A Scenario Based API Recommendation System Using Syntax and Semantics of Client Source Code

S. M. Shahnewaz, Husne Ara Rubaiyeat, Hasan Mahmud, Md. Kamrul Hasan

***Abstract*—In software development, developers frequently look for the examples and documentation provided by vendor of application programming interface (API) libraries, forums, textbooks and unofficial websites. However, a vast number of example API usage scenarios, that are embedded in the billions of line of already developed code are largely unexploited. In this research work, we present an approach that analyses the syntax and semantics of already developed source code to extract API usage scenarios for the current structural context of the developer. Our approach can extract an API usage scenario merging heterogeneous API elements. We also developed a scenario ranking algorithm based on the interrelationships of API elements. In the course of scenario extraction and ranking we have developed some heuristics based on developers coding experience. We demonstrate a qualitative evaluation by reporting a user study involving users completing four development tasks. Results show that our developed tool takes around 40% less time compared to existing systems. In addition, we present the precision-recall based quantitative evaluation. The high precision and recall values indicate that our proposed system is able to retrieve most of the best-fit API usage scenarios from candidate source files. Therefore, experimental evaluation provide evidence that scenario based recommendation is appropriate to help developers.**

***Index Terms*—API, syntax and semantics of source code, API recommendation system, Scenario extraction, Ranking, Precision, and Recall.**

I. Introduction

Application Programming Interfaces (APIs) have great significance in modern day of software development. To increase the productivity, software developers reuse the code libraries or frameworks through APIs. However, example usage and documentation about the APIs are often incomplete or out of date. As a result, both the novice and expert developers have to face several challenges to learn new APIs. Recently, a field study [1] of API learning obstacles was done over 440 professional developers of Microsoft. According to the opinions and experiences of those developers, five important factors are to be considered when designing API documentation: documentation of intent; code examples; matching of APIs with scenarios; penetrability of API; and format and presentation. Among those factors matching APIs with scenarios is the most desired one to developer community. Moreover, modern Use-case based implementation technique frequently demands a desired chunk of functionality such as ”drawing a rectangle on the screen” or ”sending file via HTTP”. According to a Microsoft developer [1]: *”If it’s not clear how I match APIs with their scenarios, if I need to draw a circle on the screen, and I dont see something that clearly says, ”This is how you draw”, I will say that’s complex.”* In the context of software development, a scenario is defined as a desired chunk of functionality such as ”drawing a rectangle on the screen” or ”sending file via HTTP”. Therefore, API elements which perform collaboratively to achieve a desired functionality are defined as an API usage scenario. However, the main challenge of learning API is discovering API elements which support a scenario. We believe that a vast number of example API usage scenarios are embedded in the billions of lines of already developed code. For example, a developer needs to establish a connection with the database via JDBC. Consider that API elements which establish connection with the database are embedded in the getMysqlConnection() method of the SqlConnection class shown below which is

* Corresponding author.

S. M. Shahnewaz, Hasan Mahmud*, Md. Kamrul Hasan are with the Systems and Software Lab (SSL), Department of Computer Science and Engineering (CSE),Islamic University of Technology(IUT), Dhaka. e-mail: {shawncit, hasan, hasank}@iut-dhaka.edu; Husne Ara Rubaiyeat is with Faculty of Natural Science, National University, Bangladesh; rubaiyeat@yahoo.com

Manuscript received March 12, 2015; revised May 14, 2015.

developed for some other project.

```
public class SqlConnection {
private String driverPath =
    "com.mysql.jdbc.Driver";
private String url =
    "jdbc:mysql://localhost:3306/localbd";
private String username = "root";
private String password = "";
public Connection getMysqlConnection()
    throws ClassNotFoundException,
    SQLException {
Class.forName(driverPath);
Connection con =
    DriverManager.getConnection(url,
    username, password);
return con;
}
public static void main(String args[])
    throws ClassNotFoundException,
    SQLException
{
SqlConnection msc = new SqlConnection();
Connection con =
    msc.getMysqlConnection();
System.out.println("Connection
    successful");
}
}
```

Now, it would be helpful for the developer, if he gets the API usage scenario as shown below:

```
1. private String driverPath =
    "com.mysql.jdbc.Driver";
2. private String url =
    "jdbc:mysql://localhost:3306/localbd";
3. private String username = "root";
4. private String password = "";
5. Class.forName(driverPath);
6. Connection con =
    DriverManager.getConnection(url,
    username, password);
```

From the example shown, we formulate our problem definition as *"To design a scenario based API recommendation system that will extract API usage scenario from already developed source codes"*. The main focus of this research work is that the candidate source files are analyzed to extract matching API usage scenarios for the current structural context of a developer. Our approach has three key contributions over existing proposals for extracting API usage for a given task. First, in scenario extraction, we consider semantics (type and identifier) and structures of the client code. We believe that consideration of syntax and semantics will help to provide better quality API usage scenarios. Existing methods only consider either types or identifiers. Second, our scenario extraction algorithm takes multiple query terms and merges heterogeneous API elements needed to implement a task. As a result, developers get most of the relevant APIs in a single search. Third, we introduced key API based ranking heuristic which ensures that best matching scenarios are placed on top of the recommendation set. To define the key APIs we proposed two heuristics. In course of evaluation of our work, we also developed a client tool MAPIS (Matching API with Scenarios) described in [27]. We choose to perform both quantitative and qualitative evaluation approach. In the quantitative evaluation, we calculated the precision and recall of the retrieved scenarios to evaluate the performance of our system. We designed and conducted a user study to evaluate the usefulness of MAPIS to developers for solving real world problems which involved reuse of existing APIs. The users were able to access the relevant scenarios, understand the examples, and complete the programming task; providing initial evidence that scenario based recommendation is appropriate to help developers. The next section started with a brief description of the related works in this field. After that we have described the details of our proposed approach in section III. The evaluation of our proposed approach is presented section IV. Some future extension of this research work is discussed in section V.

II. Related Works

To reduce the effort of a developer in reusing existing API libraries and frameworks, three major directions have been proposed so far. First, code search engines are developed to query code from software repositories. Second, IDE code completion systems are proposed to provide relevant API elements for the current context of a developer. Third, API recommendation systems are proposed to suggest relevant API usage examples for a given task. In the following sections, we describe the approaches. Code search engines like KODER [2] and KRUGLE [3] index the open source repositories and present relevant source files in response to a query. However, code search engines do not consider the structure of the source code. It treats source code as documents. As a result,

outputs of the code search engine are source files. Code completion system is a feature of an IDE that offer a list of available variables, types and methods based on the current context of a developer. Most of the previous work on code completion systems focused either on re-ordering the list of methods accessible on a given type, or on predicting the method of an API type most likely to be called next in a given context. A recent work [12] in this direction facilitates discoverability of APIs by recommending methods or types, which although not directly reachable from the type a developer is currently working with, may be relevant to solving a programming task. API recommendation system suggests useful API usage example by analyzing already developed source code. Researchers from all over the world have proposed different approaches to extract relevant API usage scenarios from source code. Structural relation of source code elements is matched with query context [8], association rule and sequential pattern based recommendation system [6], semantic feature based code search [9], adding example with API documentation [10], method sequence engineering based reverse engineering system [11], DAG based representation of source code [5] were proposed by different research community. The most recent work IDENTIFIRE [14] of this field describes an approach that mines the intentional knowledge embodied in the identifiers of existing source code. It uses term-method association index to recommend API methods. However, main limitation of this approach is that, they do not consider the source code structure that is the interrelationship among API elements. As a result, complete API usage scenarios are not recommended by this work. Another recent project MACs [6] worked on mining API code snippets for code reuse. MACs suggest API snippets based on association rule and sequential patterns. The limitation of this system is that developers have to give an initial statement to get recommendation from MACs. Moreover, definition of API usage scenarios are not reflected in the recommendation set. PARSEWeb [15] and XSnippet [16] gather the relevant code samples from Google Code Search Engine and perform a static analysis over them to answer the queries of type Tin Tout. The dynamic database (that of Google Code Search) together with the query splitting results in some reported improvements. XSnippet [16] makes use of context information along with a user query for finding relevant snippets. The Strathcona [8] API usage example recommendation tool assist developers in finding relevant fragments of code, or examples, of an APIs use based on the structural context of developers code and example repositories. Reiss [9] proposed a technique that extracts code examples from a repository, extracts semantic features from code examples, clusters them, and finds a representative code example from each cluster. Marcel Bruch, et al. [17], proposed intelligent code completion system that learns from existing code repositories by searching for code snippets where a variable of the same type as the variable for which the developer seeks advice, is used in a similar context. Thummalapenta and Xie describe the SpotWeb [18], a code search based tool, able to mine code examples gathered from open source repositories on the web. The study also proposes a method called coldspots that can detect API classes and methods that are rarely used. The key limitation of the present approaches is that heterogeneous API elements are not merged in a single scenario. For example, consider the SQLConnection class (shown in introduction) that contains the necessary API elements needed to create a connection with the database. Six statements shown in the example API usage scenario is the proper scenario for creating connection with database. However, existing approaches are only able to suggest the six required statements in two separate scenarios as below. The ideal scenario would be the composition of Scenario 1 and Scenario 2.

```
//Scenario 1:
1. private String url =
   "jdbc:mysql://localhost:3306/localbd";
2. private String username = "root";
3. private String password = "";
4. Connection con =
   DriverManager.getConnection(url,
   username, password);
//Scenario 2:
1. private String driverPath =
   "com.mysql.jdbc.Driver";
2. Class.forName(driverPath);
```

So the definition of API usage scenario is not reflected in most of the works. Moreover, current systems only consider either types or identifiers in recommending scenarios. Most of the existing systems do not support multiple query terms and interrelationships of APIs for ranking scenarios. In the direction of API recommendation system, our approach is different from other approaches by three main aspects. First, in

scenario extraction, we defined two heuristics to consider semantics (type and identifier) and structures of the client code. We believe that consideration of syntax and semantics will help to provide better quality API usage scenarios. Existing methods only consider either types or identifiers. Second, our scenario extraction algorithm takes multiple query terms and merges different API elements needed to implement a task. As a result, developer gets most of the relevant API in a single search. Third, we introduced key API based ranking heuristic which ensures that best matching scenarios are placed in top of the recommendation set.

III. Proposed Approach

In any API recommendation system there are four key steps: candidate source file collection, API usage scenario extraction, scenario ranking and representation of candidate recommendation (see Figure 1). In our proposed approach, we consider multiple query terms. Based on those query terms, candidate source files are collected from the already developed and indexed open source software repositories. Hence, the input of our proposed system is a set of query terms and already developed candidate source files. To overcome the limitation of present approaches, in our proposed approach, we introduced and applied some heuristics in scenario extraction and scenario ranking. Those heuristics are defined from the overall user experience [1] of the developers in using unknown APIs.

A. Heuristics

Our proposed API recommendation system applies two types of heuristics to recommend scenarios. First, heuristics for scenario extraction are used to generate the graph from expression statement list. Second, heuristics for ranking are used to identify key API elements from extracted scenarios. Scenario extraction heuristics are as follows:

- *Scenario Extraction Heuristic 1: Object creation rule and method call sequences are key factors in API learning.*
- *Scenario Extraction Heuristic 2: Considering both code structure and semantic (type and identifier) provides scenario with all necessary API element.*

Heuristic 1 gives the hints that developers face difficulties in discovering proper object creation rule and method call sequence to implement a task. Conversely,

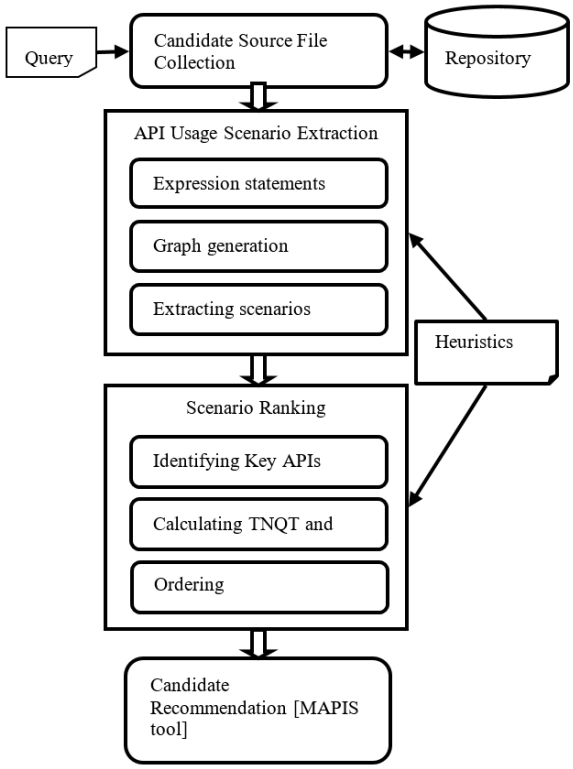


Fig. 1: Overview of the Proposed Approach

heuristics 2 give the hints that with the code structure type and identifier information helps to extract scenario with all necessary API elements. Ranking (Key API) heuristics are as follows:

- *Key API Heuristic 1: Object created and method call nodes those contains query terms are key APIs.*
- *Key API Heuristic 2: Object created and method call node those use the previously created objects and their methods as arguments are key APIs.*

Any of the present approaches do not identify the interrelationship of APIs within the scenario for ranking. We define the term ”key API” to represent the interrelationship among API elements. Our key API based ranking heuristics define the key API elements from a scenario by analysing the interrelationship among API elements.

B. Scenario Extraction Process

In scenario extraction, we consider structural relation and semantics (Type and Identifier information) of the source code. In our proposed approach, we consider multiple query terms. Based on those query terms, candidate source files are collected from

the already developed indexed open source software repositories. Hence, the input of our proposed system is a set of query terms and already developed candidate source files. The scenario extraction process consists of three key steps:

- Generating expression statement list from candidate source files
- Generating graphs (Directed Acyclic Graph and Undirected Graph)
- Extracting scenario

Consider the SqlConnection class in Introduction section. The expression statements from the field declaration and initialization statements are:

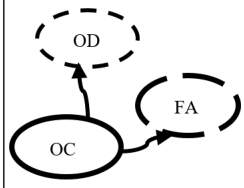
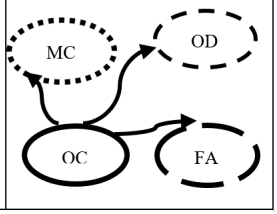
```
private String driverPath =
    "com.mysql.jdbc.Driver";
private String url =
    "jdbc:mysql://localhost:3306/localbd";
private String username = "root";
private String password = "";
```

In Object oriented programming, every member methods have access to the member fields of the class. That is why we merge those expression statements with each member method of the class. After analyzing the different types of statements of thegetMysqlConnection() method, we retrieve the following list of expression statements:

```
1. private String driverPath =
    "com.mysql.jdbc.Driver";
2. private String url =
    "jdbc:mysql://localhost:3306/localbd";
3. private String username = "root";
4. private String password = "";
5. Class.forName(driverPath);
6. Connection con =
    DriverManager.getConnection(url,
    username, password);
```

We repeat the expression statement extraction procedure for all other methods for a given class. In graph generation stage, we construct a Directed Acyclic DAG (DAG) to represent the structural relation among the expression statements. DAG is used to identify key APIs from extracted scenario. An undirected version of the DAG is used to extract all dependent API elements for a particular scenario. According to scenario heuristic 1 and heuristic 2, we defined the following key components, which are basically used in API usage. Those components represent four types of nodes of the DAG.

TABLE 1: DAG representation of source code

Expression Statements	Node Description	DAG representation
1. File f;	OD(File, f):1	
1. File f; 2. f = new File("data.txt")	OD(File, f):1 OC(File, f) : 2	
1. File f; 2. f = new File("data.txt") 3. f.read();	OD(File, f): 1 OC(File, f) : 2 MC(f, read): 3	

- Object Creation node: OC (Type, object): statement no.
- Object Declaration node: OD (Type, object): statement no.
- Method Call node: MC (Type, method name): statement no.
- Field Access node: FA (Type, object): statement no.

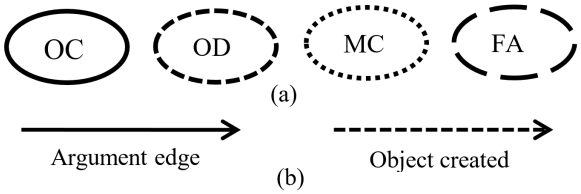


Fig. 2: Different Types of (a) nodes and (b) edges of the DAG

The edges of the DAG represent the following relation between two vertices:

- Argument
- Object created

Details of the DAG generation with different types of nodes and edges are shown in Table 1.

In scenario extraction step, query terms and undirected version of the DAG are used as inputs. Based on the query terms, our scenario extraction algorithm identifies the source nodes from the undirected graph that comprise query terms. Then, for each source node Breadth First Search (BFS) algorithm is applied to

collect all of the connected nodes. In this process a set of related statements are generated. Therefore, statements collected from different components of the graph are merged to construct scenario with heterogeneous API elements. Finally, after sorting and removing the duplicate statements, scenario extraction algorithm returns a scenario with all necessary API elements. The pseudo code of the algorithm is shown in Algorithm 1.

Algorithm 1 Scenario Extraction Algorithm

```
Input:query, graph/* query represents the search terms and DAG
represents the expression statement list
Output:scenario /* list of recommended API usage statements */
  scenarioStmList:= scenario.getStmList() := ϕ;
  begin
  Var nodeList := graph.getNodeList()
  Var termList := query.getTermList()
  VarsourceNodeList := getSourceNodeList( nodeList, termList)
  Foreach sourceNode ∈ sourceNodeList do
        Var visitedNodeList := BFS(graph, sourceNode)
        Var visitedStmList  := getVisitedStmList(visitedNodeList)
        scenarioStmList := scenarioStmList∪getDistinctStmList(
        visitedStmList)
  scenarioStmList:= sort(scenarioStmList)
  return scenario
  function : getSourceNodeList( nodeList, termList)
  begin
  Var sourceNodeList : = ϕ;
  Foreach term ∈termList do
        Foreach node ∈nodeList do
           If term⊂node.toString() then
                    sourceNodeList := sourceNodeList∪{node}
  Return sourceNodeList
```

For the expression statement list generated for getMysqlConnection()method earlier in this section, the undirected version of the DAG produced is shown in Figure 3(a). The highlighted nodes are the source nodes those contain the query terms. After applying BFS algorithm for each source node we get two lists of statements for the two different components of the undirected graph. Figure 3(b) highlights the nodes explored by BFS for query term 'Connection' and 'DriverManager'. Lists of statements extracted after applying BFS are: Statement list1 (for Connection): {2, 3, 4, 6, 6, 6, 6, 6} and Statement list 2 (for DriverManager) :{ 1, 5, 5}. According to the scenario extraction algorithm, we make the union of two extracted statement list and obtain the following final scenario statement list: Final Scenario: {1, 2, 3, 4, 5, 6} We formally present the pseudo code of the algorithm in Algorithm 1.

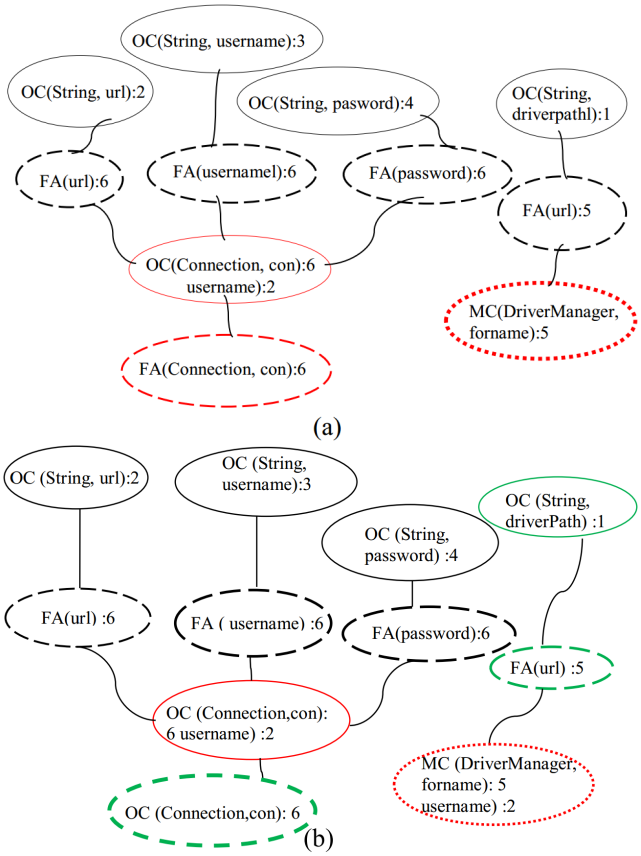


Fig. 3: Undirected graph highlighting (a) query terms and (b) scenario nodes

C. Ranking Subsystem

While the scenario extraction process returns a set of code snippets all of which that satisfy a given user query, the fit of these code snippets in solving a particular programming task may, however, vary. On average, a user can only be expected to scan the first ten or so code snippets returned by any search or mining process [19]. To best assist developers it is critical that scenarios with the potential to be best-fits be ranked within the first 10 or so results. In this section, we present a novel weight based ranking system. In the previous works [5], [16], most of the researchers have used shortest length (number of statements) and frequency of occurrence of a particular scenario in ranking. However, irrelevant code snippets may have shortest length and high frequency. To overcome that limitation, we introduced two key API based heuristics in ranking scenarios. Key API based heuristics are describes in the following section.

- *Key API Heuristic 1: Object created and method call nodes those contains query terms are key*

APIs. For example, consider the code snippet returned for query terms "Properties" and "Message" shown below. In the code snippet there are two object created nodes those contains query terms. So, according to our query heuristic 1, key API is new Properties() and new MimeMessage(session).

```
1. Properties properties = new
   Properties();
2. properties.put(mail.smtp.host,
   smtp.some-domain.com);
3. Session session =
   Session.getDefaultInstance
   (properties, null);
4. Message message = new
   MimeMessage(session);
5. Transport.send(message);
 password);
```

- *Key API Heuristic 2: Object created and method call node those use the previously created objects and their methods as arguments are key APIs.* For example, code snippet shown above construct an instance of Properties type and used that to construct an instance of Session type by calling getDefaultInstance() method. So, according to key API heuristic 2, getDefaultInstance() method of Session type is a key API.

Similarly, send method of TransportAPI is also a key API since it used the previously created Message instance as argument. We use the steps given in Table 2, to rank a set of recommendation based on key APIs. Ranking among three example scenarios are shown in Table 3. We can see that our ranking heuristics are able to suggest most useful scenarios in top recommendations. The first scenario is the best since it shows how to set the properties needed to send an email programmatically. As well as, it also shows the steps needed to send and a message object via the static send method of Transport class.

IV. Evaluation

We implemented our algorithms and developed the client tool named MAPIS [27]. The MAPIS tool was deployed on a standalone PC Pentium IV 2.8 GHz with 2 GB RAM running Microsoft Windows 7 and MySQL database. To compare our proposed approach with existing system, we have chosen IDENTIFIRE [14] which is the most recent work in this field.

TABLE 2: Ranking Steps

Step1	For each scenario calculate the total number of key APIs (TNKA) according to key API heuristic 2.
Step 2	For each scenario calculate the total number of distinct query terms (TNQT) found in key API list according to heuristic 1.
Step 3	Remove the total number of stop APIs (Primitive Types) from TNKA and TNQT
Step 4	Ordering: i. Order the scenarios in descending order of TNQT ii. Order among the scenarios with equal TNQT in ascending order of TNKA.

TABLE 3: Ranking amoung three scenarios for query terms "Properties" and "Message".

Scenario	TNQT	TNKA	RANK
`Properties properties = new Properties(); properties.put("mail.smtp.host", "smtp.some-domain.com"); properties.put("mail.smtp.port", "25"); Session session = Session.getDefaultInstance(properties, null); Message message = new MimeMessage(session); message.setFrom(new InternetAddress(from)); message.setRecipient(Message.RecipientType.TO, new InternetAddress(to)); message.setSubject(subject); message.setText(text); Transport.send(message);`	2	7	1
`Properties properties = new Properties(); FileInputStreamfis = new FileInputStream("configuration.xml"); properties.loadFromXML(fis);`	1	1	2
`JtextComponent message; Jcomponent parent; message = new JtextArea(" "); message.setEditable(false); message.setBackground(parent.getBackground());`	1	2	3

Though the source code of that work cannot be obtained, we carefully implemented their approach and compared performance with our proposed system. For two reasons, we have selected that paper. First, IDENTIFIRE [14] deals with multiple query terms. Second, it suggests methods based on the current structural context of the developer.

We have performed both quantitative and qualitative evaluation of our system. For quantitative result we present precision-recall measurements of our proposed system and for qualitative evaluation we present a comprehensive user study. In the user study we con-

sidered KODERS [2] tool also as it is frequently used by the programmers for code recommendation.

A. Building the repository

In order to create a test bed, we build a local repository and indexed eight open source java project from sourgefourge [13] which contains around 3,000 java files. We have stored type, objects and methods name against each java source file in MySQL relational database. The projects were chosen from different areas including programming environment, database, desktop application, text editor, web services, and etc. The statistics about those projects are shown in Table 4.

TABLE 4: Statistics of the indexed open source Java projects.

Projects Name	Description	Java Files	LOC
JabRef-2.7.2	Mind Mapper	598	109,373
hsqldb-2.2.8	Relational Database Engine	528	144,394
freemind-0.9.0	BibTeX Reference Manager	444	71,133
Jedit-4.5.1	Text Editor	554	176,672
pooka	A Java email client	269	70,543
tftp4java-0.8	FTP Server	39	10,359

B. Selecting sample tasks and query

To analyze the usefulness and accuracy of our proposed system for providing relevant API usage scenarios to a developer we designed ten different programming tasks (Table 5). The task was chosen to cover a wide range of applications including socket programming, desktop environment, database programming, server programming, DAG theory, reporting, and compiler design. The related query for each task is also defined. For each query candidate java files were retrieved from local repository.

C. Precision and recall based evaluation of retrieved scenarios

The performance of information retrieval systems is often evaluated in terms of recall and precision. Precision is defined as the number of relevant materials retrieved by a search divided by the total number of materials retrieved by that search. Recall is defined as the number of relevant materials retrieved by a search divided by the total number of existing relevant materials which should have been retrieved. In our contexts, precision and recall are defined in terms of a set of retrieved API usage scenarios (e.g., the list of API usage scenarios recommended by MAPIS for a query) and a set of relevant API usage scenarios(e.g., the list of API usages scenarios found in local repository by manual inspection for a particular programming task). More formally, we define precision and recall as

$$Precision = \frac{|\{relavent\ API\ usage\ scenarios\} \cap \{retrieved\ API\ usage\ scenarios\}|}{|\{retrieved\ API\ scenarios\}|}$$

$$Recall = \frac{|\{relavent\ API\ usage\ scenarios\} \cap \{retrieved\ API\ usage\ scenarios\}|}{|\{relavent\ API\ usage\ scenarios\}|}$$

Figure 4 and 5 shows the comparison of average precision recall plotting for 5, 10, 15, 20, 25, and 30 scenarios between our approach (we name it as MAPIS) and IDENTIFIRE [14]. From the result we can see Performance of MAPIS is better compare to IDENTIFIRE and MAPIS is able to recommend relevant scenarios in top recommendation. An API

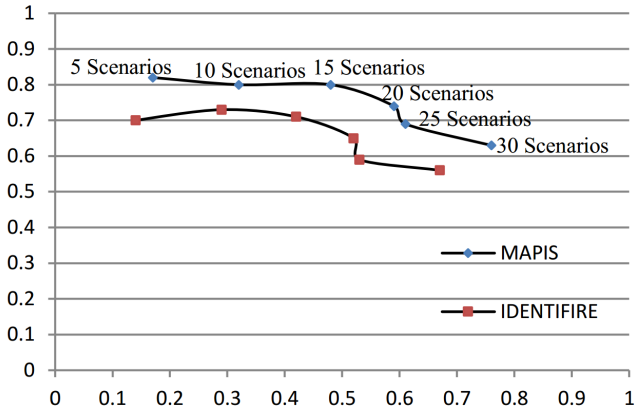


Fig. 4: Average precision recall plottings of 10 tasks.

usage scenario with lots of matching keywords may not be useful to the developer, if it does not contain proper object creation and method call sequences. Potential scenarios have all dependent object creation and method call sequences. Moreover, developers also like the code snippet that implements a scenario with few statements. Considering those facts, we identify the key APIs (key API heuristic 1) from the scenario and use it in ranking. Most of the developers have used shortest length and frequency of occurrence of a scenario as ranking parameters. We calculate the precision of four programming task (designed for user study) using shortest length heuristics and compared it to the precision of key API heuristic. Figure 6 shows the result of considering different types of ranking heuristic is better compare to shortest length heuristic. In our scenario extraction algorithm we considered

both syntax and semantics of client source code. Most of the works only consider either type information or identifier information. Hence, only object creation scenarios are recommended by the existing approaches. As we consider both identifiers and types, our approach retrieves scenarios with all dependent object creation and method call statements.

An API usage scenario with lots of matching keywords may not be useful to the developer, if it does not contain proper object creation and method call sequences. Potential scenarios have all dependent object creation and method call sequences. Moreover, developers also like the code snippet that implements a scenario with few statements. Considering those facts, we identify the key APIs (key API heuristic 1) from the scenario and use it in ranking. Most of the developers have used shortest length and frequency of occurrence of a scenario as ranking parameters. We calculate the precision of four programming task (designed for user study) using shortest length heuristics and compared it to the precision of key API heuristic. Figure 6 shows the result of considering different types of ranking heuristic is better compare to shortest length heuristic. In our scenario extraction algorithm we considered both syntax and semantics of client source code. Most of the works only consider either type information or identifier information. Hence, only object creation scenarios are recommended by the existing approaches. As we consider both identifiers and types, our approach retrieves scenarios with all dependent object creation and method call statements.

D. User Study

To evaluate the usefulness of the proposed method, we designed and conducted a user testing using MAPIS for solving real world programming problems which involved reuse of existing APIs. We selected four programming problems from different domains and given to 10 users. Users participated in the study are the undergraduate computer science students. Each user has around two years Java programming experiences. Each user was allowed to use MAPIS for two programming problems. For the remaining two problems, users were instructed to use KODERS [2] code search engine for one problem and IDENTIFIRE [14] for the other. None of the users have previously used any of the tools used in the study. So, a brief introduction about each tool was given to the users.

TABLE 5: Characteristics of programming tasks.

	Task	**Project**	**Area**	**Query**
1.	How to read blob objects?	hsqldb-2.2.8	Database programming, JDBC	Blobblob;
2.	How to I read and write object over a server socket?	hsqldb-2.2.8	Socket programming	ServerSocket server; // read write object
3.	How do I change JTablemodel?	JEDIT	Java Swing	JTablemyTable; //setModel
4.	How do I programmatically compile Java class?	freemind-src-0.9.0	Compiler design	JavaCompiler compiler;
5.	How do I copy and paste data from clipboard?	tftp4java-0.8	System programming	Clipboard clipboard; //copy paste
6.	How to ping a host IP address?	Pooka	Network programming	InetAddress address; // ping host
7.	How do I convert an object to JSON?	tftp4java-0.8	Web programming	Gsongson; //convert object
8.	How do I send an email message?	Pooka	Server programming	Message message; // send email
9.	How do I apply Breath First Search in a Undirected DAG?	JabRef-2.7.2-src	DAG theory	UndirectedDAG< Node, Edge> g = new SimpleDAG<Node, Edge>(Edge.class); //breath first search
10.	How do I print report in PDF file using JasperReport?	JabRef-2.7.2-src	Java reporting	JasperReportjasperReport; //write pdf

Problem 1: In real world software development developer frequently needs to load properties from xml file. Properties API of JDK is used to implement the task. Users were asked to display all the properties available in the xml file. Problem 2: Developer sometime needs to create zip file for a folder. In Java zip file is created via ZipOutputStream and ZipEntry APIs. Users were asked to create a zip file for a given folder.

- Problem 1: In real world software development developer frequently needs to load properties from xml file. Properties API of JDK is used to

implement the task. Users were asked to display all the properties available in the xml file.

- Problem 2: Developer sometime needs to create zip file for a folder. In Java zip file is created via ZipOutputStream and ZipEntry APIs. Users were asked to create a zip file for a given folder.

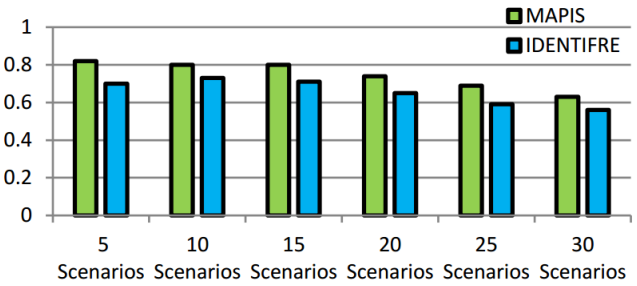


Fig. 5: Average precision plottings of 10 tasks.

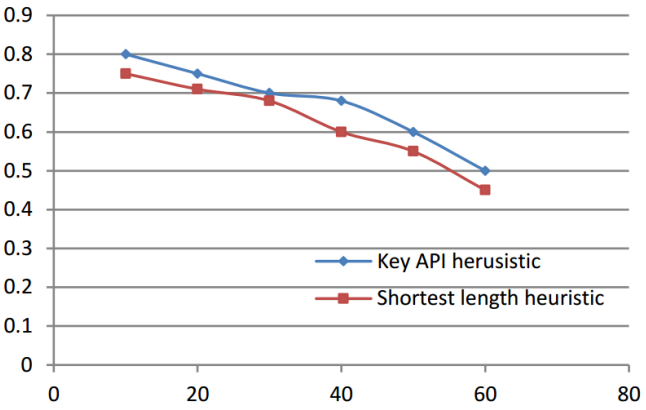


Fig. 6: Precision plotting's of key API and Shortest Length heuristic.

- Problem 3: In desktop application, file open and save dialog are commonly used. Users were asked to develop the following desktop application using swing library of JDK. The application will open a file chooser dialog when user clicks on the "open a file" button. Similarly, save file dialog will be opened when user click on the save file "save a file button". All the activities will be displayed in a text area.
- Problem 4: JDBC is the Java technology to connect to a remote database and run SQL queries on it. The user is given the URL of a remote database server. The task is to connect to the database using JDBC.

From the user testing results, we analyze the average task development time and percentage of users was able to complete each task using each tool.

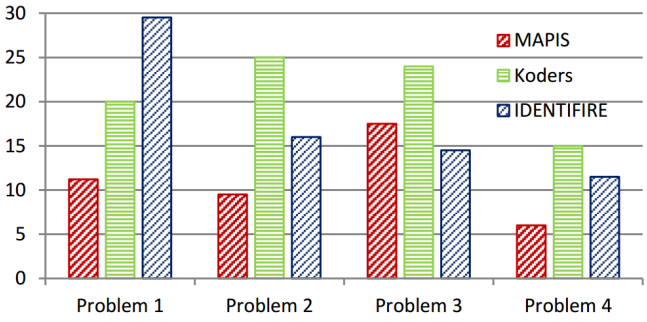


Fig. 7: Average Time Taken by the Users on the Problems using each Tool.

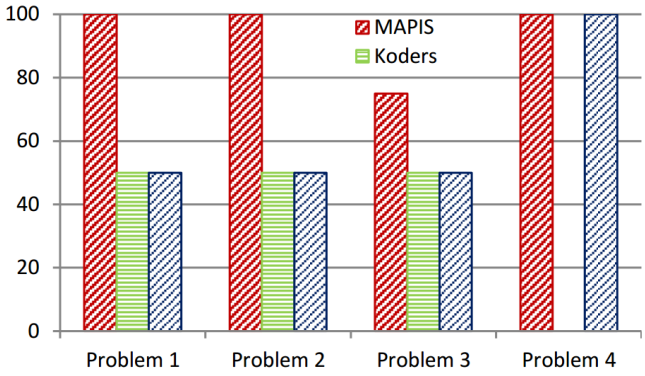


Fig. 8: Percentage (%) of users solved the problems using each tool.

First, Figure 7 illustrates the average time taken by the users on the problems using each tool. From the chart we can see that MAPIS took around 40% less time compared to KODER and IDENTIFIRE tools. Second, in Figure 8, we compare the percentage (%) of user was able to solved the problems using each tool. From the chart, we can see that 100% percent users were able to solve three tasks. However, average percentage of users completed the tasks using KODER [2] and IDENTIFIRE [14] is not more than 50% which indicates that our tool recommended useful API usage scenarios to the developers.

V. Conclusion

In this research work, we investigated how the developers can be provided useful API usage scenarios for a given task. The main difference of our approach with others is that we consider types, identifier and code structure in scenario extraction. At the same time, our key API based ranking algorithm provides best matching scenarios in the top as recommendation. In addition, our proposed system takes multiple query

terms and suggests a complete solution for a particular problem in a single hit. We presented precision-recall based evaluation to investigate the usefulness of the proposed system. We found that our scenario extraction and ranking algorithm is performing better compare to exiting approaches. From this we conclude that our proposed system is able to reduce the developers effort in finding relevant API usage scenarios, although further investigation is needed to determine the effectiveness of the retrieved scenarios. We have also performed a user study from which we got good response from software developers. MAPIS provides an easily extensible framework. So, in our future work we have planned to add more languages in MAPIS tool. In this research work, indexing algorithm was not considered. To improve the searching power, an indexing algorithm can be designed that help in getting more relevant API usage scenarios.

REFERENCES

[1] DeLine and Robert, *A field study of API learning obstacles*, Empirical Software Engineering, vol. 16, no. 6, pp. 703-732, 2011.

[2] Koders inc. [Online]. http://www.koders.com

[3] Krugle inc. [Online]. http://www.krugle.com

[4] Erik Linstead et al., *Sourcerer: mining and searching internet-scale software repositories*, Data Mining and Knowledge Discovery, vol. 18, no. 2, pp. 300-336, 2009.

[5] Hao Zhong, Tao Xie, Lu Zhang, Jian Pei, and Hong Mei, *MAPO: Mining and Recommending API Usage Patterns*, ECOOP 2009 Object-Oriented Programming, vol. 5653, pp. 318-343, 2009.

[6] Sheng-Kuei Hsu and Shi-Jen Lin, *MACs: MiningAPI-codesnippets for codereuse*, Expert Systems with Applications, vol. 38, no. 6, pp. 72917301, 2010.

[7] S. Chatterjee, S. Juvekar, and K. Sen, *SNIFF: A Search Engine for Java Using Free-Form Queries*, Funda-mental Approaches to Software Engineering.: Springer Berlin / Heidelberg, 2009, ch. 5503, pp. 385-400.

[8] Reid Holmes, Robert J. Walker, and Gai C. Murphy, *Strathcona example recommendation tool*, 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering, 2005, pp. 237 240.

[9] P. Steven Reiss, *Semantics-based code search*, in Proceedings of the 31st International Conference on Software, 2009, pp. 243-253.

[10] Juanjuan Jiang and Johannes Koskinen, *Constructing Usage Scenarios for API Redocumentation*, in 15th IEEE International Conference on Program Comprehension, 2007, pp. 259-264.

[11] M. Salah, et. al., *Scenariographer: A tool for reverse engineering class usage scenarios from method invocation sequences*, in In ICSM.: IEEE Computer Society, 2005, pp. 155-16.

[12] E. Duala-Ekoko, M. Robillard, and M. Mezini, *Using Structure-Based Recommendations to Facilitate Discoverability in APIs*, ECOOP 2011 - Object-Oriented Programming, vol. 6813, pp. 79-104, 2011.

[13] Source Fourge Org. [Online]. www.sourcefourge.org

[14] L. Heinemann, et. al., *Identifier-Based Context- Dependent*, 16th European Conference on Software Maintenance and Reengineering, 2012, pp. 31-40.

[15] Suresh Thummalapenta and Tao Xie, *Parseweb: a programmer assistant for reusing open source code on the web*, in Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, 2007, pp. 204-213.

[16] Naiyana Tansalarak and Kajal Claypool, *Xsnippet: Mining For Sample Code*, in 21st annual ACM SIGPLAN conference on Object-oriented programming systems, 2006, pp. 413-430.

[17] Marcel Bruch, Martin Monperrus, and Mira Mezini, *Learning from examples to improve code completion systems*, in Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, 2009, pp. 213-222.

[18] S. Thummalapenta and Tao Xie, *SpotWeb: Detecting Framework Hotspots and Coldspots via Mining Open Source Code on the Web*, in Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering, 2008, pp. 327-336.

[19] Amir Michail, *CodeWeb: data mining library reuse patterns*, in Proceedings of the international conference on software, 2011, pp. 827828.

[20] Java Examples. [Online]. www.jexamples.com

[21] Java frequently asked questions. [Online]. http://www.javafaq.com/

[22] Lee Wei Mar, Ye-Chi Wu, and Hewijin Christine Jiau, *Recommending Proper API Code Examples for Documentation Purpose*, 18th Asia Pacific Software Engineering Conference (APSEC), 2011, pp. 331-338.

[23] Iman Keivanloo, Laleh Roostapour, Philipp Schugerl, and Juergen Rilling, *SE-CodeSearch: A scalable Semantic Web-based source code search infrastructure*, in Proceedings of the 2010 IEEE International Conference on Software Maintenance, 2010, pp. 1-5.

[24] Lee Wei Mar , Ye-Chi Wu , and H.C. Jiau, *Recommending Proper API Code Examples for Documentation Purpose*, in 18th Asia Pacific Software Engineering Conference (APSEC), 2011, pp. 331 338.

[25] A.J. Ko and Y. Riche, *The role of conceptual knowledge in API usability*, in IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 2011, pp. 173 176.

[26] Google inc. [Online]. www.google.com

[27] S. M. Shahnewaz. *A Scenario Based API Recommendation System Using Syntax and Semantics of Client Source Code*, M.Sc. thesis, CSE department, IUT, April, 2012.

S. M. Shahnewaz is a software development professional currently working in Calgary, Canada area since 2015. Earlier he did his M.Sc. from Islamic University of Technology (IUT) Gazipur, Dhaka in 2012 and later he completed his second M.Sc. with Software Engineering Specialization from University of Calgary, Canada. His main research interests are in the area of recommended system for software developers, software release readiness and planning and software product management.

Husne Ara Rubaiyeat has received her B.Sc. in Computer Science and Engineering from Rajshahi University of Engineerign and Technology (RUET), Bangladesh in 2004. She started her career as software programmer and then pursued a Masters degree in Biomedical Engineering from Kyung Hee University, South Korea. She completed her MS degree in 2010. She also completed a Post-Graduate Diploma in Technical Education (PGDTE) from Islamic University of Technology (IUT), Bangladesh in 2012. Currently she is serving as a lecturer of Computer Science in the Natural Science Group of National University, Bangladesh.

Hasan Mahmud has received his Bachelor degree in Computer Science and Information Technology (CIT) from Islamic University of Technology (IUT), Bangladesh in 2004. He did his Master of Science degree in Computer Science from University of Trento (UniTN), Italy in 2009. He had received University Guild Grant Scholarship for the two years (2007-2009) Masters study and also awarded with early degree scholarship. He has different research articles published in several international journals and conferences. From 2009 he is working as an Assistant Professor in the department of Computer Science and Engineering (CSE) of Islamic University of Technology (IUT), Bangladesh. He is now pursuing his PhD at IUT. His research interest focuses on HCI based software systems, Gesture based Interaction, Machine learning. He is the co-founder of SSL.

Md. Kamrul Hasan has received his PhD from Kyung Hee University, South Korea. Currently he is working as an Associate Professor of CSE Department in Islamic University of Technology (IUT), Gazipur, Bangladesh where he has been serving since 2004. Previously, He obtained a B.Sc. in CIT degree from IUT. He has long experience in software as a developer and consultant. His current research interest is in intelligent systems and AI, machine learning in HCI, software engineering, and social networking. Dr. Kamrul is the founding director of the Systems and Software Lab (SSL) in the CSE department of IUT.