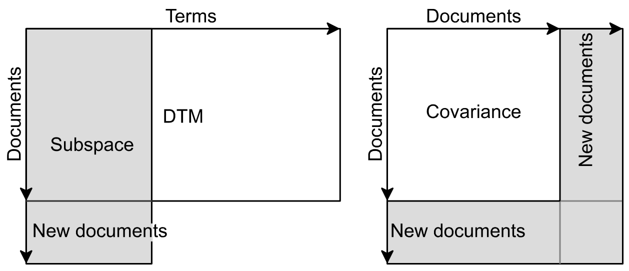


Fig. 2. Adding increment to covariance matrix.

```
INIT graph
FOR I = 1 to K
  FOR each document
    SET doci to its I-th most similar document
    IF edge between nodes representing document and
       doci will not produce cycle in graph THEN
       add edge between these nodes
    ENDIF
  END FOR
END FOR
```

Fig. 3. Pseudocode of spanning tree construction algorithm.

C. *Spanning tree construction*

Simplified model based on spanning tree structure has been selected due to several benefits. It is easy to maintain, analyze and visualize. Moreover, spanning tree clustering methods are capable of identifying clusters with irregular boundaries (e.g. not spherical) [12]. In our algorithm, inspired by swarm intelligence, we represent each document by single agent. Each agent has limited vision of *k* nearest / most similar nodes based on cosine similarity, obtained from covariance matrix (storing full covariance matrix isn't necessary). Since ants are capable of making max. 7 connections we set k to this value. The steps required to compute spanning tree are shown in Fig. 3. This approach has been chosen over other algorithms with concurrency [19] due to the time complexity, its natural incrementalism and fact that using nearly minimum spanning tree has no significant impact on quality of resulted clusters. Note that representing each document as single agent brings certain level of rivalry that leads to tendency of equally sized clusters, but not for price of having non-natural clusters.

D. *Cluster-points*

In created spanning tree, we select nodes that have more than 2 links. We call them cluster-points and they are good approximation of medoid for group consisted of selected node and nodes adjacent to him. Note that this group represents group of mutually similar documents since it was obtained from spanning tree close to minimum. TABLE III. presents measured number of incorrectly assigned medoids, total number of identified cluster-points, total error (sum of differences between total dissimilarity of real medoid and approximated medoid) and average error. As we can see, only few medoids are approximated incorrectly and even these bad approximations yield low average error. Nodes with degree 2 will be referred as connectors.

TABLE III. AVERAGE AND TOTAL ERROR OF MEDOID APPROXIMATION

	JHotDraw	**Titan**	**Elastic**	**Neo4j**	**Spring**
missed	4	8	73	68	156
total	86	310	1336	1396	2349
error	0.07	0.22	3.96	3.34	4.58
avg error	0.02	0.03	0.05	0.05	0.03

E. *Forming clusters*

Selecting cluster-points in spanning tree revealed high probability of forming groups of adjacent cluster-points. These groups refer to cluster candidates. Different strategies may be applied to obtain real clusters. The one we present could be described by pseudocode in Fig. 5. Each one of resulted clusters is represented by list of cluster-points where each cluster-point has list of assigned nodes. Example of spanning tree and obtained hierarchy tree structure may be found in Fig. 4.

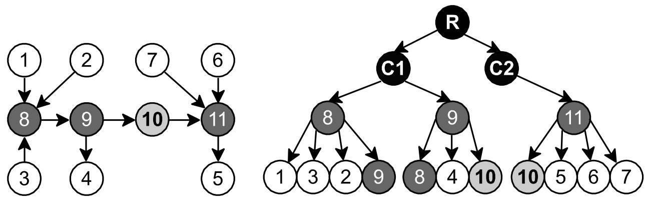


Fig. 4. Spanning tree (left) and corresponding hierarchy (right). Black nodes - root and identified clusters; dark gray – cluster-points; gray – connectors.

```
SET subg to subgraph of spanning tree that contains
    only cluster-points
FOR each component in connected components of subg
  IF number of nodes in component is smaller than
     defined level of abstraction THEN
    exclude nodes of component from cluster-points
  ENDIF
END FOR
SET level to 0
WHILE not all nodes of spanning tree are traversed
  SET level = level + 1
  FOR each cluster-point
    SET traversed to nodes traversed in spanning tree
        by BFS at current level initialized from
        current cluster-point
    FOR each node in traversed
      IF node is not cluster-point and not already
         assigned THEN
        assign node to current cluster-point
      ELSE IF level == 1 THEN
        assign copy of node to current cluster-point
        to prevent medoid representation
      END IF
    END FOR
  END FOR
END WHILE
SET subg to subgraph of spanning tree that contains
    only cluster-points
SET clusters to connected components of subg
```

Fig. 5. Pseudocode for cluster-forming strategy.

F. *Updating structure*

Two main strategies have to be considered in terms of maintaining spanning tree and cluster hierarchy structure. In first, building of vector space model and covariance matrix is managed incrementally and construction of spanning tree and cluster hierarchy is recalculated each time since it requires only single pass over documents. This strategy will probably yield more accurate results, but it causes certain level of overhead.

Second strategy maintains spanning tree and cluster hierarchy incrementally as well by reducing recalculations to affected subtrees only. Formed clusters will be more stable, but less accurate.

III. Data visualization

Most companies develop large software projects using version control system. Source code is physically located on one or multiple servers what allows us to create web based visualization tools. Main benefit of this approach is the possibility to see the results everywhere, immediately without any installation required. There are many visualization JavaScript libraries suitable for this purpose. Most of them require only JSON dump in specified format. Our software visualization, inspired by [23], use d3.js library which provides robust documentation and large database of examples. From these examples we adapt circle packing layout (see Fig. 6).

IV. Results and evaluation

Discussing our results we will mainly focus on quality of clusters from view of software engineer trying to understand the structure of existing software system. Use of external validity techniques would be the most suitable, but since we are not aware of any evaluated data set from source code domain relevance judgments [17] are applied.

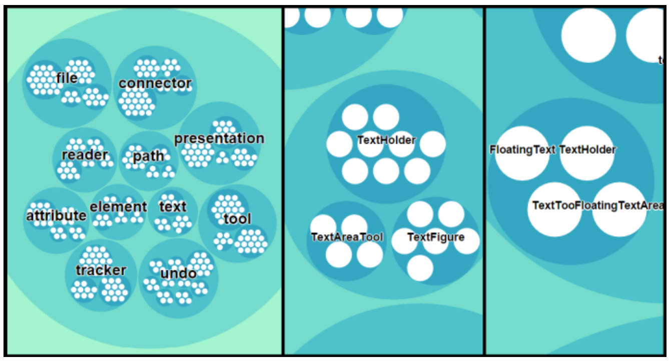


Fig. 6. Circle packing layout (visualization of JHotDraw, zoomed views after clicking "text" cluster and "TextAreaTool" cluster-point).

TABLE IV. Example of identified cluster-points in JHotDraw

Cluster-point	Classes that cluster-point represents
AlignAction	East, West, South, North, Vertical, Horizontal
Handle	NullHandle, CloseHandle, HandleMulticaster, HandleListener
OpenAction	NewAction, OpenRecentAction, Project, …
DuplicateAction	CopyAction, PasteAction, CutAction, DeleteAction

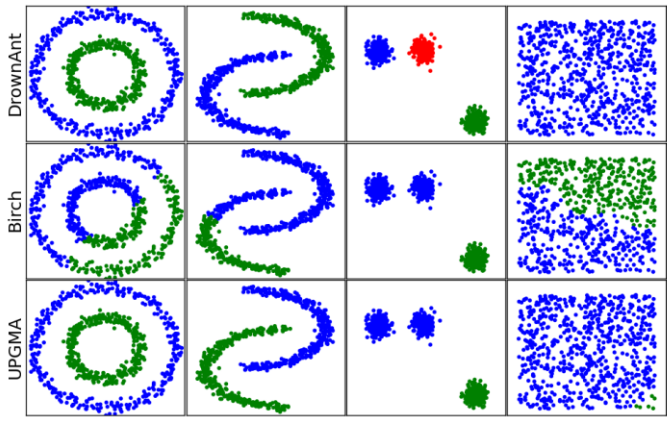


Fig. 7. Comparission of our approach (DrownAnt) with selected algorithms in 2D space.

Relatively small project as JHotDraw yield with abstraction level 1 11 clusters. As we can see in Fig. 6 (left side of picture) these clusters are relatively equally sized. We judge the using of cluster-points as representative of group of documents very positively (see TABLE IV.). A very good example is class AlignAction that represents classes North, East, West, South, Vertical and Horizontal. Similar behavior is observed in all sample projects. To proof our spanning tree based concept in terms of natural clusters, we compare our approach with selected algorithms (Birch and average linkage clustering, or UPGMA) in two-dimensional space with well separated clusters. Without use of any cluster-forming strategy or tune of parameters we were able to correctly identify all natural clusters (see Fig. 7). Such an experiment is not considered as sufficient evaluation since there’s no proof of relation between two and multidimensional space but it serves as an appropriate example to present potential of proposed technique. To briefly evaluate our approach in multidimensional space we create a table of clusters with corresponding cluster-points that were identified in JHotDraw project (see TABLE V.). With our basic knowledge about JHotDraw architecture, we were able to identify four core parts of system (see TABLE VI.). By comparing these components to obtained clusters we observed that part “Drawing, figure” corresponds to cluster “Undo, redo, decorator”, parts “Handle” and “Tool” to “Tracker, mouse, evt” and “Command” to “File, project, open”. In addition we identified two clusters related to XML since clipboard operations and external storage in JHotDraw use this format. First, cluster “Reader, xml, entity”, is related to XML parsing while second, “Element, attribute, open”, focus on XML DOM manipulation. Cluster “Path, node, bezier” is related to fact that support for lines and complex shapes in JHotDraw is based on bezier paths (instead of polygons). “Text, origin, layout” corresponds to classes handling text areas, “Presentation, figure, graphical” to layout and composition of figures, “Attribute, key, forbid” to setting attributes of figures and “Tool, bar, toolbar” to toolbars and panels of drawing view.

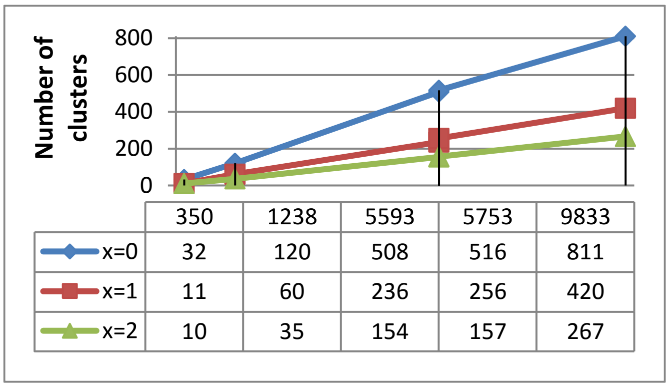
TABLE V. IDENTIFIET CLUSTERS AND CLUSTER-POINTS IN JHOTDRAW

Cluster	Cluster-points of cluster
Undo, redo, decorator	Figure, FigureEvent, AbstractFigure, AbstractFigureListener, UndoRedoManager, AttributeChangeEdit, SetBoundsEdit, RestoreDataEdit, RunnableWorker
Tracker, mouse, evt	SelectionTool, FormListener, HandleTracker
File, project, open	LoadRecentAction, NewAction, OpenAction, OpenRecentAction
Reader, xml, entity	XMLUtil, StdXMLParser, IXMLParser, IXMLValidator, NonValidator
Element, attribute, open	JavaxDOMOutput, DOMOutput, NanoXMLLiteDOMOutput, NanoXMLDOMInput, DOMInput
Connector, target, connection	ChopBoxConnector, Connector, ChopEllipseConnector, BidirectionalConnectionTool, ChangeConnectionHandle
Path, node, bezier	BezierTool, BezierPath, BezierFigure, BezierControlPointHandle
Text, origin, layout	TextFigure, TextAreaTool, TextHolder
Presentation, figure, graphical	HorizontalLayouter, AbstractLayouter, ListFigure, GraphicalCompositeFigure
Attribute, key, forbid	ColorChooserAction, DefaultAttributeAction, AttributeAction, AbstractAttributedCompositeFigure
Tool, bar, toolbar	ToggleToolBarAction, DrawApplicationModel, ToolBarPrefsHandler

During experiments we did not expect frequent occurrence of the term "Map" in project Titan (distributed database) across clusters. After short analysis we realized that term "map" occurs in 345 classes from total number of 1238 and so it is related to cross-cutting topic. IDF works correctly in this example. During the browsing of documents of one cluster, we realize that sometimes their vocabulary is not similar. This is caused by use of vector enrichment as proposed in [4]. Such an observation may be confusing for user as well, thus some abstraction of this relation should be added to visualization.

Size of the clusters is considered rational with respect to total number of documents. According to Fig. 8 the number of clusters may scale linearly or even logarithmic, but to support these assumptions statistically we need more significant number of samples to analyze.

In Fig. 9 the results of performance evaluation of computing covariance matrix are presented. Incremental approach is compared with calculation of full covariance matrix each time, to see how both of them scale by number of applied commits. As we can see, time required by base approach grows significantly faster than the time required by incremental. Moreover, incremental approach will scale even better after applying document filtering. Since our current implementation of adding increments to covariance matrix scales linearly, there is yet no point of managing spanning tree and cluster hierarchy incrementally (we can recalculate them by single pass over documents).



	350	1238	5593	5753	9833
x=0	32	120	508	516	811
x=1	11	60	236	256	420
x=2	10	35	154	157	267

Fig. 8. Number of identified clusters.

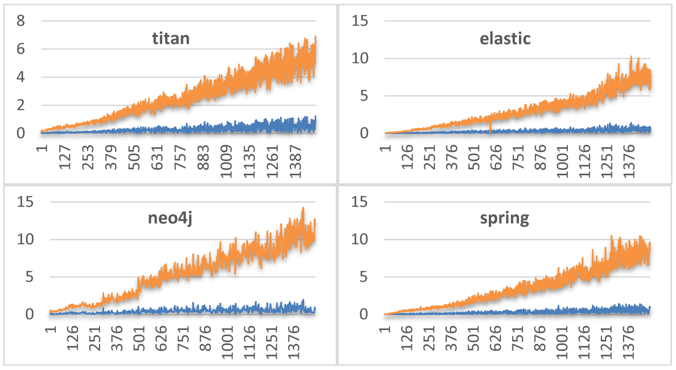


Fig. 9. Comparison of computational time between base (orange) and incremental (blue) approach of computing covariance matrix.

TABLE VI. IDENTIFIED CORE PARTS OF JHOTDRAW

Part	Responsibility
Drawing, Figure	Drawing represents two-dimensional space and consists of figures.
Tool	DrawingView inputs are delegated to its current Tool.
Handle	Used to change a figure by direct manipulation.
Command	Classes related to command design pattern that provides basic project actions (save, exit, open).

V. Conclusion and Future Work

In this paper we propose new incremental technique for semantic clustering, designed for software system visualization, inspired by behavior of fire ant colony, slightly sensible to optional tuning of single parameter, which is capable of identifying natural, but relatively equally sized clusters even with irregular boundaries. We employed low density of DTM to minimize performance overhead by calculating document to document similarities in low dimensional subspaces and by storing them to incrementally maintained covariance matrix. We reduced size of browsing space by applying three-layer hierarchy visualization with medoid based abstraction. To provide efficient preview of documents, we added to our visualization source code browsing. As we showed in evaluation, we achieve intuitive representation that provides good hindsight of software system structure and functionality.

In future work, our primary goal is to design and implement 3D visualization of our software system representation that utilize benefits of the third dimension as suggested in [21, 22]. Moreover, we would like to integrate resulted tool in the IDE and enrich it by mixed reality features and advanced interaction techniques.

VI. ACKNOWLEDGMENTS

This work was supported by the Scientific Grant Agency of Slovak Republic (VEGA) under the grant No. VG 1/1221/12. This contribution is also a partial result of the Research & Development Operational Programme for the project Research of Methods for Acquisition, Analysis and Personalized Conveying of Information and Knowledge, ITMS 26240220039, co-funded by the ERDF.

References

[1] Robert DeLine and Kael Rowan. 2010. Code canvas: zooming towards better development environments. In Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2 (ICSE '10), Vol. 2. ACM, New York, NY, USA, 207-210.

[2] Hazeline U. Asuncion, Arthur U. Asuncion, and Richard N. Taylor. 2010. Software traceability with topic modeling. In Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1 (ICSE '10), Vol. 1. ACM, New York, NY, USA, 95-104.

[3] Adrian Kuhn, Stéphane Ducasse, and Tudor Gírba. 2007. Semantic clustering: Identifying topics in source code. Inf. Softw. Technol. 49, 3 (March 2007), 230-243.

[4] Marek Uhlár and Ivan Polasek. 2012. Extracting, identifiyng and visualisation of the content in software projects. In Proceedings of the 4th World Congress on Nature and Biologically Inspired Computing (NaBIC '12), November 2012, 72-78, IEEE Press.

[5] Erik Linstead, Paul Rigor, Sushil Bajracharya, Cristina Lopes, and Pierre Baldi. 2007. Mining concepts from code with probabilistic topic models. In Proceedings of the 22nd IEEE/ACM international conference on Automated software engineering (ASE '07). ACM, New York, NY, USA, 461-464.

[6] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent dirichlet allocation. J. Mach. Learn. Res. 3 (March 2003), 993-1022.

[7] A. K. Jain, M. N. Murty, and P. J. Flynn. 1999. Data clustering: a review. ACM Comput. Surv. 31, 3 (September 1999), 264-323.

[8] Marcel R. Ackermann, Marcus Märtens, Christoph Raupach, Kamil Swierkot, Christiane Lammersen, and Christian Sohler. 2012. StreamKM++: A clustering algorithm for data streams. J. Exp. Algorithmics 17, Article 2.4 (May 2012), 1.2 pages.

[9] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. 1996. BIRCH: an efficient data clustering method for very large databases. SIGMOD Rec. 25, 2 (June 1996), 103-114. Oleksandr Grygorash, Yan Zhou, and Zach Jorgensen. 2006. Minimum Spanning Tree Based Clustering Algorithms. In Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI '06). IEEE Computer Society, Washington, DC, USA, 73-81.

[10] John C Gower and GJS Ross. Minimum spanning trees and single linkage cluster analysis. Applied statistics, pages 54–64, 1969.

[11] Jafar, O. a. M. and Sivakumar, R. Ant-based Clustering Algorithms: A Brief Survey. International Journal of Computer Theory and Engineering 2, 5 (2010), 787–796.

[12] C. T. Zahn. 1971. Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters. IEEE Trans. Comput. 20, 1 (January 1971), 68-86.

[13] Mlot NJ, Tovey CA, Hu DL. Fire ants self-assemble into waterproof rafts to survive floods. Proceedings of the National Academy of Sciences of the United States of America. 2011;108(19):7669-7673.

[14] Georgios Paltoglou and Mike Thelwall. 2010. A study of information retrieval weighting schemes for sentiment analysis. In Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL '10). Association for Computational Linguistics, Stroudsburg, PA, USA, 1386-1395.

[15] Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J-M. Loingtier and J. Irwin, Aspect-Oriented Programming, ECOOP'97 Conference proceedings, LNCS 1241, June 1997, pp. 220 – 242.

[16] Gerard Salton and Michael J. McGill. 1986. Introduction to Modern Information Retrieval. McGraw-Hill, Inc., New York, NY, USA.

[17] Ellen M. Voorhees. 1998. Variations in relevance judgments and the measurement of retrieval effectiveness. In Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR '98). ACM, New York, NY, USA, 315-323.

[18] Amit Singhal, Chris Buckley, and Mandar Mitra. 1996. Pivoted document length normalization. In Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR '96). ACM, New York, NY, USA, 21-29.

[19] Jaroslav Nešetřil, Eva Milková, Helena Nešetřilová, Otakar Borůvka on minimum spanning tree problem Translation of both the 1926 papers, comments, history. Discrete Math. 233, 1-3 (April 2001), 3-36.

[20] Ivan Polasek and Marek Uhlár. 2013. Extracting, Identifying and Visualisation of the Content, Users and Authors in Software Projects. LNCS Transactions on Computational Science XXI : Special Issue on Innovations in Nature-Inspired Computing and Applications. Springer, 269-295.

[21] Lukas Gregorovic and Ivan Polasek. 2015. Analysis and design of object-oriented software using multidimensional UML. In Proceedings of the 15th international conference on knowledge technologies and data-driven business (i-KNOW '15). ACM, New York, NY, USA.

[22] Lukáš Gregorovič, Ivan Polasek and Branislav Sobota. 2015. Software model creation with multidimensional UML. CONFENIS 2015, Held as Part of WCC 2015, Daejeon, Korea, October 4-7, 2015. Springer LNCS 9357, 2015, pp. 234-352.

[23] Ivan Polasek et al. 2013. Information and Knowledge Retrieval within Software Projects and their Graphical Representation for Collaborative Programming. In Acta Polytechnica Hungarica. Vol. 10, No. 2. 173 - 192.