Clustering the Source Code

NADIM ASIF, FAISAL SHAHZAD, NAJIA SAHER, WASEEM NAZAR
Dept. of Computer Science
The Islamia University of Bahawalpur
Baghdad campus, Bahawalpur,
PAKISTAN
nasif@softresearch.org, faisalsd@gmail.com, najia_saher@hotmail.com, waseem@uol.edu.pk

Abstract:- The systems are required to understand and present at higher levels of abstractions to perform the changes and meet the current requirements. When the changes are performed, the source code drifts away from the existing available system documentation (specifications, design, manuals), which represent the functionality of the software systems. The software systems are developed using the multi-languages with different dialects and scripts. This paper presents a clustering approach using the available source code, documentation, experience and knowledge about the domain and application to cluster the source code. The source code clustering is used for the purpose of recovering the artifacts, understanding the system and identifying the relationships among the source code to plan, design and execute the changes in the software systems.

Key Words:- Source Code Clustering, Source Code Analysis, Re-Engineering, Reverse Engineering, Design Recovery, Program Understanding, Software Maintenance, Clustering.

1. Introduction

The software engineers perform the code analysis and different maintenance activities by extracting the different types of artifacts at different levels of details using the clustering. The software artifacts exist at implementation, structural, functional and domain abstraction levels. The changes are performed in the software systems and the existing documents are drifted away form the implementation and fail to represent the current implementation of the system. The reverse engineering techniques help to represent the software systems at higher levels of abstraction than code to recover the desired artifacts, understand and comprehend the source code and elaborate the functionality of the software systems to plan, design and execute the different types of maintenance activities. The software engineers cluster the source codes in different formats to represent the systems at higher levels of abstraction for understanding and representing software systems for maintenance activities. The clusters represent the higher level of abstraction of a source code. These clusters help to explore, search the specific features and relationships among the source code, understand the code, functionality and behavior of software systems for maintenance tasks at hand [1,3,4,5]. The source code clustering is used for the following purposes

- For understanding the programs
- For identification (physically and conceptually) of codes (specific line of codes), where changes can be performed
- Categories into physical or conceptual components
- To model and perform changes.
- To plan, design and execute the changes in the source code and also predicts the impacts of these changes in the source code.
- Monitor the effects of changes

The available source code exist in many forms; may be written in multi-languages or have different dialects and scripts, can not be compiled or have errors and complete code is not available. The software engineers debug the source code to find the relationships and functionality among the source code, associate them with relevant entities to understand, which is a time consuming and laborious task. This paper presents an approach of clustering the source code using the available source code, documents, experience and knowledge of application and domain as required by the task at hand.

2. Background

A cluster is a collection of objects that are similar to one other within the same cluster and are dissimilar to objects in other clusters. The process of grouping the physical or abstract objects into entities of similar objects is called clustering. The set of techniques based on *clustering* to extract the design artifacts from the system's artifacts (source code and available documentation) are used for the purpose of maintenance tasks. The clustering approaches helps user to perform the clustering by adapting the top-down, bottom-up and combination (hybride) of both strategies required by the maintenance task at hand The ***bottom up*** strategy start by placing each object in its own cluster and then merge these cluster into larger and larger cluster, until all of the objects are in a single cluster or certain termination conditions are satisfied. The ***top-down*** strategy divide the cluster into smaller and smaller pieces, until each object forms a cluster on its own or until satisfy termination conditions, such as a desired numbers of clusters are obtained.

The ***hybride*** strategy allow the user to form the clusters starting at any level of available sample data to perform bottom-up and top-down clustering, combination of both strategies to develop the desired clusters to certain levels for the task at hand. The major clustering techniques can be divided into the following categories on the bases of the type of method it adapt to cluster the data objects [2, 29, 30, 31].

Hierarchical Methods : The hierarchical methods can be divided into two major categories on the bases of the strategy, (top down or bottom up), it adapt to cluster the objects; *Agglomerative* and *Divisive* Hierarchical Clustering . Agglomerative hierarchical clustering place each objects in its own cluster and then combines these clusters into larger and larger cluster until certain termination is satisfied. The Divisive hierarchical clustering use the top down strategy and divide the cluster into smaller clusters, until each object from a cluster on its own or satisfy certain termination condition e.g. Agglomerative (AGNES), DIvisive ANAlysis (DIANA), Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH), Clustering Using REpresentative (CURE), Chameleon.

Partition Methods : The typical methods includes k-means, k-mediods. The partition methods first create set of k partitions, then use the relocation technique to improve the partitions by moving objects from one group to another group.

Grid Methods : The grid methods first quantizes the objects space into a finite number of cells that form a grid structure, and then clustering is performed on the grid structure e.g. STatistical Information Grid (STING), WaveCluster, Clustering In QUEst (CLIQUE).

Density-Based Methods : The objects are clustered based on the notion of density. The clusters grow on the bases of the density of neighborhood objects or according to some density function e.g. A Density-Based Spatial Clustering of Application with Noise (DBSCAN), Ordering Points To Identify the Clustering Structure (OPTICS), DENsity-based CLUstring (DENCLUE).

Model Based Methods: The model based methods hypothesizes a model for each of the cluster and then find the best fit of the data objects to that model. These type of clusting methods use the statistical or neural network approach e.g. COBWEB, CLASSIT, Auto-Class.

Hybride Clustering Methods : The methods which integrate the idea of many clustering methods and do not belong uniquely to particular clustering method category. The other types of methods are the fuzzy clustering methods [2].

The clustering approaches classification based on the artifacts recovery can be divided into two categories, automatic or semi-automatic clustering techniques. The automatic techniques [10, 11, 12] use the similarity metric (association coefficient, correlation coefficient or probabilistic measure) to partition the system into related group entities. The semi-automatic techniques perform user-assisted clustering process using domain knowledge and visualization means [6, 13, 14, 15, 16, 17].

Some popular clustering techniques use source code component similarity (Hutchens and Basili,[12]; Schwanke [23]; Choi and Scacchi [21]; Muller *et al* [26,27]). Another class of techniques use the implementation information such as module, directory, and/or package names to derive the subsystems [22], and third type of techniques are based on heuristic search techniques (Mancoridis *et al.*, [24, 25]; Mitchell *et al*, [11,24]).

The Lakhotia [28] also presented twelve reverse engineering techniques based on clustering. The Rigi Tool [26, 27] operates semi-automatically, on a generic set of source code model relation. Rigi user extracts the structural information from the system artifacts and represents that information as a set of relation. The Rigi tool takes these relations as input and displays them as collection of overlapping graphs. The user than manipulate the graph(s) and identifies the source code entities (nodes) that should be clustered. The user identifies the set of entities to cluster by applying graph-theoretic algorithms (i.e. identify the strongly connected components), by quality metrics such as coupling and cohesion or by defining scripts such as searching for nodes (entities) conforming to particular naming conventions. The user produces the opaque model by using the Rigi that does not show the details of the source model. This model may be appropriate for some tasks, and while for others, the details may be beneficial. In

Rigi the nodes of graph are clustered based on source model information such as the names of entities, the clustering is performed by using the interface or by using a procedural script.

Murphy et. al Reflexion model technique cluster source code entities through the use of declarative map [32] to produce a high level model of a system. The declarative map is easy for the user to specify. The declarative map is shorter than procedural map, and the map is likely simple in format, improving the likelihood that the user specifies the desired mapping. With Rigi, however, a user must express the desired clustering of source entities, either manually or programmatically or both based on the entire source model. Even when sufficient clustering is performed to derive a high-level model with Rigi, the model that results is not necessarily a view of interest to the engineer. The Rigi method is driven from the bottom-up, and another way to improve the desired view with Rigi is to apply the domain and system-specific knowledge during the bottom-up clustering process.

Mancoridis et al. [25] describes using automatic clustering to produce high-level system organizations of source code. The approach explains a collection of algorithms that were developed and implemented to facilitate the automatic recovery of the modular structure of a software system from its source code. Automatic modularization is treated as an optimization problem and the algorithms described use traditional hill-climbing and genetic algorithms. An automatic software modularization environment is defined and a case study is shown to illustrate the effectiveness of the modularization technique. Clustering is considered as an optimization problem where the goal is to maximize an objective function based on a formal characterization of the trade-off between inter and intra-connectivity. Kamran Sartipi [15] presented another user assisted clustering technique for architecture recovery based on approximate measure, and compute on the shared properties among of highly related system entities.
Maletic and Marcus applied the information retrieval technique called Latent Semantic Indexing (LSI) for software reverse engineering [17, 18]]. They used LSI to analyze the semantic clusters of the files of Mosaic. The user can interact and navigate the visualizations of the semantical clusters, aided by complementary lower level information about the properties and interconnections between the components of the clusters. Another approach to software reverse engineering is the use of visualization techniques to represent the software entities and their relationships [19, 20] at higher levels of abstraction. Tools that use structural exploration are the Rigi [26], SHriMP [20] and Bunch [24]. SHriMP supports a top-down approach to software exploration while employing a nested-graph visualization technique. Brunch first determines the resources and relations in the source code and store the resultant information in a database. Available source code analysis tools of a variety of programming languages is used for this step. After the resources and relations have been stored in a database, the database is queried and a Module Dependency Graph (MDG) is created. MDG is a directed graph that represents the software modules (e.g., classes, files, packages) as nodes, and the relations (e.g., function, invocation, variable usage, class inheritance) between modules as directed edges. Then the clustering algorithms are used to create the partitioned MDG. The clusters in the partitioned MDG represent subsystems that contain one or more modules, relations, and possibly other subsystems. The final result can be visualized and browsed using a graph visualization tool such as dotty.

3. Source Code Clustering Process

The first step of clustering process is to identify the entities using the available documents, experience and knowledge about the domain and the application. An entity ($\boldsymbol{E}_i$) define/comprehend a concept and is used to represent higher abstraction level of components/modules, data sources and processes in a domain, which are used in the high level models to represent the software systems [6,7,8,9]. The sub-entities represent the lower levels of abstractions as compared to an entity. For example account is an entity in a banking domain and personal account, corporate account are examples of sub-entities represent the specific types of accounts. The user specifies the entities and writes the abstract regular expressions to cluster the source code physically or conceptually.

In the second step, the entities are used to cluster the source codes which represent the conceptual or physical (directories, files) association with the entities. The process is repeated until the desired clusters are formed.

The source code clustering must satisfy the following requirements.

1. **Unit cluster** contains minimum a single line of code.
2. Let **Cp** represent a single cluster of source code formed by using the physical relationships, the files or type of files and directories association with source code.

3. Let **Cc** represent a single cluster of source code developed using the conceptual relationships, the components/sub-components, classes/sub-classes and functions association with the source code.

4. The cluster $\mathbf{C_i}$ will be similar to the cluster $\mathbf{C_j}$ physically, if they have the same line of code in the same sequence then $\mathbf{C_i = C_j}$. Clusters dissimilar physically, if they have different line of code in different sequence then $\mathbf{C_i \neq C_j}$.

5. The cluster $\mathbf{C_i}$ will be similar to the cluster $\mathbf{C_j}$ conceptually, if they perform the same function but may differ physically $\mathbf{C_i = C_j}$. Clusters dissimilar conceptually, if they perform different functions then $\mathbf{C_i \neq C_j}$ **OR** similar physically $\mathbf{C_i = C_j}$.

6. The cluster formed using the entity E_i is represented by .

$$\mathbf{Cp} = \sum_{i=1}^{n} \mathbf{C_i}$$

$$\mathbf{Cc} = \sum_{i=1}^{n} \mathbf{C_i}$$

The clusters **(Cc)** formed using the concepts represented by entity (E_i)**,** which abstract the concepts**.** The components, modules, classes, functions which represents the entity (concepts) implemented in the source code. The source code is organized physically in the files or types of files (*.pp, *.jar , *.exe etc) and directories. The cluster **Cp** is formed using the entity (E_i), which associate the files (code of lines exists in different files and directories) to the cluster.

7. Each entity (E_i) contribute in the cluster has weight equal to 1 ($\mathbf{W_{Ei}}$ = 1). The Similarity and dissimilarity of clusters are represented by $\mathbf{S_o}$ and $\mathbf{D_o}$.

$$\mathbf{S_o = C_{(i,j)}\, S(E_i,E_j)}$$

$$\mathbf{D_o = C_{(i,j)}\, D(E_i,E_j) = T_o - S_o \times 2}$$

Where $\mathbf{S_o}$ is the number of similar objects in cluster $\mathbf{C_i}$ and $\mathbf{C_j}$ and the entities $\mathbf{E_i}$ and $\mathbf{E_j}$ are used to form the cluster $\mathbf{C_{(i,j)}\, S(E_i,E_j)}$**.** The$\mathbf{T_o}$ represent the total number of objects in clusters $\mathbf{C_i}$ and $\mathbf{C_j}$.

The $\mathbf{C_{(i,j)}\ D(E_i,E_j)}$ represent the dissimilarity between the clusters $\mathbf{C_i}$ and $\mathbf{C_j}$ and the centre points of clusters are the entities $\mathbf{E_i}$ and $\mathbf{E_j}$ used to form the cluster.

The $\mathbf{C_{(i,j)}\, S(E_i,E_j) = C_{(j,i)}\, S(E_j,E_i)}$ and

$$\mathbf{C_{(i,j)}\, D(E_i,E_j) = C_{(j,i)}\, D(E_j,E_i)}$$

The mean and average similarity, and dissimilarity of clusters are calculated by using the following equations.

$$\mathbf{C_nS} = \sum_{i=1}^{n}\sum_{j=i+1} \mathbf{C_{(i,j)}\ S(E_i,E_j)}$$

$$\mathbf{C_nD} = \sum_{i=1}^{n}\sum_{j=i+1} \mathbf{C_{(i,j)}\ D(E_i,E_j)}$$

The Similarity and dissimilarity of entity $\mathbf{E_i}$ with the other entities ($\mathbf{E_1, E_2, \ldots\ E_n}$) is represented by the following equations

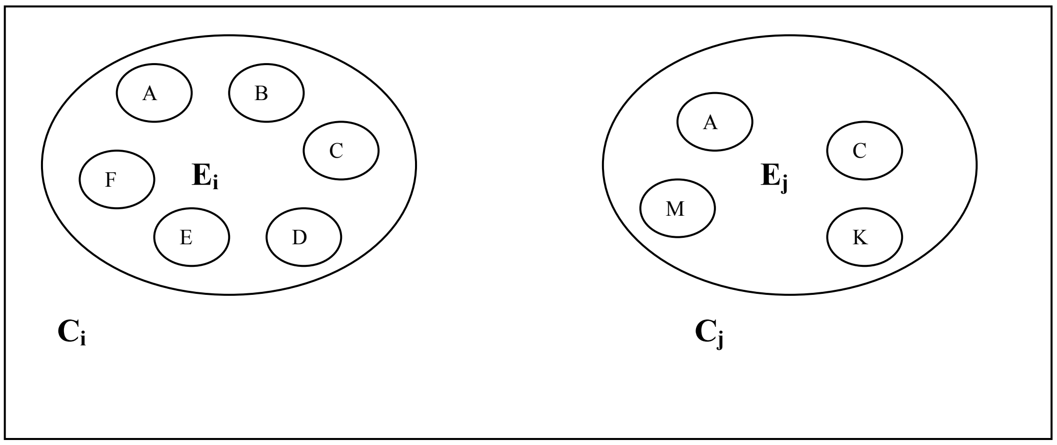
$$\mathbf{C_nS} = \sum_{j=1}^{n} \mathbf{C_{(i,j)}\ S(E_i,E_j)}$$

$$\mathbf{C_nD} = \sum_{j=1}^{n} \mathbf{C_{(i,j)}\ D(E_i,E_j)}$$

8. The cluster $\mathbf{C_i}$ and $\mathbf{C_j}$ will be merged if the difference $\mathbf{C_{(i,j)}\ D(E_i,E_j)}$ = 0 or both have equal number of similar objects then

$$\mathbf{C_{(i,j)}S(E_i,E_j) = C_{(i,j)}D(E_i,E_j)}\ .$$

For example, clusters $\mathbf{C_i}$ and $\mathbf{C_j}$ are formed using the entities $\mathbf{E_i}$ and $\mathbf{E_j}$ in figure 1. The cluster $\mathbf{C_i}$ contains 6 objects (A,B,C,D,E,F) and cluster $\mathbf{C_j}$ contains 4 Objects (A, C, K, M).



The Similarity and dissimilarity of clusters are calculated below.

$$\mathbf{S_o = C_{(i,j)}\ S(E_i,E_j)} = 2$$

$$\mathbf{D_o = C_{(i,j)}\ D(E_i,E_j) = T_o - S_o \times 2}$$

$$= 10 - 2 \times 2 = 6$$

4. Case Study

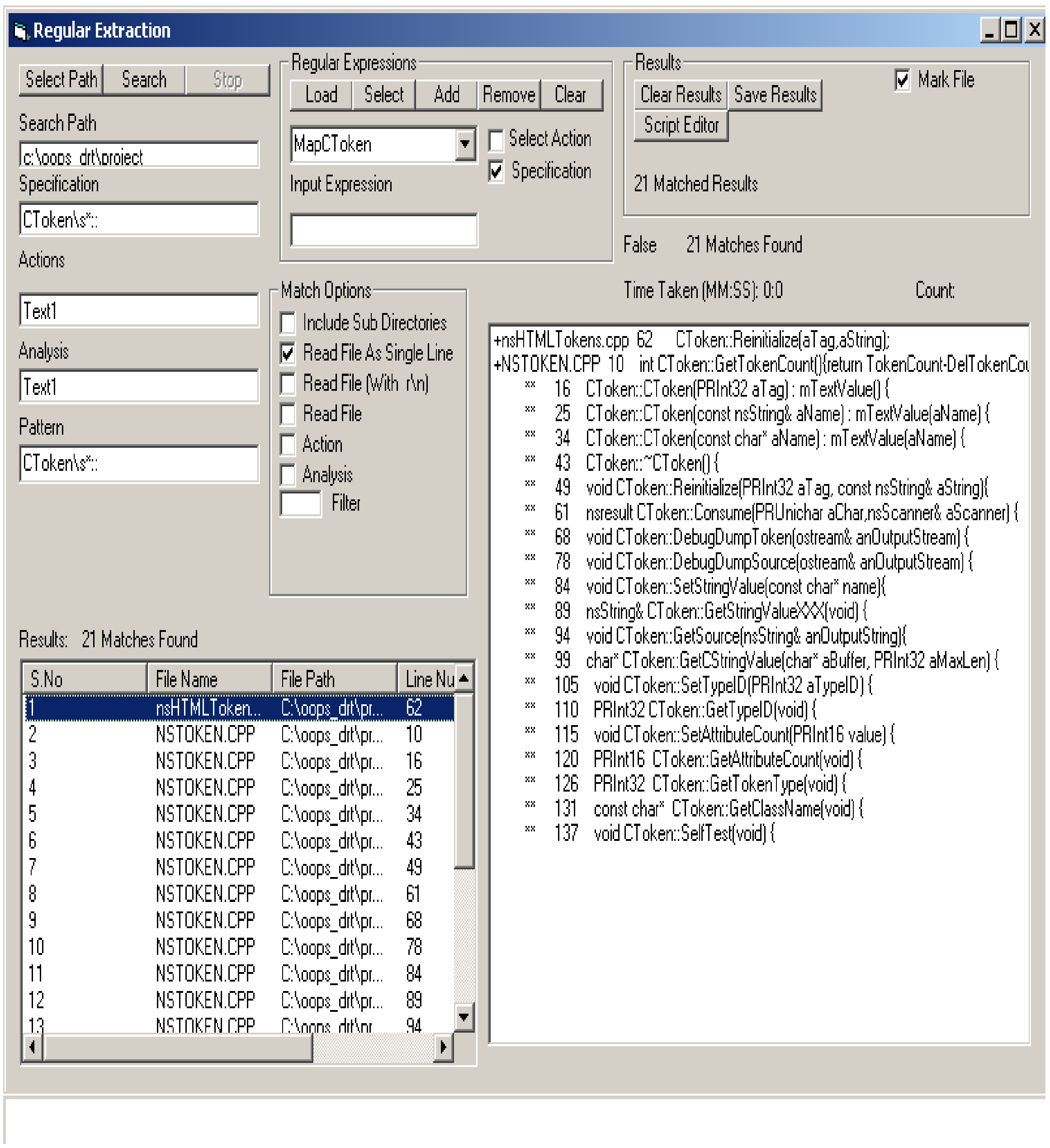
The clusters are developed using the entities ***CToken, Scanner*** and ***Parse***, which are identified from the existing available Mozilla HTML Parser source code, documentation and knowledge about the application and domain. The clusters depicted in figures 2, 3, 4 & 5 represent the conceptual relationship with classes and functions.

```
+AllClasses.txt  7    class CRTFControlWord : public  CToken {

**    8    class CRTFGroup: public CToken {
**    9    class CRTFContent: public CToken {
**    12   class CTokenFinder: public
                     nsDequeFunctor{
**    17   class CTokenDeallocator: public
                     nsDequeFunctor{
**    18   class CTokenRecycler : public
                      nsITokenRecycler {
**    30   class CHTMLToken : public CToken {
**    71     class CToken {

  **     72    class CTokenHandler : public
                     CITokenHandler {
```

Figure 2 Cluster the classes using the Entity CToken



Search Path: c:\oops_drt\project
Specification: CToken\s*::
Actions: Text1
Analysis: Text1
Pattern: CToken\s*::

Regular Expressions: MapCToken
[ ] Select Action
[✓] Specification
Input Expression:

[✓] Mark File
21 Matched Results

Match Options:
[ ] Include Sub Directories
[✓] Read File As Single Line
[ ] Read File (With r\n)
[ ] Read File
[ ] Action
[ ] Analysis
Filter

False   21 Matches Found

Time Taken (MM:SS): 0:0   Count:

```
+nsHTMLTokens.cpp  62     CToken::Reinitialize(aTag,aString);
+NSTOKEN.CPP  10   int CToken::GetTokenCount(){return TokenCount-DelTokenCou
      **    16   CToken::CToken(PRInt32 aTag) : mTextValue() {
      **    25   CToken::CToken(const nsString& aName) : mTextValue(aName) {
      **    34   CToken::CToken(const char* aName) : mTextValue(aName) {
      **    43   CToken::~CToken() {
      **    49   void CToken::Reinitialize(PRInt32 aTag, const nsString& aString){
      **    61   nsresult CToken::Consume(PRUnichar aChar,nsScanner& aScanner) {
      **    68   void CToken::DebugDumpToken(ostream& anOutputStream) {
      **    78   void CToken::DebugDumpSource(ostream& anOutputStream) {
      **    84   void CToken::SetStringValue(const char* name){
      **    89   nsString& CToken::GetStringValueXXX(void) {
      **    94   void CToken::GetSource(nsString& anOutputString){
      **    99   char* CToken::GetCStringValue(char* aBuffer, PRInt32 aMaxLen) {
      **    105   void CToken::SetTypeID(PRInt32 aTypeID) {
      **    110   PRInt32 CToken::GetTypeID(void) {
      **    115   void CToken::SetAttributeCount(PRInt16 value) {
      **    120   PRInt16  CToken::GetAttributeCount(void) {
      **    126   PRInt32  CToken::GetTokenType(void) {
      **    131   const char*  CToken::GetClassName(void) {
      **    137   void CToken::SelfTest(void) {
```

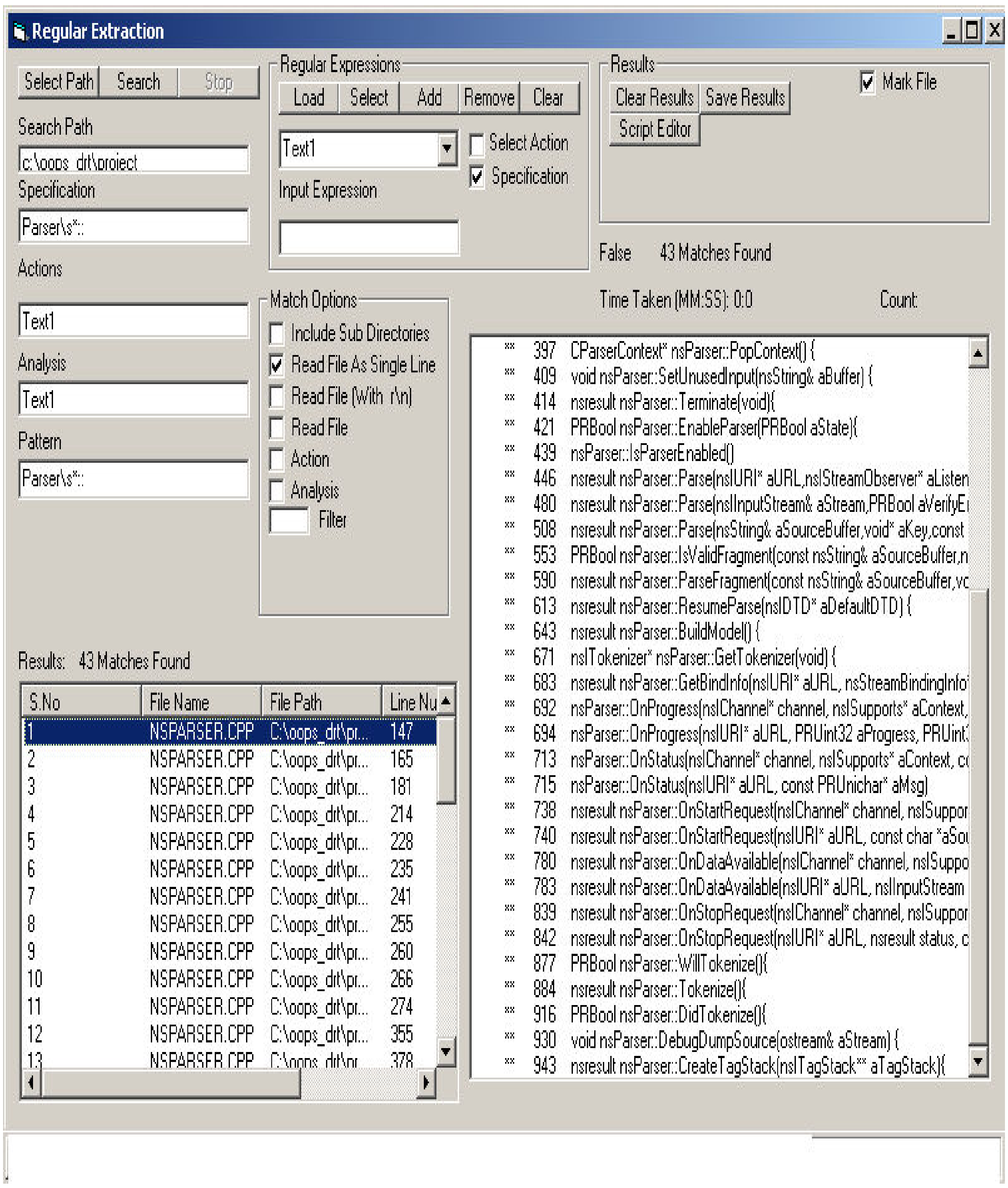
Results: 21 Matches Found

S.No	File Name	File Path	Line Nu
1	nsHTMLToken...	C:\oops_drt\pr...	62
2	NSTOKEN.CPP	C:\oops_drt\pr...	10
3	NSTOKEN.CPP	C:\oops_drt\pr...	16
4	NSTOKEN.CPP	C:\oops_drt\pr...	25
5	NSTOKEN.CPP	C:\oops_drt\pr...	34
6	NSTOKEN.CPP	C:\oops_drt\pr...	43
7	NSTOKEN.CPP	C:\oops_drt\pr...	49
8	NSTOKEN.CPP	C:\oops_drt\pr...	61
9	NSTOKEN.CPP	C:\oops_drt\pr...	68
10	NSTOKEN.CPP	C:\oops_drt\pr...	78
11	NSTOKEN.CPP	C:\oops_drt\pr...	84
12	NSTOKEN.CPP	C:\oops_drt\pr...	89
13	NSTOKEN.CPP	C:\oops_drt\pr	94

Figure 3. Clustering the CToken Class Functions

```
+nsHTMLTokens.cpp  30    CHTMLToken::CHTMLToken(const nsString&
aName,eHTMLTags aTag) : CToken(aName) {
      **     35    CHTMLToken::CHTMLToken(eHTMLTags aTag) : CToken(aTag) {
      **     40    void CHTMLToken::SetStringValue(const char* name){
      **     589      CHTMLToken::Reinitialize(aTag,aString);
```

Figure 4. Clustering the CHTMToken Class functions



Search Path: c:\oops_drt\project

Specification: Parser\s*::

Actions: Text1

Analysis: Text1

Pattern: Parser\s*::

Regular Expressions: Text1 — ☐ Select Action, ☑ Specification

Input Expression:

☑ Mark File

False 43 Matches Found

Time Taken (MM:SS): 0:0 Count:

Match Options:
- ☐ Include Sub Directories
- ☑ Read File As Single Line
- ☐ Read File (With \r\n)
- ☐ Read File
- ☐ Action
- ☐ Analysis
- Filter

```
**  397  CParserContext* nsParser::PopContext() {
**  409  void nsParser::SetUnusedInput(nsString& aBuffer) {
**  414  nsresult nsParser::Terminate(void){
**  421  PRBool nsParser::EnableParser(PRBool aState){
**  439  nsParser::IsParserEnabled()
**  446  nsresult nsParser::Parse(nsIURI* aURL,nsIStreamObserver* aListen
**  480  nsresult nsParser::Parse(nsIInputStream& aStream,PRBool aVerifyE
**  508  nsresult nsParser::Parse(nsString& aSourceBuffer,void* aKey,const
**  553  PRBool nsParser::IsValidFragment(const nsString& aSourceBuffer,n
**  590  nsresult nsParser::ParseFragment(const nsString& aSourceBuffer,vc
**  613  nsresult nsParser::ResumeParse(nsIDTD* aDefaultDTD) {
**  643  nsresult nsParser::BuildModel() {
**  671  nsITokenizer* nsParser::GetTokenizer(void) {
**  683  nsresult nsParser::GetBindInfo(nsIURI* aURL, nsStreamBindingInfo
**  692  nsParser::OnProgress(nsIChannel* channel, nsISupports* aContext,
**  694  nsParser::OnProgress(nsIURI* aURL, PRUint32 aProgress, PRUint
**  713  nsParser::OnStatus(nsIChannel* channel, nsISupports* aContext, c
**  715  nsParser::OnStatus(nsIURI* aURL, const PRUnichar* aMsg)
**  738  nsresult nsParser::OnStartRequest(nsIChannel* channel, nsISuppor
**  740  nsresult nsParser::OnStartRequest(nsIURI* aURL, const char *aSou
**  780  nsresult nsParser::OnDataAvailable(nsIChannel* channel, nsISuppo
**  783  nsresult nsParser::OnDataAvailable(nsIURI* aURL, nsIInputStream
**  839  nsresult nsParser::OnStopRequest(nsIChannel* channel, nsISuppor
**  842  nsresult nsParser::OnStopRequest(nsIURI* aURL, nsresult status, c
**  877  PRBool nsParser::WillTokenize(){
**  884  nsresult nsParser::Tokenize(){
**  916  PRBool nsParser::DidTokenize(){
**  930  void nsParser::DebugDumpSource(ostream& aStream) {
**  943  nsresult nsParser::CreateTagStack(nsITagStack** aTagStack){
```

Results: 43 Matches Found

S.No	File Name	File Path	Line Nu
1	NSPARSER.CPP	C:\oops_drt\pr...	147
2	NSPARSER.CPP	C:\oops_drt\pr...	165
3	NSPARSER.CPP	C:\oops_drt\pr...	181
4	NSPARSER.CPP	C:\oops_drt\pr...	214
5	NSPARSER.CPP	C:\oops_drt\pr...	228
6	NSPARSER.CPP	C:\oops_drt\pr...	235
7	NSPARSER.CPP	C:\oops_drt\pr...	241
8	NSPARSER.CPP	C:\oops_drt\pr...	255
9	NSPARSER.CPP	C:\oops_drt\pr...	260
10	NSPARSER.CPP	C:\oops_drt\pr...	266
11	NSPARSER.CPP	C:\oops_drt\pr...	274
12	NSPARSER.CPP	C:\oops_drt\pr...	355
13	NSPARSER.CPP	C:\oops_drt\pr...	378

Figure 5. Clustering the nsParser Class Functions with Design Recovery Tool (DRT)

The approach has the following features required for clustering the source code.

User-oriented: The approach involves the user and also allows the user to cluster the source code using the experience, domain and application knowledge.

Iterative: The process is iterative and the clusters are formed to the desire level required by the task at hand.

Partial: only the desired source code clusters are formed for the task at hand.

5. Design Recovery Tool

The Design Recovery Tool (DRT) is used in the clustering process [33,34,35,36]. The tool has the following features.

Flexibility: Many issues related to source code exists like language dialects, robustness of extraction mechanism used, incorrect syntax, incomplete code and mix-mode source code. Different tools have limitations in handling all above issues. The DRT is flexible enough to handle all of these issues. Its pattern matching power can be extended by designing new pattern specifications with the help of existing pattern specifications.

Mappings : DRT maps the source code and documents for extraction and abstraction purpose. Through mapping we can increase the extraction performance; specifying the required artifacts constructs.

Extraction and Abstraction : Most reverse engineering tools extract and abstract the artifacts in different styles at different levels, which are not very much relevant to maintenance task at hand. In DRT the extraction and abstraction operations may be refined by the user according to its need.

Presentation : In the case of large and complex systems, the numbers of artifacts are large in number and have different type of relationships make it difficult to present the system artifacts. Using DRT the extracted and abstracted artifacts can be represented in different formats.

Scalable : The reverse engineering tool should be applicable to large systems and different type of source codes (Languages). Through innovative pattern specifications of DRT, it can be used for large systems and can support different languages.

Speed : Speed is also very important attribute because the software under study may have million lines of code and it may not be structured that whole search operation can be performed on subset of code.

Robustness : The reverse engineering tools should have the tolerance of errors, especially in unexpected cases. It is tolerant in that there are few constraints on the condition of the artifacts. For example, we can extract from source that cannot necessarily be compiled. In DRT, the robustness is the key concern and work is in progress to achieve it.

Analysis : The matched patterns may be further analyzed to extract further relationships between the patterns and may be represented in different formats. Different scripts can be used for the analysis of extracted artifacts.

Precision : The tools should have the ability to match the required patterns with accuracy and it should ignore the false matches. We can get the 100% precision in DRT by refining the pattern specifications.

The DRT language for specifying the pattern designs has three parts: Patterns, actions and analysis. Patterns specifications are used to extract the artifacts of user's need, actions to execute after pattern is matched to a portion of a system artifacts and analysis operations that extract a source code from an intermediate representation produced during scanning. We designed our pattern specifications keep in view the requirements of our source code consisting of different programming languages. The simple syntax of regular expression makes it easy for the user to extend the vocabulary of the pattern

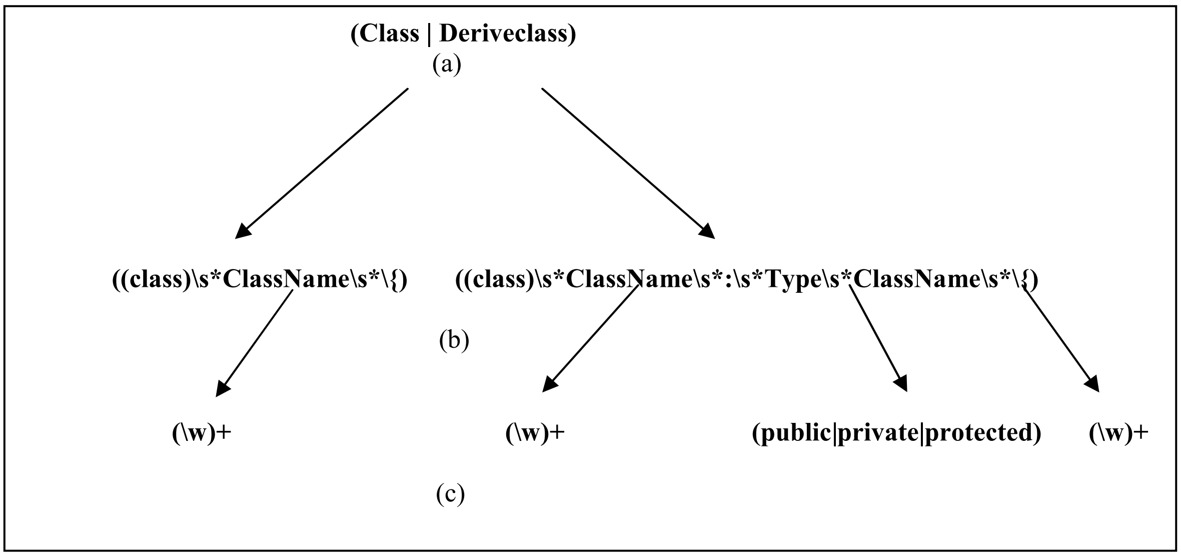

(Class | Deriveclass)
(a)
((class)\s*ClassName\s*\{)
((class)\s*ClassName\s*:\s*Type\s*ClassName\s*\{)
(b)
(\w)+
(\w)+
(public|private|protected)
(\w)+
(c)

Figure 6. Regular Expression Pattern used to cluster the Mozilla HTML Parser Classes

specification language according to requirement to cluster the source code.

The user can design abstract pattern using the existing patterns to cluster the source code. For example, in Figure 6 the “Class” and "Deriveclass" are the abstract reserve word, which represents the regular expression pattern (b) for the classes and derived classes. The regular expression pattern (b) contains the abstract patterns "ClassName" and "Type", which represent the sub regular expression pattern (c) for the class name and class type in this case.

6. Conclusion

The software source code exist in many forms; may be written in multi-languages or have different dialects and scripts, can not be compiled or have errors and complete code is not available. The software engineers debug the source code and find the relationships and functionality and associate them with relevant entities to understand and find the relationships among the different pieces of source code exist in different types of files and directories, which is a time consuming and laborious task. The approach clusters the source code using the available documentation, experience, knowledge of application and domain.

The source code clusters are formed using the entities which represent the concepts implemented in the software source code. The approach clusters the source code conceptually (using conceptual relationships – components, classes, functions, variables) and physically (directories, types of files where the lines of source code exist). The clusters are formed using the top-down, bottom-up and hybride (combination of both) strategy as required by the task at hand to the desired level of clustering.

References

[1] A.K Jain, M.N Murty and P.J Flynn, Data Clustering: A survey. *ACM Computing Survey*. 31, 1999, pp. 264-323.

[2] L. Kaufman and P.J Rousseeuw , *Finding Groups in Data: An Introduction to Cluster Analysis*. New York: John Wiley & Sons, 1990.

[3] Nadim Asif, M. Dixon, J. Finlay and G. Coxhead, Recover the Design Artifacts. *In proceedings of International Conference of Information and Knowledge Engineering (IKE02)*, 24th –27th June, 2002, Las Vegas, Nevada, USA, CSREA Press, pp. 656-662.

[4] Nadim Asif, Reverse Engineering Methodology to Recover the Design Artifacts: A Case Study. In *proceedings of International Conference of Software Engineering Research and Practice (SERP03)*, 23rd-26th June, 2003, Las Vegas, USA,CSREA Press, pp.932-938.

[5] Nadim Asif, Muthu Ramachandran, Recover the Use Case Models. In *proceedings of International Conference of Software Engineering Research and Practice (SERP05)*, 27th -30th June, Las Vegas USA, 2005, CSREA Press.

[6] Nadim Asif, Developing High Level Models for Artifacts Recovery and Understanding Using Statistical Information. *In proceedings of 8th Islamic Countries Conference on Statistic*. 19th- 23rd Dec, 2005.

[7] Nadim Asif, *Software Reverse Engineering*, SoftResearch Press, 2006. (ISBN : 969-9062-00-2).

[8] Nadim Asif, Artifacts Recovery at Different levels of Abstraction. *Information Technology Journal*, 7(1), pp. 1-15, 2008.

[9] Nadim Asif, Artifacts Recovery Techniques, *International Journal of Software Engineering*, Vol.1, No 1, 2007, pp. 26-66

[10] Kunz, T, Black, JP, Using automatic clustering process for design recovery and distributed debugging. *IEEE Transactions on Software Engineering;* 21(6), p515-527,1995.

[11] Mitchell, Brian S., Mancoridis, Spiros, On the Automatic Modularization of Software Systems Using the Bunch Tool. *IEEE Transactions on Software Engineering*; Vol. 32 Issue 3, p193-208, March 2006.

[12] D. Hutchens and R. Basili. System Structure Analysis: Clustering with Data Bindings. *IEEE Transactions on Software Engineering*, 11:749-757, Aug. 1995.

[13] Finniga. P et. al. , The Software Bookshelf, *IBM Systems Journal*, 4, 564-593, 1997.

[14] Muller, H.A et al , A reverse Engineering Approach to subsystem structure identification. *Journal of Software Maintenance: Research and Practice.*5(4),181-204, 1993.

[15] Sartipi, K. ,Kontogiannis K. A User-assisted approached to component Clustering. *Journal of Software Maintenance: Research and Practice*, 00:1-32, 2003.

[16] T. Wiggerts. Using clustering algorithms in legacy systems remodularization. *In Proc. Working Conference on Reverse Engineering*, 1997.

[17] J. I. Maletic and A. Marcus. Supporting program comprehension using semantic and structural information. *In Proceedings of the International Conference on Software Engineering (ICSE 2001)*, pages 103–112, 2001.

[18] A. Kuhn, S. Ducasse, and T. Gˆýrba. Enriching reverse engineering with semantic clustering. *In Proceedings of Working Conference On Reverse Engineering (WCRE 2005)*, Nov. 2005.

[19] M. Lanza and S. Ducasse. Polymetric views—a lightweight visual approach to reverse engineering. *IEEE Transactions on Software Engineering*, 29(9):782–795, Sept. 2003.

[20] J. Michaud, M.-A. Storey, and H. Muller. Integrating information sources for visualizing Java programs. *In Proceedings of IEEE International Conference on Software Maintenance (ICSM'01)*, pages 250–259. IEEE, Nov. 2001.

[21] S.Choi and W. Scacchi. Extracting and restructuring the design of large systems. In *IEEE Software*, pages 66–71, 1999.

[22] N. Anquetil, C. Fourrier, and T. Lethbridge. Experiments with hierarchical clustering algorithms as software remodularization methods. *In Proc. Working Conf. on Reverse Engineering*, October 1999.

[23] R. Schwanke. An intelligent tool for re-engineering software modularity. *In Proc. 13th Intl. Conf. Software Engineering*, May 1991.

[24] Mancoridis, S., B.S. Mitchell, Y. Chen, and E.R. Gansner. Bunch: A clustering tool for the recovery and maintenance of software system structures. *In Proceedings of International Conference of Software Maintenance*, pages 50–59, August 1999.

[25] Mancoridis, S., Mitchell, B.S., Rorres, C., Chen, Y. and Gansner, E. R., Using Automatic Clustering to Produce High-Level System Organizations of Source Code. *In: Proceedings of the Sixth International Workshop on Program Comprehension*, 24th–26th June, IEEE Computer Soc. Press. pp. 45-52, 1998.

[26] Muller, H.A et al., 2007. Rigi. Available from: < *http://www.rigi.csc.uvic.ca*>

[27] Wong, K., Tilly, S., Muller, H. and Storey, M., Structural Redocumentation: A Case Study. *IEEE Software*, Vol. 12, No. 1: January, pp. 46-54, 1995.

[28] Lakhotia, A., A Unified Framework for Expressing Software Subsystem Classification Techniques. *Journal of Systems and Software*, 36, pp. 211-231, 1997.

[29] A.K Jain, M.N Murty and P.J Flynn, Data Clustering: A survey. *ACM Computing Survey.* 31: pp. 264-323, 1999.

[30] Romero, C., Ventura, S. Educational Data Mining: A survey from 1995 to 2005, *Expert Systems with Applications*; Vol. 33 Issue 1, pp. 135-146, July 2007.

[31] N. Anquetil and T.C. Lethbridge, Comparative study of clustering algorithms and abstract representations for software remodularisation, IEE Proc.-Software., Vol. 150, No. 3, June 2003.

[32] Murphy, G., Notkin, D., and Sullivan, K., Software Reflexion Models: Bridging the Gap between Design and Implementation. *IEEE Transaction on Software Engineering*. Vol. 27. No 4: April, pp. 364-380, 2001.

[33] Nadim Asif, Recovery of Artifacts. *International Journal of Software Engineering*, Vol. 2, No. 1, pp. 11-16, 2008.

[34] Ghulam Rasool, Nadim Asif, Software Architecture Recovery. *International Journal of Computer, Information, and Systems Science, and Engineering*. Vol.1, No. 3, 2007.

[35] Ghulam Rasool and Nadim Asif, Design Recovery Tool. *International Journal of Software Engineering*, Vol. 1, No. 1, pp 67-72, 2007.

[36] Nadim Asif, Faisal Shahzad, Najia Saher, Rafaquet Kazami, Imran S. Bawaja, Shahid Naveed, Munsib Ali Waseem Nazar, Shahzad Mumtaz. High Level Models for Artifacts Recovery and Understanding. *Computer and Simulation in Modern Science*, WSEAS Press USA. 2009. ISSN: 1790-2769. ISBN: 978-960-474-117-5