Journal of Software Engineering

ISSN 1819-4311

Academic Journals Inc.

www.academicjournals.com

Journal of Software Engineering 7 (3): 86-105, 2013
ISSN 1819-4311 / DOI: 10.3923/jse.2013.86.105

A Novel Integrated Framework to Increase Software Quality by Mining Source Code

[1]Shaheen Khatoon, [2]Azhar Mahmood, [2]Guohui Li and [1]Jianfeng Xu
[1]School of Software, Nanchang University (NCU) Jiangxi, China
[2]School of Computer and Applied Technology, Huazhong University of Science and Technology (HUST) Wuhan, China

Corresponding Author: Shaheen Khatoon, School of Software, Nanchang University Jiangxi, China

ABSTRACT

Source code contain lot of structural features that embody latent information that if identified can help software engineers to develop quality software in least amount of time. For instance, many programming rules are hidden in set of function calls, variable usage, data accesses in functions, object interaction etc. that seldom exist outside the minds of developers. Violations of these rules may introduce bugs which are difficult to uncover, report to bug-tracking systems and fix unless the rules are explicitly documented and made available to the development team. In order to address this problem there is a need to apply strong analysis techniques on source code to find latent programming patterns that can be potentially useful for performing various software engineering tasks. This study demonstrates how data mining techniques can be applied on source code to improve software quality and productivity by proposing a framework. This new approach is able to find different programming patterns such as programming rules, variable correlation, code clones and frequent API usage patterns. Furthermore, efficient algorithms are proposed to automatically detect violation to the extracted rules. Proposed framework is validated by developing a prototype and evaluated on various projects of significant size and complexity. Results shows proposed technique greatly reduced time and cost of manually checking defects from source code by programmers.

Key words: Data mining, source code mining, programming patterns, programming rule, rule violation, copy-paste code, bug detection, API usage

INTRODUCTION

Software organizations produce large volumes of data in software development process. Such data refer as source code, change history, execution traces, bug reports and open source packages. Useful information can be extracted from these large volumes of data that can play an important role in improving software quality and productivity.

Data mining is the process of non-trivial extraction of implicit, previously unknown and potentially useful knowledge from huge amount of data (Agrawal and Srikant, 1994, 1995). In relation to software development large amount of data created as part of the software development process is available. Data mining techniques can be applied on Software Engineering (SE) data and use the results to improve processes and products. Source code is one of an important artifact of development process which can be mined for interested patterns. It is typically a structured entity and contains semantically rich programming constructs such as variables, functions, data structures

and program structures which indicate patterns. Various data mining applications in software engineering have employed source code to aid software maintenance, program comprehension and software components' analysis. Since, the primary goal of software development is to deliver high quality software in the least amount of time. To achieve this goal software engineers are looking for tools which automatically detect different type of bugs to deliver high quality software and want to reuse existing frameworks or libraries for rapid software development. One of solution to achieve quality objective is effective and efficient testing of system under development. However, common testing approaches focus typically on correctness of functionality and performance and do not ensure the absence of programming rule violations which if left undetected may silently compromise the results of computations. Furthermore, common testing approaches uses well documented specification to verify against correctness of functionality whereas programming patterns/rules are not documented. To address these issues in this study data mining techniques are applied on source code to discover informative relationships and trends through large amount of source code which can facilitate in identifying different programming patterns.

In order to reduce the development time developers often reuse the code. One of the solution developer's often use is copying and pasting (code clones) the existing code from the same application. However, many programming errors occur when programmers create and update copied cod. For instance (Li *et al*., 2004) identified that a some of bugs in Linux were introduced when a programmer copied code but failed to update identifiers in the pasted code. To address this issue there is a strong need of tool to identify similar code automatically by comparing the internal representation of source code. Proposed framework accepts a source code file and finds all segments of copied codes by applying mining algorithm. Once all the duplicated code segments are identified it fed to automatic system to find copy paste code related bugs.

Another solution for rapid software development is reusing software libraries, application framework and open source repositories. The programmer usually uses internet resources to search for useful APIs by using common search engine such as Google. However, search engine usually return the large amount of document most of which are not relevant. Even if some document are relevant but the programmer fade up after browsing through first few documents and go for another way. To address these challenges proposed framework provides a mechanism where programmer can find relevant .Net APIs by searching over internet and guides programmers toward the results that will be most helpful for their current task.

The main contributions of proposed framework are:

- The first practical frameworks to automatically identify multiple patterns. All the previous approaches identify one specific patterns such as function pairing (Engler *et al*., 2001) or one specific type of bug detection (Li and Zhou, 2005). In contrast proposed source code mining framework integrates multi features components under one umbrella
- Since, the traditional data mining algorithm are meant for domains like market-basket analysis, DNA sequence mining and so on. The source code domain is completely different to apply mining algorithms. In order to convert source code into data mining problem data transformation method is proposed called data generator
- In order to automatically extract various programming patterns, code miner is developed which uses the data mining techniques to discover valuable patterns from source code. The numbers of patterns generated by code miner are too many and all are not of user interest. Also the mining process in code miner has lack of user interaction. To improve the mining process and making it more users' focused constraint-based mining is proposed

- The proposed framework can identify various kinds of rule violation bugs. By evaluating framework on large software's, the result shows it can detect many violations to extracted programming rules

RELATED WORK

One major area in this direction is rule mining techniques (Engler *et al.*, 2001; Li and Zhou, 2005; Chang *et al.*, 2007; Lu *et al.*, 2007; Ramanathan *et al.*, 2007) which induces set of rules from source code of existing projects and anomalies are uncover by looking for violation of specific rule. For example, if A occurs then B and C happen X amount of the time. Here, Engler *et al.*, 2001 work and PR-Miner discover patterns involving pairs of methods calls and functions, variables, data types that frequently appear in same methods and do not contain control structures or conditions among them, also the order of method calls is not considered. However, compared with (Engler *et al.*, 2001) work which extracts only function-pair based rules, PR-Miner extracts substantially more rules by extracting rules about variable correlations. However, PR-Miner uses the gcc front end, hence coupled with specific compiler implementation. In contrast, in this study a light weight parser to analyze source code based on language standards is developed. Furthermore, the proposed approach can find additional patterns classes such as multi-variable access correlation, duplicate code and API usage. CHRONICLER developed by (Ramanathan *et al.*, 2007) is used to identify the precendance relationship in procedures. It applies path-sensitive static analysis to automatically infer accurate function precedence protocols. In recent work (Chang *et al.*, 2007) applies graph mining to mine implicit conditional rules and to detect neglected conditions. The user has to apply constraint on the context of rule being mined. However, the approach does not handle directed and multi graphs.

Recently, (Khatoon *et al.*, 2011a) surveyed approaches, including those using frequent-pattern mining, to mine source code for various software engineering tasks. Also in their initial work (Khatoon *et al.*, 2011b), proposed a framework for mining programming patterns from source code by applying on ERP solution. However, in this work framework is applied on large software systems and also efficient algorithms are proposed for detecting violation from mined patterns.

In area of clone code identification only two approaches CP-Miner (Li *et al.*, 2004) and Clone detection (Wahler *et al.*, 2004) are found which uses data mining to detect clones. CP-Miner uses frequent token sequence and flag bugs by identifying deviations in mined patterns. In this technique basic code blocks are transformed into sequence of number. The ColSpan algorithm is applied to the resulting sequence database to find basic copy-pasted segments. By identifying the abnormal mapping of identifiers, it detects copy-paste bugs.

Whereas, (Wahler *et al.*, 2004) approach finds clones at a more abstract level by converting the Abstract Syntax Tree (AST) to XML by using frequent item set-mining technique. This tool first converts source code into AST which contains complete information about source code by using parser. XML configuration file inputs to frequent itemset mining algorithm which finds frequent consecutive statements. However, this technique only identifies exact and parameterized clones at a more abstract level.

In direction of API usage patterns techniques (Michail, 2000) developed CodeWeb which describes how data mining can be used to discover library reuse patterns in existing applications. To use CodeWeb developer must find similar applications of interest in advance. Given an API sample, Strathcona (Holmes and Murphy, 2005), Prospector (Mandelin *et al.*, 2005), XSnippet (Sahavechaphan and Claypool, 2006), MAPO (Xie and Pei, 2006) and Parseweb

(Thummalapenta and Xie, 2007) provide example code of that API. Strathcona generates relevant solutions when the exact API is included in the search context. However, mostly programmer has no knowledge of which API has to be used for solving the query. Prospector and XSnippet are limited to the queries of a specific set of frameworks or libraries. Both use the code relevance to define the code context which best fits the required code. However, Strathcona only use the lexically visible types to define the code relevance whereas, XSnippet uses parents of the class under development as well as lexically visible types for a given method contained in class to define code relevance. The major problem with both of approaches is that they use the local repositories which are limited in relevant code examples.

MAPO uses a query to define the relevant API and gather the relevant code samples from open source repositories and conduct data mining. However, for using MAPO Programmers need to know the API to be used to identify usage patterns. PARSEWeb like MAPO takes queries of the form "source object type to destination object type" as an input and suggests what API method sequence should be used to obtain one object from another from potential solution. However, it only suggests the frequent sequences and code samples but cannot directly generate compile-able code.

PROPOSED SOURCE CODE MINING FRAMEWORK

The core idea of proposed framework is applying data mining techniques on source code to discover valuable patterns from large software and exploit such extracted patterns to improve software quality and productivity. Benefiting from frequent itemset and sequential mining proposed framework can identify the programming rules that contain multiple elements from source code such as functions and variables. The Framework also suggests the solution for current programming task by mining frequent usage patterns of related code sample which assist the programmer for rapid software development. To make mining process more effective and user focused constraint based mining is used. Constraints are applied on the form of rules to be mined. Hence user can explicitly specify the rule form in which he is interested. Instead of searching for all rules the mining time is reduce by focusing the mining process towards the constrained rules only. Additionally, efficient algorithms are proposed to detect various kinds of bugs including rule violating, variable access correlation violation and copy paste code related bugs.

The schematic workflow of proposed approach is shown in Fig. 1. It accepts source code file and source parser convert it into AST. A database is generated by traversing AST. By applying mining algorithms programming rules are generated which can be used by following ways:

- Instances of rules are provided to knowledge base or as an appendix to a developer's guide which is useful way to inform developer's about the system specific rules, so that they can use these rules in future development
- Instance of rules are use to assist in development of current task such as API usage patterns which in turn benefits for rapid software development
- The instance of cloned code are used to optimize code as well as help in identifying copy paste code related bugs
- Instance of the rules are fed to automatic violation detection component that can identify rule violation at specific location of code, hence programmer can fix that violation which assist in improvement of software quality

The main components of proposed framework are as follow.

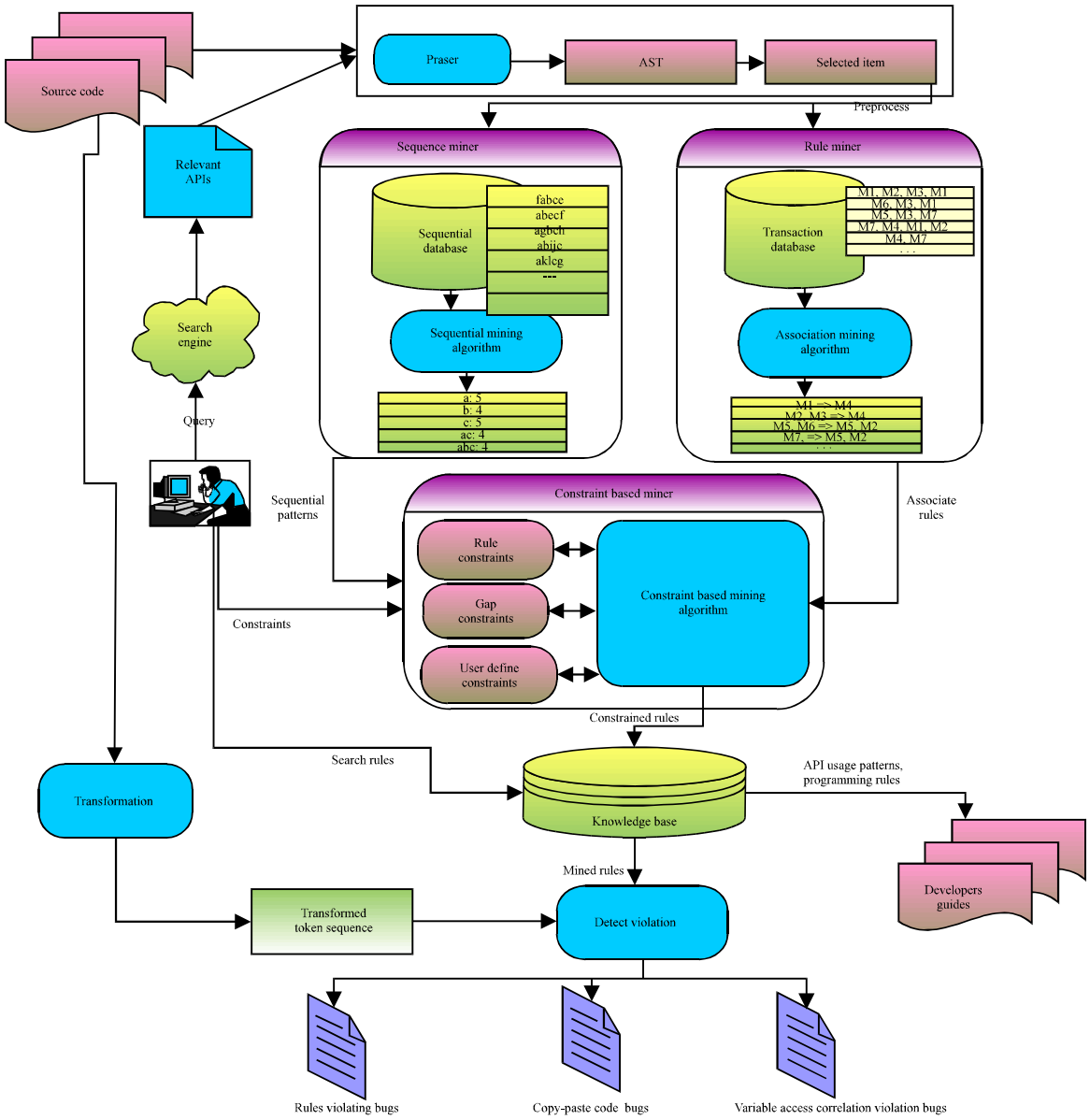

Fig. 1: Workflow of source code mining framework

Data generator: Algorithms for mining association rules and sequential patterns are traditionally used for market basket analysis. This work completely addresses a different application domain that of source code analysis thus a suitable data model is needed which format the source code with the requirements imposed by existing mining algorithms.

The task of data generator is transformation of source code into database suitable to apply mining algorithm. To transform source code into transaction database it is very important to define a code unit which is map into itemset. The code unit is defined in a way in which majority of code elements are contained. In proposed framework modules are used for defining basic code unit and treated as itemset, all elements of modules (function, procedure, classes) are mapped to items. Modules can be easily identified within a program as their start and end follow specific conventions.

Modules have also the advantage that it contains the significant amount of code that consists of several statements such as variable declarations and assignments and so on. Therefore, modules may contain a large number of attributes. Selecting the modules as basic code unit also facilitates in capturing the common attributes which enhance the quality of association rules and of consequent groups.

Parsing source code: The purpose of parsing source code is building transaction and sequential database to convert source code into data mining problem. A user gives source code in a given language to source parser which transforms source code into data structure called Abstract Syntax Tree (AST). Converting the source code into ASTs ensures the preservation of syntactic details of every programming construct since it contains all the information about parsed program. Each node of the AST represents the program element such as identifier name, data type name, control structure etc. The AST is than passes by subsequent phases and finally uses as the basis to construct the transaction and sequential databases suitable to apply mining algorithm. In next step each AST is traversed to select the items, such as identifier names, data type specifiers, operators and control structures. In this way, each function is converted to an itemset. The sample code shown in Fig. 2 used to generate AST. The corresponding AST generated by parser is shown in Fig. 3.

Generating transactions database: A database of system entities and their relationship is generated by traversing the function bodies in ASTs. Each node of the AST represents the program element such as identifier name, data type name, control structure etc. In order to convert a function into an itemset the AST of that function is traversed and each selected element is hashes into a number. The procedure for generating hashing values from AST is shown in Fig. 4. By hashing each element into a number entire function is converted into itemset which is written as a tuple in itemset database. Let f_1, f_2, f_3,...f_n is the number of function present in the software system. The itemset I is the union of all functions:

$$I = f_1 \cup f_2 \cup f_3 \cup ... \cup f_n$$

```
……..
Void f ()
{
if (a[0]>a[1]) {
t = a[0];
a[0] = a[1];
a[1] = t;
}
if (a[1] > a[2]) {
t = a[1];
a[1] = a[2];
a[2] = t;
 }
 max = a[2];
}
……..
```

Fig. 2: Sample code used to generate AST

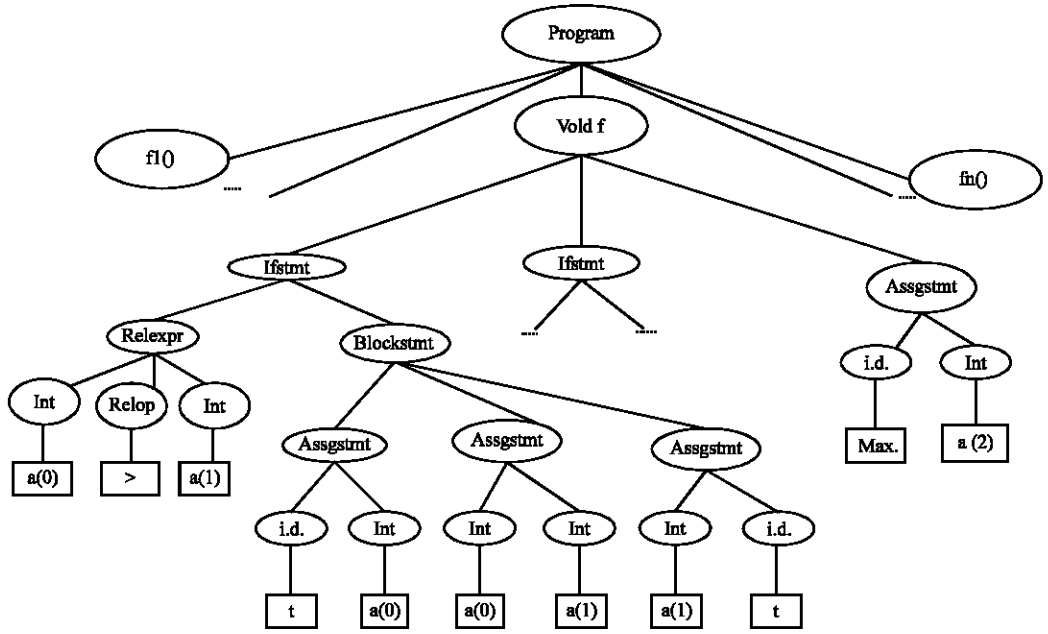

Fig. 3: Abstract syntax tree

The itemset of all function construct the transaction database which is provided as input to mining algorithm at next phase. Each transaction T corresponds to a subset of I consist of different types of function.

Generating sequential database: For copy paste code detection, the code fragments that have some semantic notion of sequencing involved such as sequences of declarations or statements are needs to identify. For this purpose program is broken into set of basic blocks. A basic block is the maximum length sequence of statements $S_1,...S_n$, where $n \leq 1$ having one entry point (S_1) and one exit point (S_n) and there is no branch statement except the last statement. For this purpose abstract syntax tree is traversed where each node represents a basic block and list of each basic block is computed. For each basic block statements are mapped to a number. Similar statements are mapped to the same number by mapping all identifiers of the same type into the same value, regardless of their actual names. The basic block is mapped into sequence of numbers by hashing all statements without any branch. As a result, a program is mapped into a database of many sequences.

Consider the AST as shown in Fig. 3, it contain five basic blocks for function Void f. The tree is traversed and each basic block is hashed to sequence of numbers. Here function HashPJW (Aho *et al.*, 2007) is used which accept pointer to string and return an integer. Corresponding to these basic blocks hash values are associated to each sequence as shown in Table 1.

The hash value from Table 1 is converted into following sequences where each sequence represent basic block:

(10624768)
(20142602, 88720163, 28375133)
(10863203)
(10043845, 70043621, 81438763)
(10845126)

```
Procedure: Generate Hash Values
Input: AST
Output: Hash values
Method:
Traverse( Node rootNode )
begin
 Stack stack = new Stack()
 Node node=rootNode.getFirstChild();
 while (node!=null)
  begin
   node.getNodeValue();
  if (node.hasChildNodes())
  begin
  if (node.getNextSibling()!=null)
  stack.push.ApplyHash( node.getNextSibling() );
  node = node.getFirstChild();
 end
else
    begin
     node = node.getNextSibling();
     if (node==null && !stack.isEmpty())
     node=(Node) stack.pop();
    end
  end
end

PUBLIC unsigned ApplyHash(unsigned char *Node)
begin
   unsigned    HashValue = 0;
   unsigned   Check;

    for (; *Node; Node++)
 begin
         HashValue = (HashValue << TWELVE_PERCENT) + * Node;
         if ( (g = HashValue & HIGH_BITS) != 0)
         HashValue =(HashValue^(Check>>SEVENTY_FIVE_PERCENT))&
~HIGH_BITS;
       end
return HashValue;
```

Fig. 4: Procedure to generate hash values from AST

Code miner: Source code mining framework parses the source code and generates the transaction and sequential database. Code miner applies the different mining techniques on these databases to uncover interesting patterns. Code miner consists of three major components each of which perform a specific operation: rule miner, sequence miner and constraint based miner. Below the detail of each component is provided.

Rule miner: It inputs a given set of transactions defined in transaction database and finds all the frequently occurring subsets of items that have at least a users specified minimum support (Agrawal and Srikant, 1994). It generates a number of association rules by applying Association Rule Mining algorithm Apriori. A sub-itemset (a subset of an itemset) is considered frequent if the

Table 1: Hash values assigned to each statement of function Void f

Statement	Hash value
if (a[0]>a[1])	10624768
t = a[0];	20142602
a[0] = a[1];	88720163
a[1] = t;	28375133
if (a[1] > a[2])	10863203
t = a[1];	10043845
a[1] = a[2];	70043621
a[2] = t;	81438763
max = a[2];	10845126

number of its occurrences in the database (denoted as its support) is greater than or equal to a specified threshold. Items in a frequent pattern are likely to have some correlation in between. The set of corresponding frequent program elements discovered by mining algorithm are called programming patterns which indicates that the program elements are correlated and frequently used together. For example, let I be the set of all items present in database. Association rule mining search for the power set of I for each patterns classes satisfying minimum threshold:

$$I= \{\{A, B, C, D, E\}, \{A, B, D, E, F\}, \{A, B, D, G\}, \{A, C, H, I\}\}$$

The support of sub-itemset {A, B, D} is 3 and its supporting itemsets are {A, B, C, D, E}, {A, B, D, E, F} and {A, B, D, G}, If min support is specified as 3, the frequent sub-itemsets for I are {A}:4, {B}:3, {D}:3, {A, B}:3, {A, D}:3, {B, D}:3 and {A, B, D}:3, where the numbers are the supports of the corresponding sub-itemsets. The location (where the items locate in the source code) of each frequent itemsets is also recorded which is required when evaluating if a violation is detected.

Once all patterns Z are identified, all rules are generated by splitting Z in head X and body Y such that: $X \cup Y = Z$ and $X \rightarrow Y$ with support s and confidence c satisfy the rule. For each pattern class set of rules are generated.

To describe such mining problem, consider a set of items such as $I = \{i_1, i_2, i_3, i_n\}$. A set of transactions T_x in dataset D can be described as a unique id given to each transaction in D such that $T_x \subseteq D$. From D, association rules are mined. These association rules can be expressed $A \rightarrow bl$ where, A, $B \subseteq I$ and $A \cap B = \phi$. In this implication A is called antecedent and B is calle the consequent of an association rule. If N denotes the number of transactions in a dataset then the interesting association rule can be measured with support and confidence of rule as given in equation:

$$\text{Support } S(A \rightarrow B) = \frac{A \cup B}{N}$$

$$\text{Confidence } S\ (A \rightarrow B) = \frac{S(A \cup B)}{S(A)}$$

With these equations, support of a rule is the significance of correlation while confidence of a rule measures the correlation degree between the set of items. Based on these measures, a rule is interesting which satisfies the minimum support and minimum confidence criteria specified by the user.

Consider a simple example of 6 items such as I = {A, C, D, E, F, G} with transactions T_x = {ABCD, ABC, ABD, ACD, ABC, BD} and $D_= \{t_1, t_2, t_3, \ldots, t_x\}$. Let the minimum support be 2,

(20%) and rules mined from D composed of T_x such that $\forall T_X \subseteq D$ with unique transaction id in D. Therefore, association rules that can be mined from a given D are A→B (66.67, 80%) A→C (66.67, 80%) A→D (50, 60%), B→C (50, 60%), B→D (50, 60%), C→D (33.33, 50%), A, B→C (50, 75%), A, B→D (33.33, 50%), A C→D (33.33, 50%).

Sequence miner: Sequence miner inputs the set of sequential database and applies the sequential-pattern mining algorithm to finds all the frequently occurring subsequences that have at least a user-specified minimum support (Agrawal and Srikant, 1995). To detect copy paste code segments proposed framework first convert the given source file into sequence frequent mining problem. In this step mining framework applies Prefixscan (Pei *et al*., 2001) algorithm on sequence database to find copy paste code segment. The reason to choose Prefixscan is that it does not compute every possible combination for potential candidate sequence, hence increase the efficiency of framework by avoiding unnecessary and redundant database scanning. These frequent subsequences are exactly copy-pasted segments in the original program. From Table 2 the framework identifies sub-itemset {10624768, 20142602, 88720163, 28375133} as frequent subsequence as it appears twice in sequential database. The corresponding code statements from lines 123-126 and 152-155 are copy-paste code segment.

The proposed framework is efficient in a sense that it can identify the modification in copied segment. This is done by mapping the similar elements such as function, variables and types into same value regardless of their name. As shown in Table 2 statements from line 123-126 and 152-155 are mapped to similar hashing values although their values are not matching.

Extracting API usage patterns: In order to facilitate the developer to search for specific code example relevant to current task, the proposed framework recommends API code snippets to the programmers by loading and analyzing the relevant code examples from web. It forms a database

Table 2: Example of assigning values to statements

Statement	Value
110. Void f (int)	16345621
111. {	…..
…… 10624768	
123. if (a[0] > a[1])	20142602
124. t = a[0];	88720163
125. a[0] = a[1];	28375133
126. a[1] = t;	….
….	
}	
….	
{	
152. if (a[1] > a[2])	
153. t = a[1];	10624768
154. a[1] = a[2];	20142602
155. a[2] = t;	88720163
156. max = a[2];	28375133
… 10845126	
}	
…	

of frequent API usage patterns mined from relevant source files in a code search engine. The user provides query statement by specifying one or more line of code in search context. The framework automatically retrieves relevant source code files (relevant APIs) from web or from previous projects. This process consists of two phases: Forming the API patterns database and making recommendation to programmer. At first stage it transforms the retrieved API statement in the form suitable to apply mining algorithm. This is done by constructing the AST of top 10 results set. By traversing the AST a transaction database is build. The Apriori algorithm computes all the patterns and constructs the pattern database. In second stage developers searches the specific usage pattern from pattern database for a given task. This is done by considering the rule antecedent as source object and rule consequent as destination object. The tool only mines the patterns matches the given rule template. The mined patterns results in more than one possible solution for desired API usage. To assist programmer to identifying desired usage patterns quickly the mined results are ranked by corresponding support value. The results having higher support value are shown on top.

Constrained miner: In proposed framework a user may place various constraints under which mining is to be performed. These includes interestingness constraints which involves specifying thresholds on measures of interestingness or rule constraints which place restrictions on the number of predicates that exist among them. To illustrate the concept of constraint based rule mining consider set of transactions shown in Table 3. Here it is desire to identify customer characteristics with sales of speakers. The rule for the extraction of such information can be of the form P(X,Y)→buys(X, "speaker") where variable X represents a customer and variable Y takes on values of the attribute assigned to the predicate variable P. During the mining process only the rules which match given rule are found. One example of a matching rule is age(X, "young")→buys (X, "speaker"). Since, both the predicate variable P and variable Y may vary, rules gender (X, "male")→buys(X, "speaker") and age(X, "old")→buys(X, "speaker") also satisfy the given rule.

In source code mining framework the constraint based architecture is proposed for supporting human centered and exploratory discovery of knowledge. The constrained association queries are uses as a means of specifying the constraints to be satisfied by the antecedent and consequent of a mined association. User specifies the constrained query imposed on the antecedent and consequent of the rule to be mined as shown in Fig. 5. In phase *I* user specify the constrained query and threshold, the algorithm finds the intended candidates for the antecedent and consequent of the association. The output of this phase is list of antecedent and consequent satisfying the given threshold. User inspects the resultant candidates and can refine the constraint or threshold. User can repeat through this phase as many times as required. If the user is satisfied with current candidate list it will move to phase 2. In this phase user instruct the system to find association among the selected candidates by setting the confidence measure. The user can select the specific

Table 3: Transaction set showing sales of item

Transaction i.d.	Items
T_1	CD, memory disk
T_2	Speaker, CD
T_3	CD
T_4	CD, memory disk, speaker
T_5	Memory disk, speaker, microphone

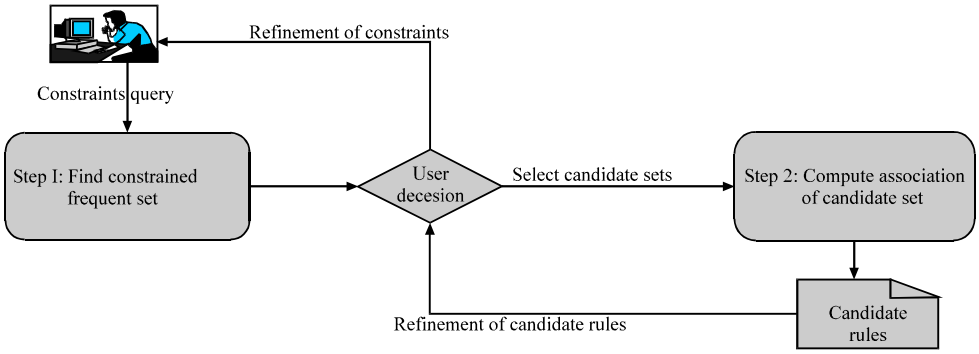

Fig. 5: Workflow of Constraint Miner

rules or small set of candidate rules by setting the properties of data. Also user can refine the selected candidate by increasing the confidence measure to make the mining process more focused. Finally, the output of phase 2 consists of all associations that satisfy the constraint conditions specified at the beginning of phase 2. The result are displayed in GUI, after examining the generated rules user can change any parameter depending which parameter want to reset. Parameter resetting may trigger the phase 1 or 2 computation.

Violation detector

Rules violation detection: Locating source code fragment that match certain patterns and violation of identified patterns are critical for programmer who writes code, testers who involves in testing activities and maintainer who involved in understanding code for re-engineering tasks. In order to detect violation to extracted rules identified in previous steps, an efficient method is proposed which automatically finds the locations where given rule is not followed. Rules violation algorithm is shown in Fig. 6. The algorithm can apply on both function used together as well as variable access correlation violation. User specified the rule to be checked in given source file. The algorithm transforms the source file into tokens and performs token by token comparison to identify the code location where given rule violates. Once complete source file is checked for violation, a bug report is generated contains the set of rules and locations where the given rules are not followed. Programmer can check the specified location and inspect or fix the given violation.

Detecting copy-paste code related bugs: In order to automatically detect the defects introduced by copying and pasting the code a naive method is developed as shown in Fig. 7. The copied code segment identified in mining process is fed into it and it automatically flags the code location for potential bugs. It based on tokenization to find the inconsistencies in source code based on given code segment. A clone is 100% consistent if it exactly map to other copied segment. For a segment in which statement insertion and deletion are carried out the similarity ratio can identify the percentage of inconsistence between original clone and copied clone. To measure the amount of similarity between original code and copied codes a similarity ratio is computed:

$$\text{Similarity ratio} = \frac{\text{Similar No.}}{\text{Total No.}}$$

Here, similar number shows the total number of occurrences where the given token is unchanged and total number is number of token in given clones. Observation shows 100%

Procedure: Rule violation bugs detection
Input:
SOURCE: Source code file
Set of Rules R_n where n = 1,2,3,...n
Output:
P: Source code locations where rule violated
RV: set of rules violation
Method:
1. Generate Tokens T←Read (SF)
// T contain set of token read from source file
2. Call Exclude Token Class
//This class is called to exclude keywords, all programming keywords are related
to syntax, complier can identify syntax error if any
3. For (k= R_i to R_n)
4. While not end (T)
5. COMPARE R_i with $T_{i=1....n}$
6. IF (T_i= R_i) // Token Found
7. Return Result [Position, R_i] //Return True and Store source code position and rule element
8. Next
9. Traverse Result [P_iR_i ..... P_nR_n] //Order by Position Found
10. For Result[i] to Result[n]
11. IF DIFF (R_i, R_{i+1})!=-1 or Diff (R_i, R_{i+1})!= RuleLength-1
// Rule Length is No. of elements found in provided rule
12. Return P_{ith} source line and R_i
13. //Rule violation identified
14. Next
15. Print [$P_{i....n}$][$R_{i....n}$]

Fig. 6: Procedure to detect rule violating bugs

similarity mean all the given clone is mapped to copied segment. If similarity measure is less than 100% and greater than 50% it shows the probability of bug at unchanged location. Proposed framework flag this location and report copied code instance and its mapping information to help programmer to identify cause.

EXPERIMENTAL EVALUATION

In order to validate the proposed framework a working prototype is built. Three software applications: PRWHP (Promotion of Rain Water Harvesting in Pakistan), Metro HRM and HCC Bank, developed in local software house are used to evaluate the proposed framework. The specific system detail including number of files, number of function and number of external interfaces are shown in Table 4. PRWHP is web based application developed to record the usage of rain water throughout the country. Metro HRM is installed on Metro fast train stations to manage human resource operation; it includes HR induction, HR plans and its monitoring, resource allocation, individual resources promotion and packages. System also maintain payroll, leaves records, shift change on each metro station. HCC Bank manages account details of individual and group, these details includes all cash inflows and outflows. System is capable to handle all transactions and major commercial banking operations. Table 5 shows the list of the hardware and software used for developing the prototype.

Procedure: Copy-paste bugs detection
Input:
SOURCE: Source code file
CC: Clone code
Output:
CPB: Copy paste code related bugs
Method:
1. Generate Tokens of Clone Code CT←Read (CC)
   // CT contains set of token of given clone code based on unique identifier
2. Total Clone Token←TCT
   //Total number of Clone token generated
3. Generate Tokens ST←Read (SF)
   // ST contain set of token read from source file based on same unique identifier
4. While not end (ST)
5. Read ($ST_{i....n}$)
6. Match CT_i with $ST_{i=1....n}$
7. IF ($CT_i = ST_i$)
8. Int Found, Position=0
9. Found=Found+1
10. Position=ST_i
    //Store the source code position where first token found
11. For (CT_{i+1} to TCT)
    // Match next clone token with Source token
12. Read ST=ST_{i+1}
13. IF ($CT_{i+1} = ST_i$) // Token Found
14. Found=Found+1
15. Next
16. IF (Found>TCT/2)
17. Return Source Code line
18. \\ Probability of potential bug
19. Print bug location←Position
20. Read Next Source Token ST_{i+1}
21. End

Fig. 7: Procedure to detect copy paste related bugs

Table 4: Systems used for evaluating the proposed framework

Systems evaluated	No. of files	No. of functions	No. of external interfaces	Application type	Database
PRWHP	85	1,820	12	Web	MS SQL 2005
Metro HRM Solution	789	1335	10	Desktop	Oracle 10 g
HCC Bank	1279	4220	9	Web	Oracle 10g

Table 5: Software and hardware support detail

Operating system	Windows sever 2003
Programming Language	C# (MS Visual Studio 2005)
Database	MS SQL Server 2005
PC hardware specifications	Pentium Intel 2.2 GHz Core2 Duo/2GB Ram

RESULTS AND DISCUSSION

The prototype system takes three parameters along with source code file to be checked. The parameter includes: min support, the confidence threshold and gap constraint. The support and confidence of extracted rules is user defined. In this case support of each rule is 10, confidence 90% and gap constraint is 2.

Extracting programming rules: Table 6 shows total numbers of rules identified by running the prototype using systems under evaluation. These rules include function to function as well as multi variables access correlations rules. The rules having confidence lower than 90% are automatically pruned. Since, the results of frequent-pattern mining are sensitive to the externally supplied minimum-support value, multiple run were performed with different minimum-support values and a minimum confidence.

In order to demonstrate how to choose support and confidence values in proposed framework, sensitivity analysis is performed on minimum support and minimum confidence values Fig. 8 and 9 shows the support and confidence distribution of rules extracted by prototype. It can be observed from figures, the numbers of rules are decreased by increasing the corresponding support or confidence value. Rules are ranked with increasing support value, since rule with larger support are trusted more and can increase the program quality.

The length of the rule can give an overall idea of pattern complexity which is the measure of total number of programming elements such as function, variable, data type etc. in certain rule. Fig. 10 shows the length of rules in systems under evaluation. It can be observed that more than 60% rules contain 2-4 elements whereas less than 10% rules contain 20 or more elements.

The distribution of the variables with different numbers of correlated peers is identified in order to capture the variable correlation. The results of PRWH, HRM solution and HCC bank applications shows that most variables are only correlated with a small number of peers as shown in Fig.11, it can be observed from figure that around half of the variables are correlated with only one

Table 6: No. of rules extracted by integrated system

System evaluated	No. of rules identified
PRWHP	741
Metro HRM solution	988
HCC bank project	1202

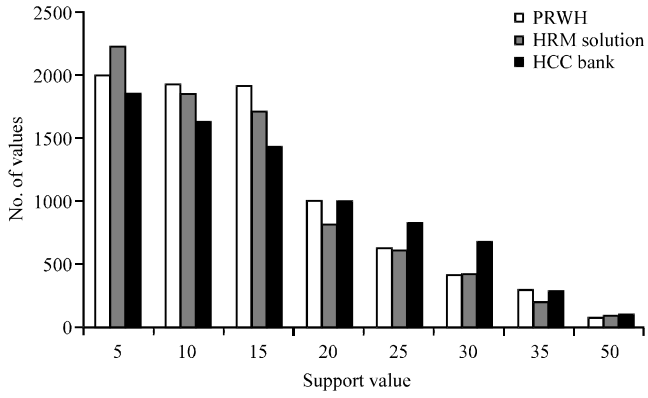

Fig. 8: Support distribution of rules

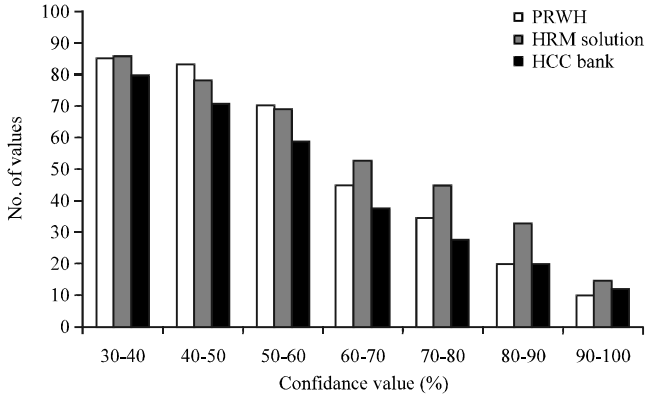

Fig. 9: Confidence distribution of rules

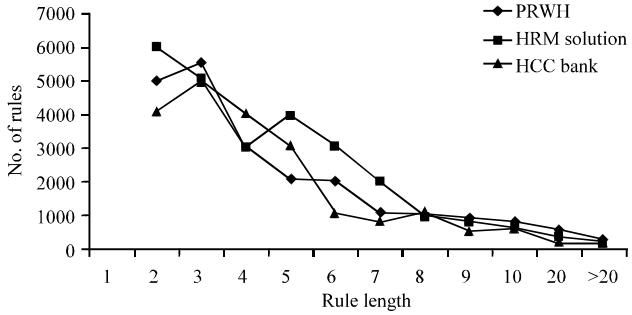

Fig. 10: Distribution of number of rules VS rules length

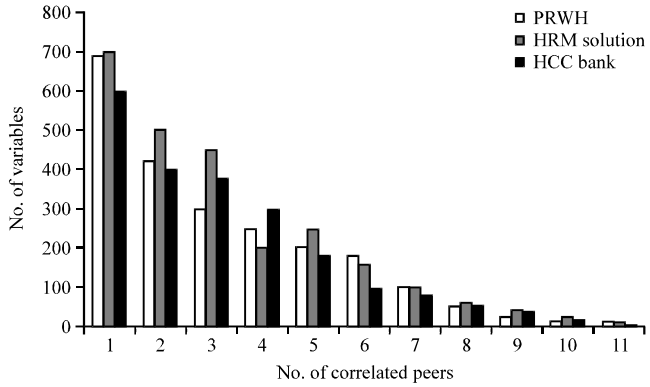

Fig. 11: Distribution of correlated variables with different number of peers

variable and around 20% are correlated with two variables. This result indicates that access correlations do not exist between any two random variables. Even though most structures contain many fields, only those fields that have true semantic connections have access correlations.

Extracting copy-paste code: The percentage of code reuse by copying and pasting the code for each application is shown in Fig. 12. The findings strongly validating the observation that developers are often copying and pasting the code to save development time.

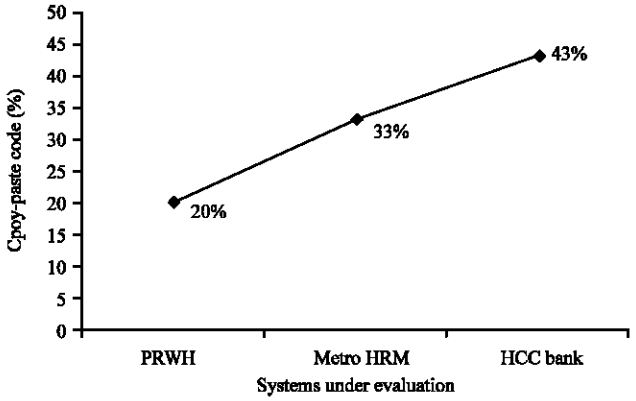

Fig. 12: Percentage of copy-paste code in evaluated systems

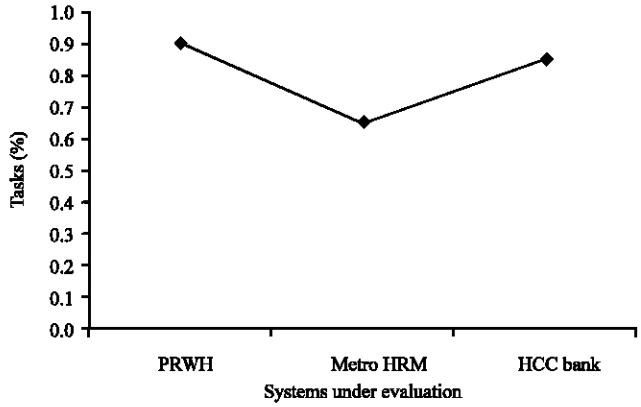

Fig. 13: Percentage of API usage patterns suggested by proposed framework

Extracting API usage patterns: To investigate the usefulness of proposed framework for providing code related to task at which programmer currently working it is also evaluated on systems under evaluation. The user provides query statement by specifying one or more line of code in search context. The framework automatically retrieves relevant source code files (relevant APIs) from web or from previous projects. The framework is evaluated for API usage to see either it successfully suggest solution for given query. A task is considered successful if the desired task on which developer is currently working is enabled with at least one recommended solution. The current implementation of framework only suggests frequent code samples relevant to current programming tasks. Therefore, code is manually transformed into appropriate compilable code snippet. The percentage of task successfully completed by proposed framework is shown in Fig.13. The x-axis shows the systems under evaluation and y-axis shows the percentage of task completed for each system.

Detecting rules violations: Before finding the violation of extracted rules, they are validated by developers. Only the validated rules are used to detect the violation. The developed prototype system has reported many violations of validated rules in systems under evaluation. The system is also evaluated in term of space and time efficiency for detecting violation as shown in Table 7.

Table 7: Time and space evaluation of violation detector

	Detecting violation	
Software	Time	Space (Mb)
PRWHP	43 sec	2.8
Metro HRM Solution	54 sec	4.8
HCC Bank Project	2 min	6.2

Table 8: No. of rule violation bugs detected by prototype system

System evaluated	No. of rule violations bugs report by prototype system	No. of new bug found	No. of new bugs confirmed
PRWHP	57	25	18
Metro HRM solution	43	30	21
HCC bank project	30	18	11
Total	130	73	50

Table 9: No. of copy paste bugs

System evaluated	No. copy paste bugs report by prototype system	No. of new bug found	No. of new bugs confirmed
PRWHP	33	24	18
Metro HRM Solution	71	53	42
HCC Bank Project	63	48	36
Total	167	125	96

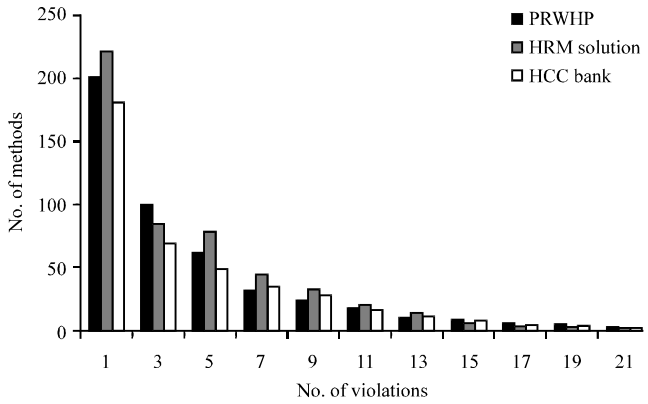

Fig. 14: Number of violation per functions in systems under evaluation

It can be observed from table violation detector is space and time efficient for large applications, e.g., in HCC Bank application it took less than a minute and reported 69 violations. Those violations occurred in 81 methods since one method may contain several violations.

The total number of rule violation bug detected by proposed framework from systems under evaluation is shown in Table 8. The top 130 bugs in table are examined. It is identified that 73 are true bugs which are send to developer for confirmation. The developers has confirmed 50 out of 73 bugs as true bugs and fixed. Similarly, the copy-paste code related bugs are shown in Table 9. In copy paste code related bugs top 167 bugs are examined and send for validation to corresponding developers. Out of 167, the developers have confirmed 96 as true bug and need to fix.

The ratio of number of violation to number of methods can give a general idea how many of method containing are violation. The number of violation per function in systems under evaluations is shown in Fig. 14. By inspecting violations manually it is

identified 70% of them were defects, other were pointing to code that could be improved. Confirmed violations are reported on bug tracking system for corresponding developers to fix.

CONCLUSION

This study proposed an innovative approach that applies data mining on software engineering data. It is demonstrated how programming patterns, in this case function usage, data access in variables, code clone and API usage pattern can potentially helpful in various software engineering tasks. Multiple kinds of bugs are uncovered by feeding extracted patterns to violation detection tool. By running the tool on various software systems a number of interesting and non-obvious rules that are critical for developers to understand and follow are identified. Experiments show that identifying programming patterns and their violation helps in software testing, monitoring software quality, meeting project schedule by providing speedy solution, analyzing source code for re-engineering purpose and program optimization. So far this is first integrated framework intends to extract multiple programming patterns. Also the proposed method for violation detection can detect different kind of bugs in single pass. Therefore, reducing cost for deployment of multiple tools to detect different kind of bugs in software systems. Results indicates that maintaining the programming rules in specification database are very important, since developer can refer those rules during development which dramatically reduce the time for static verification. Also maintaining clone code segments is very helpful for developers because it is commonly used in large software systems and can easily introduce difficult- to- detect bugs. This study is useful to motivate the software development organizations to integrate functionality to maintain programming rule in software development environments such as Microsoft Visual Studio.

REFERENCES

Agrawal, R. and R. Srikant, 1994. Fast algorithms for mining association rules. Proceedings of the 20th International Conference on Very Large Data Bases, September 12-15, 1994, San Francisco, USA., pp: 487-499.

Agrawal, R. and R. Srikant, 1995. Mining sequential patterns. Proceedings of the 11th International Conference on Data Engineering, March 6-10, 1995, Taipei, Taiwan, pp: 3-14.

Aho, A.V., M.S. Lam, R. Sethi and J.D.Ullman, 2007. Compilers: Principles, Techniques and Tools. 2nd Edn., Addison-Wesley, New York.

Chang, R.Y., A. Podgurski and J. Yang, 2007. Finding what's not there: A new approach to revealing neglected conditions in software. Proceedings of the International Symposium on Software Testing and Analysis, July 9-12, 2007, ACM Press, New York, pp: 163-173.

Engler, D., D. Chen, S. Hallem, A. Chou and B. Chelf, 2001. Bugs as deviant behavior: A general approach to inferring errors in systems code. ACM SIGOPS Oper. Syst. Rev., 35: 57-72.

Holmes, R. and G.C. Murphy, 2005. Using structural context to recommend source code examples. Proceedings of the 27th International Conference on Software Engineering, May 15-21, 2005, ACM Press, New York, pp: 117-125.

Khatoon, S., G. Li and R.M. Ashfaq 2011a. Framework for Automatically Mining Source Code J. Software Engin., 5: 64-77.

Khatoon, S., A. Mahmood and G. Li, 2011b. An evaluation of source code mining techniques. Proceedings of the 8thInternational Conference on Fuzzy Systems and Knowledge Discovery (FSKD), July 26-28, 2011, China, pp: 1929-1933.

Li, Z. and Y. Zhou, 2005. PR-Miner: Automatically extracting implicit programming rules and detecting violations in large software code. Proceedings of the 10th European Software Engineering Conference and 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering, September 5-9, 2005, Lisbon, Portugal, pp: 306-315.

Li, Z., S. Lu, S. Myagmar and Y. Zhou, 2004. CP-Miner: A tool for finding copy-paste and related bugs in operating system code. Proceedings of the 6th Conference on Symposium on Opearting Systems Design and Implementation, December 2004, San Francisco, CA., pp: 289-302.

Lu, S., S. Park, C. Hu, X. Ma, W. Jiang, Z. Li, R. Popa and Y. Zhou, 2007. MUVI: Automatically inferring multi-variable access correlations and detecting related semantic and concurrency bugs. ACM SIGOPS Oper. Syst. Rev., 41: 103-116.

Mandelin, D., L. Xu, R. Bodik and D. Kimelman, 2005. Jungloid mining: Helping to navigate the API jungle. ACM SIGPLAN Not., 40: 48-61.

Michail, A., 2000. Data mining library reuse patterns using generalized association rules. Proceedings of 22nd International Conference on Software Engineering, June 4 -11, 2000, ACM New York, NY, USA, pp: 167-176.

Pei, J., J. Han, M.A. Behzad, P. Helen, Q. Chen and M.C. Hsu, 2001. PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. Proceedings of the 17th International Conference on Data Engineering, April 2-6, 2001, Heidelberg, Germany, pp: 215-224.

Ramanathan, M.K., A. Grama and S. Jagannathan, 2007. Path-sensitive inference of function precedence protocols. Proceedings of the 29th International Conference on Software Engineering, May 20-26, 2007, Minneapolis, MN., USA., pp: 240-250.

Sahavechaphan, N. and K. Claypool, 2006. XSnippet: Mining for sample code. ACM SIGPLAN Not., 41: 413-430.

Thummalapenta, S. and T. Xie, 2007. Parseweb: A programmer assistant for reusing open source code on the web. Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering, November 4-9, 2007, Atlanta, Georgia, USA., pp: 204-213.

Wahler, V., D. Seipel, J.W.V. Gudenberg and G. Fischer, 2004. Clone detection in source code by frequent itemset techniques. Proceedings of the 4th IEEE International Workshop Source Code Analysis and Manipulation, September 16, 2004, IEEE Computer Society, Chicago, IL., USA., pp: 128-135.

Xie, T. and J. Pei, 2006. MAPO: Mining API usages from open source repositories. Proceedings of the 2006 International Workshop on Mining Software Repositories, May 22-23, 2006, Shanghai, China, pp: 54-57.