Extracting, Identifiyng and Visualisation of the Content, Users and Authors in Software Projects

Ivan Polášek and Marek Uhlár

Faculty of Informatics and Information Technology
Slovak University of Technology in Bratislava
Slovakia
{polasek, uhlar}@fiit.stuba.sk

Abstract. The paper proposes a method for extracting, identifying and visualisation of topics, code tiers, users and authors in software projects. In addition to standard information retrieval techniques, we use *AST* for source code and *WordNet* ontology to enrich document vectors extracted from parsed code, *LSI* to reduce its dimensionality and the swarm intelligence in the bee behaviour inspired algorithms to cluster documents contained in it. We extract topics from the identified clusters and visualise them in 3D graphs. Developers within and outside the teams can receive and utilize visualized information from the code and apply them to their projects. This new level of aggregated 3D visualization improves refactoring, source code reusing, implementing new features and exchanging knowledge.

Keywords: Software Project, Visualisation, Source Code, WordNet Ontology, Topic Identification and Extraction, Latent Semantic Indexing, Bee Behaviour Inspired Algorithms, AST, Swarm Intelligence, Authorship.

1 Introduction

This paper describes our approach to information and knowledge mining, which aims to support collaborative programming and helps software developers in medium and large teams to understand complicated code structures and extensive content as well as to identify source code authors and concrete people working with existing modules. Accordingly newcomers as well as other colleagues can reference to real source code authors and users more efficiently. The goal of our research is to provide a new perspective on software projects for participants to find and reuse particular parts of the source code. Subsequently applied graphs demonstrate the structure of the software project and the main topics contained in the source code in a natural way for them. This can help newcomers to quickly adapt to the project, but also senior developers can benefit from these methods. Determination of the source code topics can be used for purposes of identifying domain expertise of developers. It is also possible to support program comprehension by identifying common topics in source code. The contribution of our method is to find new level of aggregated visualization for the identi-

fied and interconnected information from the source code for collaborative development.

2 Related Works

Our approach is based on the combination of swarm intelligence, *LSI* (*Latent Semantic Indexing*) and the standard techniques of information retrieval for parsing software project.

We were inspired by method proposed by Kuhn et al., (2007). The steps of mentioned method are creation of term-document matrix, using *LSI* to reduce its dimensionality and afterwards aggregating it with standard hierarchical average-linkage clustering algorithm. *LSI* is used as inverted index to extract topics (top relevant terms) from identified clusters [1].

The results are displayed with correlation matrix (Fig. 1). It has documents on sides. The similarity between two documents d_i and d_j is $a_{(i,j)}$.

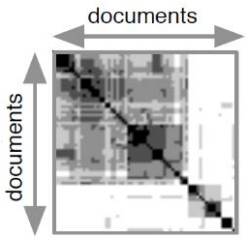

Fig. 1. Correlation Matrix [1].

In our work we try to evaluate the suitability of swarm intelligence bee inspired algorithms on text document clustering applied in many fields of research [2], [3], [4], [18]. We use *Flower Pollination Algorithm* (*FPA*) [5] and *Artificial Bee Colony Algorithm* (*ABC*) [6]. Main advantages of *FPA* in clustering are the inutility of input parameters such as number of clusters and initial cluster centers.

3 Our approach

Based on the mentioned method [1] we created our own approach. We added term-document matrix enrichment via identifying relations between documents and *WordNet*. Our idea is to improve the topic identification and replace clustering with the basic hierarchical average-linkage by clustering algorithms inspired by swarm intelligence bee behaviour which are intensively researched nowadays and were proven suitable for clustering [6]. Also, we proposed visualisation for identified clusters (topics) as a coloured structure in software project.

Fig. 2 is the model of our approach, where simply all steps except *parsing*, *clustering* and *visualising* are optional and their purpose is to improve the results or speed of

the process. The system is modular, so it is easy to replace either the clustering module or another one with a different component (using different algorithm) for further research.

First step of our method is parsing of the source code contained in software project. This can be done on three levels of granularity:

1. Source files - one document is equal to one source file. It can contain more classes.
2. Classes - one document is equal to one class or interface.
3. Methods - one document is equal to one method.

Currently, our prototype is able to work with projects created in .NET framework, C# language. We start by parsing the solution files (*.sln) which involves information about projects contained in solution. Afterwards, we parse individual project files (*.csproj) which contain information about individual files.

3.1 Parsing and indexing

We index all extracted documents in the open source information retrieval software library *Lucene*. We use these pre-processing steps to reduce noise in the indexed documents:

1. Splitting identifier names by camel case naming convention. (CamelCase -> Camel Case)
2. Splitting identifier names by under score naming convention. (Under_Score -> Under Score)
3. Words with less than three characters are removed. (is, &&, ||)
4. All words are converted to lower case.
5. Removal of stop words – words that do not contribute to semantic meaning. (English stop words – and etc., C# keywords – private, static, while, switch).

These procedures contribute to better topic identification in particular phases of our method. Afterwards, we identify relations between indexed documents. We will use them in the second step of enriching the *term-document matrix*. For this task we use open source library *NRefactory*. Through it we are able to extract *AST* (*Abstract Syntax Tree*), from which we can identify relations between extracted documents.

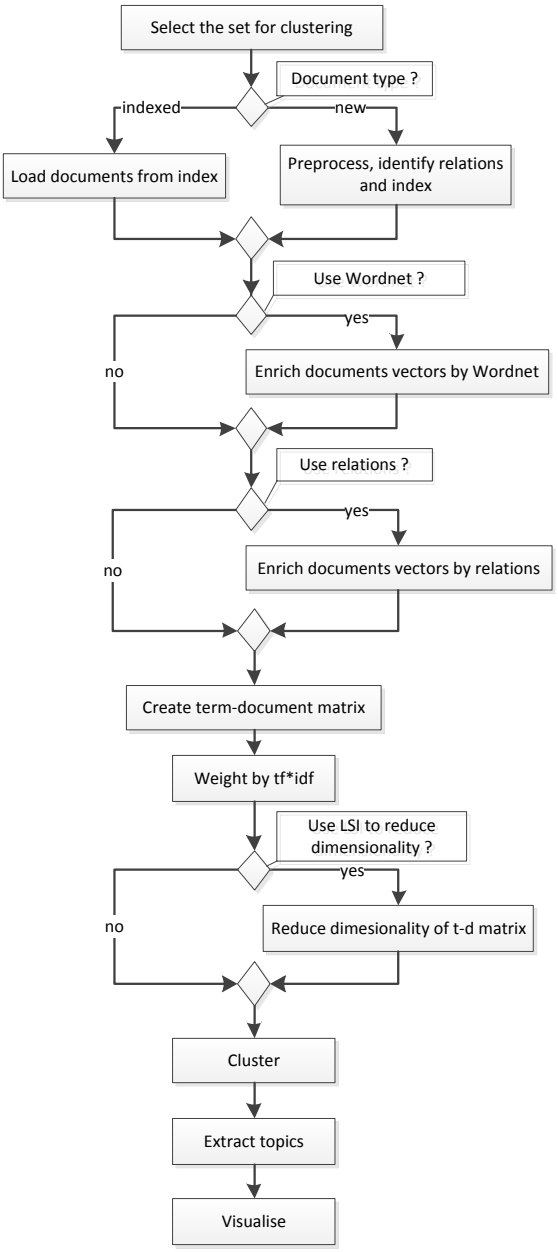

Fig. 2. Method steps.

After identification of relations we create term vector for each document (source file, class, method etc.). Finally, we create term-document matrix containing term frequencies in a particular document. In Fig. 3 we can see a simple example.

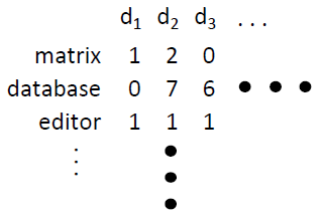

Fig. 3. Term-document matrix example.

We normalize the matrix with standard *tf*idf* algorithm with formula:

$$tfidf(d,t) = tf(d,t) * log_2\left(\frac{|D|}{1+df(t)}\right)$$

where ***d*** is a document from a collection we are working on, ***t*** is a concrete term from this collection and ***tf(d,t)*** is the frequency of the term *t* in the document *d*. ***|D|*** is the number of documents in our collection and ***df(t)*** is the number of documents in the collection which contains the term *t*.

3.2 Using Relations in Source Code to Enrich the Document Vectors

We use relations identified in the previous step to enrich the document term vectors. The type of identified relations depends on the selected granularity level we are working with (source file, class, method).

Source Files

On this level, we are enriching the term vector of each document (source file) in our collection with terms extracted from name of solution, project and eventually the directory it belongs to. Enrichment is done by adding the extracted terms to each document's term vector with particular weight.

We have estimated the weighting coefficient for these enrichments (Table 1) where ***L*** is *Label* and ***EV*** is *Estimated value*.

Table 1. Weighting coefficients.

L	Description	EV
W_S	Terms extracted from solution. Solution also contains project and files with different topics. Terms are usually very general. → Weight coefficient is very low.	0,1
W_P	Terms extracted from project names are usually very close to the topic contained in the project's source files → we set the coefficient higher	0,5
W_D	Directory names are very specific but they can also be general and from the topic contained in the files (e.g. Tools) → the coefficient is not as high as for the project.	0,2

The adding is done after *tf***idf* weighting and it can enhance the identification of the topic contained in the documents and extend the vocabulary.

Classes and Interfaces

On this level, we use relations identified through *NRefactory* in the previous step. We are able to identify *aggregation*, *generalization*, *nested classes* and *partial classes*.

For partial classes we offer the possibility to work with them as one class. There is a switch in the prototype to turn this behaviour off.

In case of generalization we add parent class terms into child class term vector with coefficient 0.5. But we are also adding term in reverse direction (child -> parent), but with lower coefficient (0.2). Terms in child class could broaden the topic in parent class.

Furthermore, for aggregation we add terms from referenced class into a referring one. Terms contained in the name of referenced class could be not explanatory enough. We recommend very low coefficient (0.1).

Methods

We work with two types of relations on this level of granularity:

- Override – hiding of method from parent class with our own implementation.
- Overload – methods with the same name but with different parameters.

We are not estimating the indexes. They will be specified by further testing or manually by domain expert.

3.3 Using WordNet Ontology to enrich the Term-Document Matrix

One of the problems in the semantic clustering of source code identified by [1] is the size of present vocabulary. It is considerably smaller than in the common text documents. As a consequence of this feature the topic extraction was not adequate in some cases.

The authors provide an example of false identification in the source file, in which the main topic was obviously 'text editor'. However the extracted topic was out of the context because the vocabulary simply does not contain general terms 'text' and 'editor'. The authors propose a solution of using a general English ontology (e.g. WordNet) to improve the results.

WordNet is the ontology (lexical database) containing English nouns, verbs, names, adjectives and adverbs organized in synsets. Synset is a group of semantically equivalent elements. The name used for synset in WordNet is concept. Concepts are interconnected by semantic relations (hypernyms, hyponyms, coordinated terms, holonyms, meronyms, etc.).

We use WordNet to enrich the term-document matrix. Our method is designed on the basis of research performed by [8]. Authors identify three steps to enrich the term-document matrix:

1. Mapping of terms to concepts.
2. Selecting a strategy how to join terms from matrix with identified concepts.
3. Selecting a strategy of choosing the concept.

Only noun terms to concepts are mapped in the current prototype. As the strategy of joining terms with concept we choose adding the concepts to terms. We want to extend the vocabulary as much as possible because of the problems described in the preceding text.

Currently we use functionality included in WordNet for selecting the concept. It is able to order the concepts found for particular term by internal ranking system which is mainly based on using frequency in English language. Concepts identified by this method can be completely out of context. Therefore we plan to implement the context based selection strategy. Steps involved in this strategy as described by [9]:

1. Define the semantic vicinity of the concept (all hypernyms and hyponyms up to level 1) as $V(c) = \{b \in C | c$ is hypernym or hyponym of b$\}$
2. Determine all terms, which could be a concept from vicinity of c by $U(c) = \bigcup_{b \in V(c)} Ref_c^{-1}(b)$.
3. Find most relevant concept on basis of context by $dis(d,t) = first\{c \in Ref_c(t) \mid c\ s\ max.\ tf(d, U(c))\}$.
4. Set the term weights by $cf(d,c) = tf(d, \{t \in T \mid dis(d,t) = c\})$.

Additionally, the identified concepts could be out of the context due to the generality of the WordNet. We plan to use domain ontology to overcome this obstacle in our further research.

3.4 Reducing Term-Document Matrix Dimensions

The vocabulary of software project is sparse but the dimensionality of term-document matrix especially in large projects can be high. Therefore, *LSI* (*Latent Semantic Indexing*) is used for possible improvement in topic identification and extraction [1]. This method is able to dramatically reduce term dimension of term-document matrix preserving the similarity between documents.

It is a huge advantage for clustering algorithm in the next step in terms of speed. Especially for bee inspired algorithms which we use and which are sensible to dimensionality of clustered data.

LSI is based on a mathematical method *SVD* (*Singular Value Decomposition*) originally used in signal processing to reduce noise preserving the original signal. In our case the noise in term-document matrix is synonyms and polysemy. So this technique not only reduces the dimensionality of the matrix, but also removes the noise occurring in natural language. It transforms set of documents and occurrences of terms to vector space. (Fig. 4)

The technique is really powerful. For example, during our testing we reduced the term dimensionality of matrix from 1580 to 77 keeping clustering results at almost the same quality. On the other hand, it is really resource consuming. In our testing it took almost 2 minutes to produce the reduced matrix for relatively small software project (180 classes - documents). Also, the matrix has to be recomputed every time any changes are done on the original matrix.

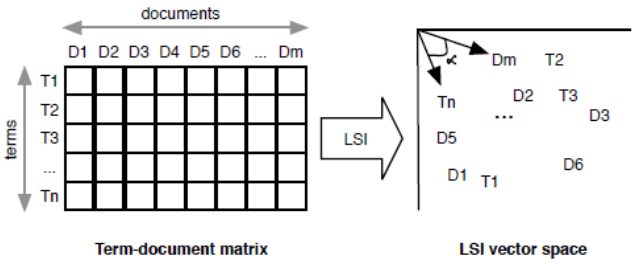

Fig. 4. LSI. [1]

Therefore, we implement this feature as optional in our prototype. It is possible to work completely without it.

3.5 Clustering the Term-Document Matrix with bee behaviour inspired algorithms

In the next step of our method we cluster the reduced matrix to extract the documents containing similar topics.

Originally, we started our research to explore the possible use of swarm intelligence bee behaviour inspired algorithms for source code or text document clustering

in the area of software development. Consequently, we pick up two algorithms from this area and use them in our method.

We chose *FPA* (*Flower pollination*) [5] (Fig. 5 shows visualisation of clustering progress in our laboratory tests) because it needs no input parameters such as number of expected clusters and initial centroids. Therefore we use it instead of the classical heuristic (m x n) / t used in text document clustering to identify the number of expected clusters. Also, it is able to cluster the dataset very well. But its results are not as good as some of the more advanced techniques for text clustering such as *SVM* or *C4.5* (Fig. 21).

We applied *ABC* (*Artificial Bee Colony*) algorithm [6] for optimization of clustering results. It needs initial inputs as a number of expected clusters and centroids and is very competitive in this domain.

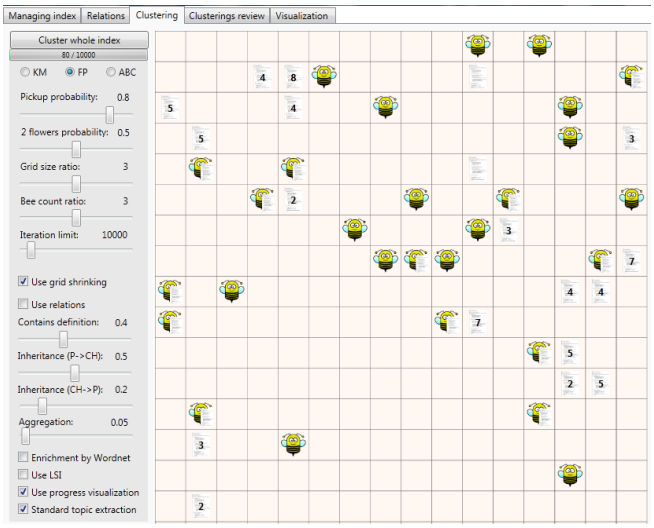

Fig. 5. Visualisation of clustering via Flower Pollination Algorithm.

We used the output of *FPA* (number of clusters, cluster centers) as the input for *ABC*. We modified *FPA* algorithm by replacing the Euclidean distance with cosine similarity metric for better results. Also, we proposed a new feature called "grid shrinking". As the number of identified clusters is decreasing, we decrease the size of the grid. So the movement of bees on it is more effective. This feature is switchable and parameterizable via GUI.

3.6 Extracting topics

After clustering we need to label the identified clusters by contained topics - the set of most relevant terms for a particular cluster.

If we have used *LSI*, then we can apply *LSI index* to retrieve the top-n list of the most relevant terms [1] and search the index by documents that are a part of the clusters and their result is a set of terms. In their case they take precisely top 7 terms.

In case we did not use *LSI*, we propose our own method of extracting top terms. We filter the documents in clusters to the ones having cosine similarity with centroid larger than 0.8 (it is in range 0-1). From this set we take the globally top 10 terms determined by *tf*idf* and use them as the topic label for the particular cluster. Alternatively, we can simply take the top n terms from each centroid cluster.

It is also possible to use one of the advanced techniques [7].

3.7 Visualising results

The last step of our method is to visualise our results. Through previous steps, we have saved the information about indexed documents, relations between them, identified clusters and topics in the database. In this step, we visualise the information in a 3D graph (Fig. 6) with code name *Tent graph* to display the project structure in a uniform and well-arranged way.

The top node represents solution and subnodes are its particular projects. The leaves are classes and interfaces. The graph is implemented in modular way and it is able to show code authors and users, patterns and antipatterns in the code, finished and uncompleted modules, etc.

On granularity level of classes it displays identified clusters in a particular colour. Therefore, the classes and interfaces from particular cluster are tinted with the same colour. The intensity of it represents the similarity between the particular document and the centroid of its cluster. After clicking on the document all other documents from its cluster are highlighted and the topic (set of terms) is displayed in right top corner (Fig. 19). The user can comfortably click on the cluster and highlight only important classes (Fig. 19). Also the list of identified topics is present when no class is selected. Classes and interfaces for particular project are organized in distribution matrix.

With granularity level of methods we use the graph with the code name *Manhattan* (Fig. 20). You can access it by double clicking on a particular class/interface in the *Tent graph*. It displays all methods contained in a class like skyscrapers. The methods from particular cluster are tinted with the same colour. The intensity of it represents the similarity between the particular document and the centroid of its cluster. The height of the skyscraper represents the length of a particular method.

4 Additional Project Information

Currently we are working on expanding the capabilities of our methods to provide more comprehensive knowledge about the source code and the projects. This chapter contains methods for supplemental information [17] we recognize together with topic identification. We will observe and analyze the synergy of this methods and their information about users, authors, tiers and topics (e.g., which author has knowledge

about particular expert domain, which author use *copy-paste technique*, which author has better pattern/antipattern ratio, etc.).

4.1 Tier recognition

Tier recognition is used to automatically identify domains in three-tier based systems. It allows, for example, to estimate developer's orientation on a given tier or to simplify navigation in source code trough clustering.

In our approach we determine tiers in two ways.

Standard Types

Standard types are types in well-known frameworks that represent building blocks of many source codes. The aim of this method is to search for places in given source codes where these types are used to determine their tier using knowledge of relationships between used standard types and code tiers.

Table 2 presents a fragment of a lookup table that represents relationships of some standard .Net namespaces to three code tiers. The table is constructed manually in our work, but we are planning to use automated crawling techniques and clustering algorithms in the future.

Table 2. A fragment of a lookup table, which maps .Net namespaces to tier ratios

Namespace	Presentation tier	Application tier	Data tier
System.ComponentModel	0.4	0.3	0.3
System.Data	0.05	0.05	0.9
System.DirectoryServices	0.15	0.7	0.15
System.Drawing	1.0	0.0	0.0
System.Globalization	0.6	0.1	0.3
System.IdentityModel	0.2	0.8	0.0
…	…	…	…

Fig. 7 shows steps to determine tier association for given type declarations. First, used standard types are extracted and their namespaces are identified. Using a predefined lookup table, tier ratios of the extracted namespaces are determined.

If a namespace is not found in the lookup table, its parent namespace is searched for, and so on up to the root namespace. If not even the root namespace is found, the given namespace is ignored.

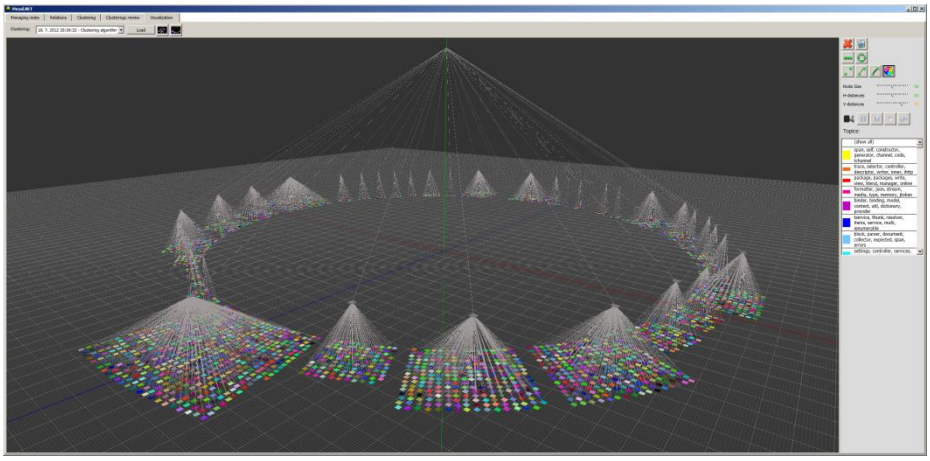

Fig. 6. Tent graph.

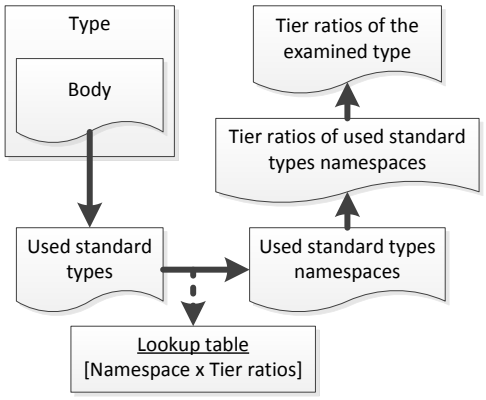

Fig. 7. Determining tiers by examining used standard types.

In our approach we calculate the arithmetic average of all extracted ratios for each tier separately to get the final ratio. Fig. 8 shows an example result of this method. Tiers were determined for a fragment of a system, where mostly the presentation tier is present. Empty rows represent types, which were not determined, as they can belong to any tier.

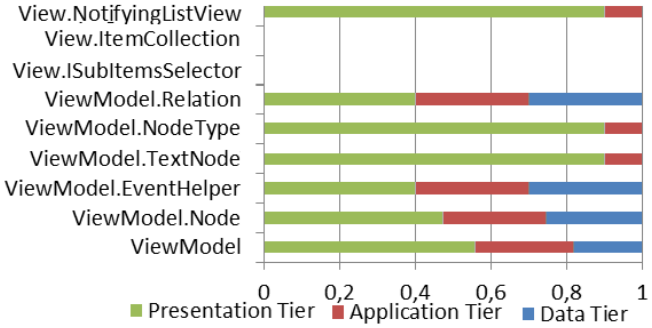

Fig. 8. Example - final ratios of examined types.

Keywords

It is common that the name of a type in object-oriented source code describes its purpose. A type with the name "DbCommand", for example, suggests that it represents a database command. Our method uses this practice to identify known keywords in code identifiers to recognize the tier of a given type. Four identifiers are used for this purpose: *type name, base type name, namespace and project name*.

Keyword dictionary, which represents a set of known keywords and their tier assignments, is used as an input. Each tier assignment of a keyword is perceived as a rate, which defines how much the keyword is specific for a given tier. An example of a simple manually constructed dictionary is presented in the following table.

Table 3. Example keywords dictionary

Keyword	Data	App	Presentation
Data	0,90	0,10	0,00
Table	0,60	0,00	0,40
Workflow	0,10	0,80	0,10
Control	0,00	0,00	1,00
…	…	…	…

Identifier must be separated into words that can be compared with keywords in the dictionary. For example in case of interface "IDBTable" its name will be divided into "DB" and "Table", ignoring first letter "I" that is usually used to mark interface types. For this purpose, regular expressions that define the split points in the identifier's name are used.

After extracting the words from the identifiers, the partial association rate of each identifier type is computed. Each partial rate is in the range of <0.0, 1.0>. Words, which haven't been successfully matched with any keyword, are not included in the computation. Therefore they are not lowering the resulting rate. These partial rates are merged into final weighted rate. For this purpose for each identifier type a weight in

the range of <0.0, 1.0> must be given. Example of these weights is presented in the following table.

Table 4. Example identifier type weights

Identifier	Weight
Name	1
Base Type Name	0.7
Namespace	0.6
Project Name	0.6

Pseudo code shown in the following figure represents computation of final weighted rate.

```
GetTierWeightedRate(IN: weights[], IN: unitRates[], IN:baseRates[],
                    OUT: weightedRate)
 weightSum = 0
 for i in [0..3] :
    if baseRates[i] != 0 : weightSum+=weights[i]
 if weightSum == 0 :
   weightedRate = 0
 else:
   for i in [0..3] :
      if baseRates[i] != 0 :
         weightedRate+=unitRates[i]*weights/weightSum
```

Fig. 9. Pseudo code of final wighted rate computation.

Case Study

In this part we are going to show an application of this method on a very small set of types (Table 5).

Table 5. Example types

Type	BaseType	Project	Namespace
IGetBlob Helper		Frm. DataInterfaces	DataInterf. Blob
Graph3d Control	UserControl	Graph3d. WinForm	Graph3d. WinForm
IWorkflow Helper		Frm. DataInterfaces	DataInterf. Workflow
IWorkflow Entity		Frm. DataInterfaces	DataInterf. Workflow
WSFrmData	DataContext	Frm.	DataClasses

Context		DataClasses	
IDBHistorical Table		Frm. DataInterfaces	DataInterf. HistTable

The following charts display computed partial rates for each keyword type.

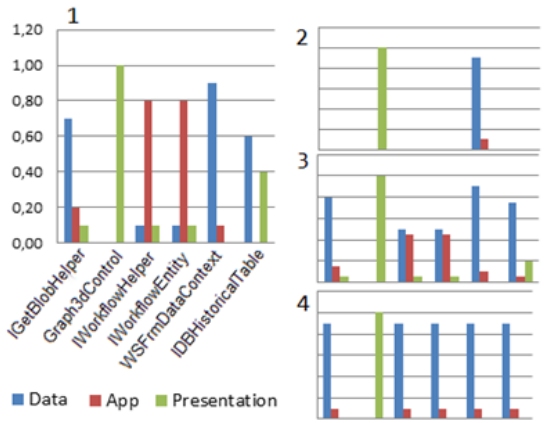

Fig. 10. Unit rate assignments (1-by name; 2-by base types; 3-by namespace; 4-by project).

These rates are then composed into the final result shown in the following figure.

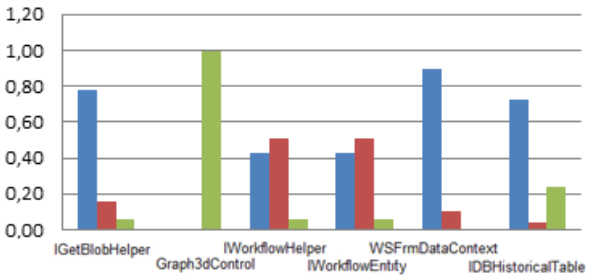

Fig. 11. Weighted assignment rates.

4.2 Source code users recognition

Our method collects information about users and their activities on code entities. Two methods, that gather and process different kinds of such information to increase collaboration in software development team is presented here.

Code entities checkout

In common revision control systems users are able to discover, which source code files are being modified by which users at the current time. This usually helps to avoid conflicting changes of multiple users in a single file.

We go deeper into files and look for concrete entities the source code is composed of. This enables us to determine, which specific parts of source code are being modified and lock for modification only those parts rather than whole files.

The following is the algorithm this method uses to determine currently edited source code entities:

```
GetChangedCodeEntities(OUT:changedCodeEntitySet)
  changedLocalFiles = GetChangedLocalFiles()
  for localFile in changedLocalFiles
    originalFile = DownloadOriginalVersionFromRCS(localFile)
    changedLineIndices = Compare(originalFile, localFile)
    originalFileAst = ExtractAst(originalFile)
      for lineIndex in changedLineIndices
        changedCodeEntity= originalFileAst.GetCodeEntityAt(lineIndex)
        changedCodeEntitySet.Add(changedCodeEntity)
```

User activity on source code entities

We monitor concrete activities that users perform on source code entities. Activities include modifying of a source code entity, pressing a mouse button over it, reaching it in the source code etc.

We measure the *intensity* of activities performed by each user. Each activity is assigned a value in the closed interval <0.0, 1.0>. 1.0 means the user is performing the activity intensively, while 0.0 represents inactivity.

Final intensity of a source code entity

A single user can perform multiple activities on a single source code entity at the same time. This includes different activities as well as the same activity performed at different places (e.g. in different code editors). Fig. 12 shows, how the final intensity of multiple activities performed by a single user on a single source code entity is calculated.

First, from all activities of the same type the one with the highest intensity is taken. This gives information, what different activities and how intense the user performs on the source code entity. By summing up all these partial activities the final intensity is calculated.

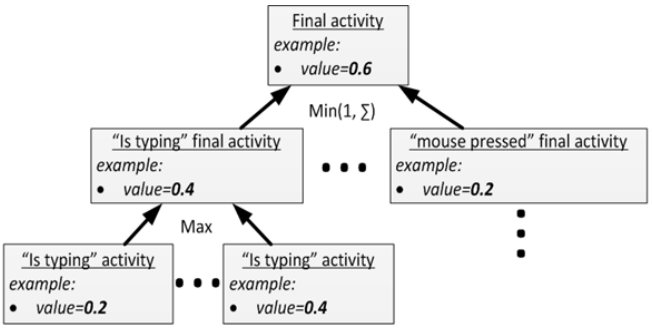

Fig. 12. The final intensity calculation of all activities performed by a single user on a single code entity.

Activity initial intensity and cooling of activities

Each type of activity has a predefined maximum and minimum intensity. When an activity occurs, its intensity is maximal. When it is not performed for a period of time, its intensity starts to decrease down to the minimal value. We call this process *cooling of activities* and it expresses the decreasing interest of the user for the source code entity.

Case study

Fig. 13 shows a prototype we implemented. Three activities are monitored for a single user:

- In viewport – the code entity is reached in a code editor (scrolling etc.)
- Mouse down – the user pressed the mouse over the code entity
- Is typing – the user types into the code entity

The figure also shows how activities are being cooled down.

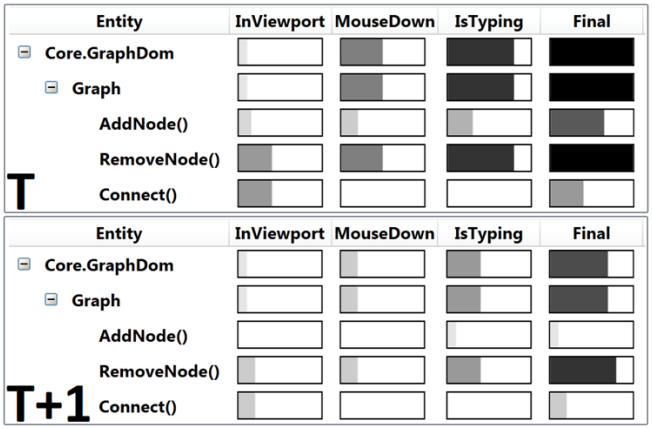

Fig. 13. Activities for a single user. Activities are being cooled down.

4.3 Source code authors recognition

Introduction

To evaluate programmer's coding skills and quality of his work, we usually review some set of functional units - code entities and apply selected metrics. In collaborative environment many authors participate in source code development, so we must distinguish particular author contributions on given code entities. Everyone, who changes the content of a source code file, can be an author of contained source code. In our approach, we divide authors into three basic groups:

- real authors, who modify logical nature of source code (add, modify or delete code entities)
- editors, who modify form of a source code, but not its logical meaning (they are refactoring, sorting entities, formatting code, …)
- reviewers, who comment code or update code due the newer version of used libraries

Authors, members of a development team, share and contribute to source code via Revision Control System (RCS). Author's contribution is defined by a set of changes in one or more source code versions. Change information should cover: author, commit ID, date of change and type of change (add, edit, delete, move...).

There are code entity authors, whose source code parts have persisted to latest or particular source code version and authors, whose source code parts were deleted or considerably modified by the time. In code evolution every version is based on previous versions to certain degree. Ideas represented in older versions are kept, transformed or reused in newer versions. Therefore, it is a matter of principle to assess source code in its whole history, not only in resultant form. For proper authorship metrics, we must consider both persisted and perished changes.

Difference built AST

Our intention for proper authorship evaluation comes from the use of a Microsoft Team Foundation Server (TFS) Client extension for Visual Studio called *Annotate* [10]. Annotate downloads every version of a particular file and annotates the output with attributes showing the changeset, date, and user who last changed each line in the file. The implementation does not show deleted lines [11]. We can say it works on principle of overlapping author information for individual lines of code using line oriented changes from first to latest source code file version (Fig. 14).

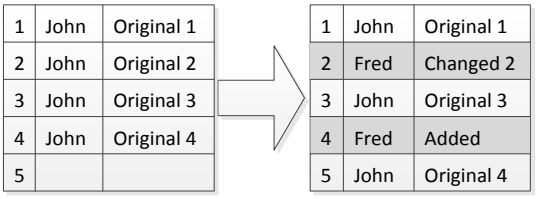

Fig. 14. Principle of Annotate function.

Annotate is based on file and line representation of source code versions/changes in TFS. It is a fast method, but it takes into account only changes, which persisted to the latest version; therefore, it is inaccurate for our authorship metrics. Another disadvantage is the fact, that it ignores deleted lines. Furthermore, this solution does not solve similarity of source code parts, thus moving a part of code is interpreted as deleting lines in one place and adding lines in another place. This also deforms authorship information.

To address the disadvantages of Annotate (or code line representation of changes in general), we created a solution, where, for every version of source code, line changes are projected to code entities. This information is joined to histories of code entity changes. This approach determines authorship of source code entities in object oriented paradigm, where syntax units can be represented as nodes in an Abstract Syntax Tree (AST). It is based on extraction of source code entity changes. The extraction can be divided into several phases (Fig. 15):

1. Extraction of source code files from a software solution. Files can be added into, moved within or deleted from the software solution. It is important to identify all source code files contained in the solution throughout its history.
2. Extraction of source code from all files valid for given versions. This is done repetitively for each two adjacent versions.
3. Representation of each two adjacent versions as ASTs using appropriate software tools [12]. Transformation to AST representation is restricted to level of meaningful syntactical units (classes, functions, properties). Lower level content is represented as lines.
4. Comparison of selected source code versions using *Diff* function based on solving the longest common subsequence problem [13]. Output of *Diff* function can be represented as a set of code changes, which can be of type Add, Delete or Modify. Each change holds information about the author and the range of affected lines – add in newer version of source code, delete in older and modify in both versions.
5. Mapping of code differences to code entities and representation of source code entity changes. In this phase, we create source code entity changes, which are relations between source code elements and source code changes. Code change falls to code entity, if intersection of their ranges is not an empty set. One change can fall to:
6. one code entity in new, old or both compared versions of source code
7. to multiple code entities in one level of AST (for example two functions)
8. to multiple code entities in multiple levels of AST (method, class, namespace)

9. One code change can produce many code entity changes, so it is important to create relations (between change information unit, old and new version of a code entity) only between syntax units of the same type and on the same level of AST.

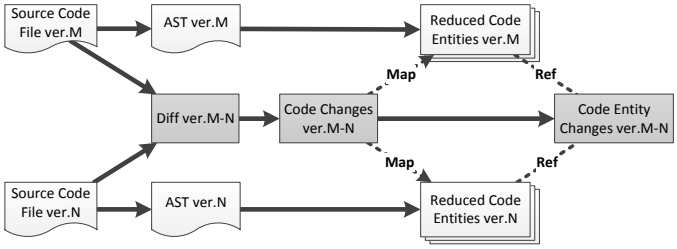

Fig. 15. Determination of authorship is based on extraction of source code entity changes, divided to several phases.

The following pseudo code represents presented extraction process:

```
ExtractCodeEntityChanges(IN: solutionPath, OUT: CodeEntityChanges[])
for filePath in ParseSolution(solutionPath)
  oldSrcCodeFile = null
  for newSrcCodeFile in GetHistory(filePath)
    ast = ParseAst(newSourceCodeFile)
    newCodeEntities = ReduceCodeEntities(ast.Root, empty)
    if oldSourceCodeFile is not null
      codeChanges[] = GetCodeChanges (oldSrcCodeFile, newSourceCodeFile)
      codeEntityChanges+=MapCodeEntityChange(oldCodeEntities,
newCodeEntities, codeChanges)
    oldSrcCodeFile = newSrcCodeFile
    oldCodeEntities = newCodeEntities
```

Fig. 16. Extraction of source code entity changes pseudocode.

The result of this process is represented by sets of changes attached to corresponding code entities. For each code entity, its change history can be constructed. Using this information we can construct relations between versions of abstract syntax trees (Fig. 17).

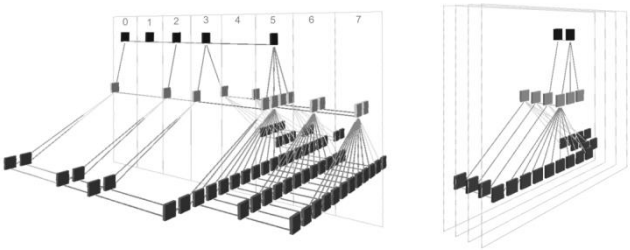

Fig. 17. Relations between versions of abstract syntax trees.

The author of each change can be determined and the authorship of each entity can be evaluated and visualized (Fig. 18).

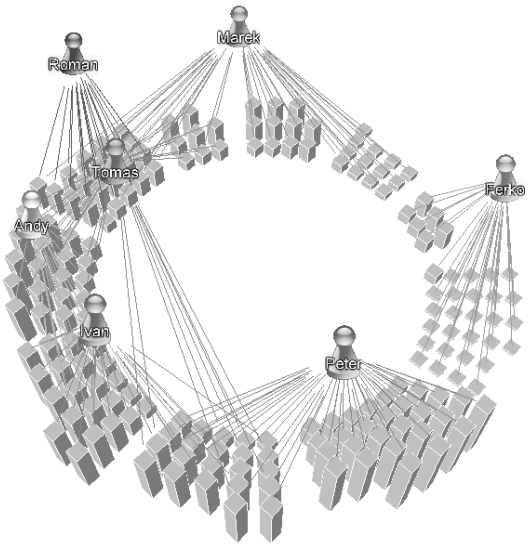

Fig. 18. Authorship of source code entities presented in *Tent graph*.

The extracted information provides variables for various metrics. The variables are:

- author
- entity
- type of change
- line range of change

Also this process has some drawbacks, which can affect the accuracy of metrics results. The most significant are:

- Change in the order of code entities at one AST level. This change is represented by *Diff* as removed and added entities. We would need to compare the similarity to other entities, rather than just using only the information describing the change.
- Renaming of entities. In this case, a relation between entities with different identifiers (name of entity and its placement in AST) is changed. Using only the extracted information, it is not possible to determine, whether the change represents a rename of an entity or it is caused by some other change overlapping neighbour entities.
- Renaming of the identifier of a superior entity. This change is represented as a change of a superior entity. Child entities are not changed. In the process of history construction, entity versions cannot be mapped using their name and placement in the AST. This scenario is interpreted as extinction of the original entity and crea-

tion of a new entity. Information concerning the similarity of entities would also be needed.

- A multiline change overlapping multiple entities at single AST level. For example, if two adjacent entities are affected by a single change, four change information units are created. Each change information unit relates to different combination of entities (two original and two new versions). An entity version graph is created in the process of entity history construction. Therefore, it is not clear, which graph path represents the entity history. Also here, the information concerning the similarity of entities would be required.

From previous, we can sum up, that the main problem in the authorship evaluation is the following. An author is defined by changes, that he made in some version and a part of source code. But if we want to evaluate authorship in whole history of code entity, we must match code entities between several versions of source code, which is not trivial due to change of identifiers of code entities and changes of entities positions in AST [14]. Identification of source code entities is given by their similarities or matching [15].

Conclusion and future work for author recognition

Deficiencies in previous solutions led to design of a RCS based on the AST representation. AST RCS does not substitute RCS used to manage source code versions in daily collaborative development. AST RCS is a supplement of RCS. It imports file and line oriented data and transforms them to historical AST, containing linked code entity versions, using principles similar to [14] and [15], eliminating all deficiencies presented previously. This platform offers unified representation of data imported from several RCSs.

AST RCS is also used for source code marking, where source code parts are labeled with metadata. These metadata – marks – can, among others, contain information about source code quality – effectivity of code, use of code patterns, occurrence of design patterns, smells, etc. This way, we can not only use quantitative, but also qualitative variables in authorship metrics to asses developer's contributions and effects, these contributions have to source code quality. By code marking, we can also label concepts reused in source code development process, identified using methods as presented in [16]. Besides detection of identical source code reusing (copy), source code parts move or refactoring, provided by AST RCS platform, this allows to identify application of concepts or ideas of other authors.

Application of authorship metrics in the context of source code entities by using this approach should be much more accurate and resistant then code line based metrics to specific, but frequently occurring source code changes presented in this paper. The creation of qualitative authorship metrics is matter of metadata type available and can contribute to construction of user profiles (developer profiles) in the wider context of our next research.

5 Prototype MEAD.NET for Topics Recognition

We implemented prototype in .NET framework, C# language and named it Mead.NET. It consists of four screens:

1. Managing index – with adding and viewing indexed documents.
2. Relations – with view of identified relations.
3. Clustering screen – to adjust parameters and view the process of clustering.

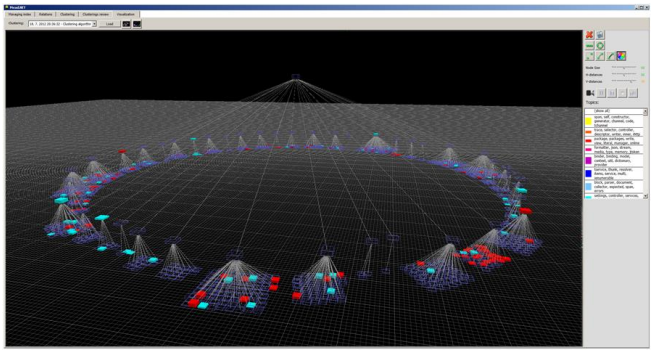

Fig. 19. Tent graph with selected topics.

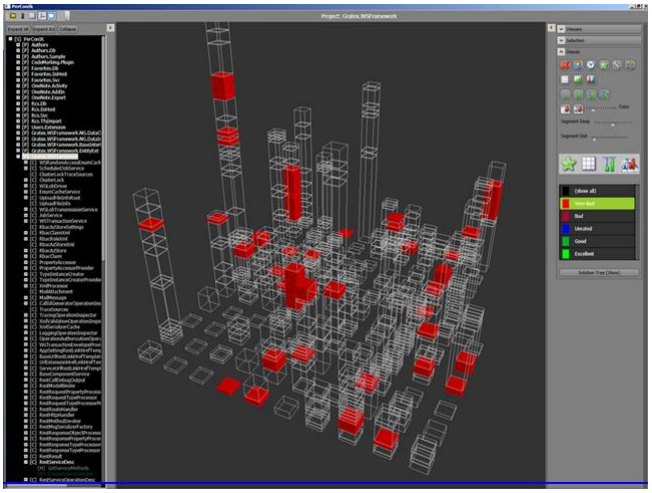

Fig. 20. Manhattan graph with selected topics.

4. Visualisation – to view the graph for a particular clustering (Fig. 6, Fig. 19 and Fig. 20).

6 Case Studies, Evaluation, Further Research

The main problem with the testing of our approach is the non-existence of a sample dataset. We propose the creation of a dataset containing a project ideally clustered and labeled with topics.

Our goal is to show that our changes and add-ons to proven and useful method [1] tend to better results. We have done it in two steps.

First of all, one of the crucial parts of this method is clustering. So we tested used unsupervised clustering algorithm (*FPA*) together with *LSI* in comparison with some supervised algorithms (*SVM* and *C4.5*).

We used *Tech-TC100* dataset. We compared the purity of the clusters identified by mentioned algorithms. The results (Fig. 21) were positive having in mind the future use of *ABC* algorithm which promises even better results.

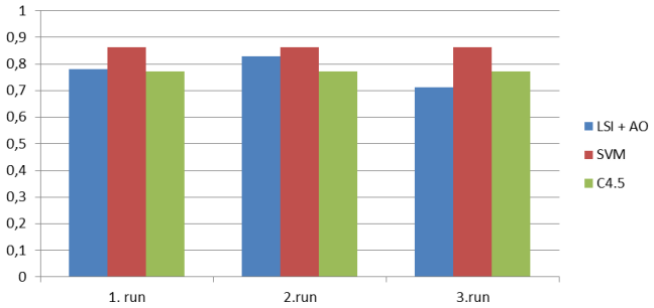

Fig. 21. Clustering results.

Afterwards, we tested the whole process of our method on a wide range of software projects. The results are in Table 6. The method is applicable on all of the tested projects with promising results, but we deal with time consumption methods in complex projects and huge applications.

The tested projects are of varying size and complexity. The tests were performed with the following configuration:

- Size of the side of the grid (gs) is specified by formula $gs = \sqrt{10.(Number\ of\ documents)}$
- Bee count is equal to grid side size.
- We used enrichment via relations. The enrichment via Wordnet ontology was turned off.
- Each project was tested with 100, 1000 and 10000 clustering iterations.

We used research (Mead.NET, PerConIk), commercial (Asp.NET MVC, Robot Emil) and open-source (Quartz.net, IronPython) projects. In the case of research projects some of the identified clusters were analyzed by developers of the particular project. The results are promising. Most of the clusters include the documents within similar topic. Also the extracted top terms represent the topic which is contained in

them. Few exceptions mostly due to the generality of Wordnet were identified. Consequently, the enrichment via Wordnet was disabled for further testing.

Every clustering provides slightly different view on the software project. The main parameter is the number of iterations which directly influences the granularity and the purity of clusters.

Specialized datasets in the form of categorized structural parts of the software project on different levels of granularity will be needed to thoroughly test the impact of proposed approaches. The categories will be topics identified in the particular software project.

For daily use application we propose two approaches. First approach (Fig. 22) consists of the following steps:

1. Use method presented in this paper to cluster software project and extract topics. Use of *FPA* algorithm. Number of iterations depends on size of the project.
2. If the software developer adds new classes or methods, system can insert them randomly to the grid, which is the result from the first step. Then, it can cluster the set for a reasonable number of iterations. It will take only milliseconds and the user will not wait for the latest results.
3. In the last step we apply optimization of existing clustering results through *ABC* in the next parallel thread until the next change in project.

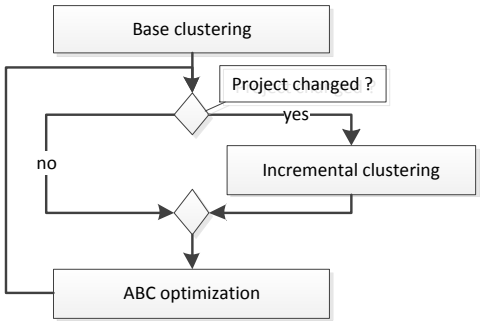

Fig. 22. Incremental clustering approach.

Second approach (Fig. 23) is designed with performance in mind. It consists of the following steps:

1. Use *FPA* to cluster and extract topics as in the first approach.
2. Add new classes or methods to existing clusters based on the similarity with cluster centers. It will take milliseconds and minimize the time consumption for clustering.
3. Optimization via ABC until the next change in the project.

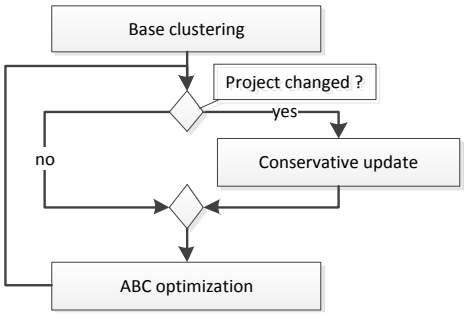

Fig. 23. Update approach.

7 Future Work

Our primary goal is to create a dataset suitable for testing the extraction of topics from the source code. The idea is to manually label classes with topics in some open source project of sufficient size (cca. 300 classes). With this dataset we will be able to thoroughly test the contribution of term-document matrix enrichment in topic extraction.

As we mentioned before WordNet is not ideal for our particular field. Therefore, our plan is to extract our own ontology from software projects, *MSDN* (*Microsoft Developer Network*) and *SDN* (*Sun Developer Network*) which could be a lot more suitable.

Also, implementing time optimization (*Incremental clustering* and *Update approaches*) for daily use and another term extraction algorithms in our prototype can contribute to better results. Furthermore, we would like to add a module for parsing projects written in Java to test the suitability of our method in a different programming language.

Table 6. Evaluation on real-world projects.

Project	summary counts					optimal algorithm settings (*)		iterations	duration of		found	result
	prj/srcFiles	code lines	classes	interfaces	methods	Bee Count	Cluster Cells		indexing	clustering	clusters	shrink
Mead.Net	10/88	15 000	**107**	3	442	32	1024 (32x32)	100	0:08	0:05	54	45%
								1000	0:08	0:45	24	22%
								10 000	0:08	8:05	11	10%
PerConIK	45/466	81 200	**757**	46	2 720	87	7569 (87 x 87)	100	0:18	0:41	423	56%
								1000	0:18	4:32	221	29%
								10 000	0:18	41:00	101	13%
PerConIK+WsFm	52/650	212 000	**1923**	(?)	7 072	138	19044 (138 x138)	100	0:32	2:30	1507	78%
								1000	0:32	24:10	830	43%
								10 000	0:32	(4:00:00)	(400 ?)	(20%)
GKO.RIMIS	7/87	19 000	**141**	6	255	37	1369 (37 x 37)	100	0:10	0:02	79	56%
								1000	0:10	0:18	42	29%
								10 000	0:10	2:30	22	15%
GKO	17/705	230 000	**1021**	158	6 679	100	10000 (100x100)	100	0:25	2:00	679	66%
								1000	0:25	14:45	371	36%
								10 000	0:25	1:40	151	14%
Quartz.Net	4/299	65 000	**323**	46	2 221	56	3136 (56x56)	100	0:24	0:10	165	51%
								1000	0:24	1:10	91	28%
								10 000	0:24	9:20	35	10%
Iron Python 4	14/952	254 000	**1906**	76	14 316	138	19044 (138 x 138)	100	1:40	4:15	1213	63%
								1000	1:40	35:36	673	35%
								10 000	1:40	5:40	269	14%
ASP.Net MVC	41/2482	342 000	**3207**	123	17 310	179	32041 (179 x179)	100	4:00	13:40	2001	62%
								1000	4:00	1:50:00	1098	36%
								10 000	4:00	(18:20:00)	(500?)	(15%)
Robot Emil	1/6	8 300	7	0	**145**	38	1444 (38 x 38)	100	0:05	0:05	50	33%
								1000	0:05	0:35	31	21%
								10 000	0:05	1:50	13	8%

8 Conclusions

We proposed a method based on previous research [1], and enhancement and combination of methods. We designed changes such as different clustering algorithms, optionality of *LSI*, more advanced parsing, and relation identification. Also, we proposed add-ons to this method such as *WordNet* term-document enrichment and new results visualisation tool.

We implemented the prototype and tested usefulness of our contributions. Our future work is to optimize (*Incremental clustering* and *Update approach*) and prepare the method for daily use, advance ABC clustering algorithm, prepare our own ontology and new dataset for the testing of our approach.

Acknowledgment

Part of this work was published in the Proceedings of the Fourth World Congress on Nature and Biologically Inspired Computing (NABIC 2012) [19]. This work was partially supported by the grants VG1/0971/11, VG1/1221/12, and it is the 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. Kuhn, A., Ducasse, S., Girba, T.: Semantic clustering: Identifying topics in source code. In: Information and Software Technology, vol. 49, no.3, pp. 230-243. ISSN 0950-5849, 2007
2. Karaboga, D., Bahriye, A..: A survey: algorithms simulating bee swarm intelligence. In: Artificial Intelligence Review, vol.31, no.1, pp. 61-85, 2010
3. Návrat, P. et al.: The Bee Hive At Work: Exploring its Searching and Optimizing Potential. INFOCOMP Journal of Computer Science, v. 11, no. 1, pp. 32-40. ISSN 1807-4545, 2012
4. Rajasekhar, A. et al.: A Hybrid Differential Artificial Bee Algorithm based tuning of fractional order controller for PMSM drive. In: Proceedings of the Third World Congress on Nature and Biologically Inspired Computing (NABIC 2011), IEEE, pp. 1-6. ISBN 978-1-4577-1122-0, 2011
5. Kazemian, M. et al.: Swarm Clustering Based on Flowers Pollination by Artificial Bees. In: Proceedings of Swarm Intelligence in Data Mining, Springer Berlin Heidelberg, pp. 191-202. ISBN: 978-3-540-34956-3, 2006
6. Karaboga, D., Ozturk C.: A novel clustering approach: Artificial Bee Colony (ABC) algorithm. In: Applied Soft Computing, vol.11, no.1, pp. ISSN 652-657, 2011
7. Carmel, D., Roitman, H., Zwerdling, N.: Enhancing cluster labeling using Wikipedia. In: Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval (SIGIR '09). ISBN 978-1-60558-483-6, 2009
8. Amine, A., Elberrichi, Z., Simonet M.: Evaluation of text clustering methods using wordnet. In: Int. Arab J. Inf. Technol., pp. 349-357, 2010
9. Hoth, A., Staab S., Stumme G.: Wordnet improves Text Document Clustering. In: Proceedings of the SIGIR 2003 Semantic Web Workshop, pp. 541-544, 2003
10. View File Changes Using Annotate. Team Foundation Server 2010. MSDN, Microsoft. [Online; accessed September 25, 2012]. Available at: `http://msdn.microsoft.com/en-us/library/bb385979.aspx`
11. Hodges, B.: Annotate (also known as blame) is now a power toy. MSDN Blogs, Microsoft. [Online; accessed September 25, 2012]. Available at: `http://blogs.msdn.com/b/buckh/archive/2006/03/13/annotate.aspx`
12. Grunwald, D.: NRefactory. SharpDevelop [Online; accessed September 25, 2012]. Available at: `http://wiki.sharpdevelop.net/NRefactory.ashx`
13. Eppstein, D.: Longest Common Subsequences. ICS 161: Design and Analysis of Algorithms Lecture notes for February 29, 1996. [Online; accessed September 25, 2012]. Available at: `http://www.ics.uci.edu/~eppstein/161/960229.html`
14. Fluri, B., Wursch, M. Pinzger, M., Gall, H.: Change Distilling: Tree Differencing for Fine-Grained Source Code Change Extraction, In: IEEE Transactions on software engineering, Vol. 33, No. 11. ISSN 725-743, 2007
15. Neamtiu, I., Foster, J.S., Hicks, M.: Understanding source code evolution using abstract syntax tree matching. In: Mining Software Repositories (MSR 2005), ACM New York, pp. 1–5. ISBN 1-59593-123-6, 2005
16. Marcus, A., Sergeyev, A., Rajlich, V., Maletic, J.I.: An Information Retrieval Approach to Concept Location in Source Code. In: Proceedings of the 11th Working Conference on Reverse Engineering (WCRE'04). ISSN 1095-1350, 2004
17. Polášek, I., Ruttkay-Nedecký, I., Ruttkay-Nedecký, P., Tóth, T., Černík, A., Dušek, P.: Information and Knowledge Retrieval within Software Projects and their Graphical Repre-

sentation for Collaborative Programming. In: Acta Polytechnica Hungarica, Óbuda University, Vol. 10, No. 2. ISSN 1785-8860, 2013

18. Navrat, P., Sabo, S.: What's going on out there right now? A beehive based machine to give snapshot of the ongoing stories on the Web. In: Proceedings of the Fourth World Congress on Nature and Biologically Inspired Computing (NABIC 2012), IEEE, pp. 168-174. ISBN 978-1-4673-4767-9, 2012
19. Uhlár, M., Polášek, I.: Extracting, identifiyng and visualisation of the content in software projects. In: Proceedings of the Fourth World Congress on Nature and Biologically Inspired Computing (NABIC 2012), IEEE, pp. 72-78. ISBN 978-1-4673-4768-6, 2012