JOURNAL OF SOFTWARE ENGINEERING & INTELLIGENT SYSTEMS
ISSN 2518-8739
31st August 2017, Volume 2, Issue 2, JSEIS, CADMEI Copyright © 2016-2017
www.jseis.org

Detecting software defect patterns and rule violation identification in source code

[1] Rana Muhammad Ashfaq, [2] Shahbaz A.K. Ghayyur
[1,2]DCS & SE International Islamic University, Islamabad, Pakistan
Email: [1]ashfaqasp@gmail.com, [2]shahbaz.ahmed@iiu.edu.pk

ABSTRACT

Software engineer as a rule takes after a wide range of sorts of examples in source code, a large portion of which are excessively monotonous, making it impossible to archive by developer. At the point when these examples are abused by developer who are unconscious of or disregard those, deformities can undoubtedly present. In this manner, it is profoundly attractive to build up a device that consequently separate different sorts of examples and recognize infringement naturally. We discover 30 systems for discovering designs furthermore discover 17 designs from Literature and from IT master. 30 procedures take after various their own particular systems and own calculations. In this study, we proposed a strategy which consequently extricates different sorts of examples from source code and absconds recognition technique to discover infringement from removed examples. Proposed method distinguishes 17 sorts of infringement, for example, Function utilized together infringement, duplicate glue or clone related imperfections, and variable connection related deformities, reuse API and others. Proposed procedure is accepted by building up a model that created in any mechanical dialect like VB and applies on substantial application like ERP. Results indicate proposed procedure enormously lessened the exertion of physically checking imperfections or infringement from source code by software engineers.

Keywords: source code; data mining; copy paste; clone detection; rule violation; function used

1. INTRODUCTION

Software engineering (SE) is a profession dedicated to designing, implementing, and modifying software so that it is of higher quality, more affordable, maintainable, and faster to build. It is a "systematic approach to the analysis, design, assessment, implementation, test, maintenance and reengineering of software, that is, the application of engineering to software". Therefore, software development may include research, new development, modification, reuse, re-engineering, maintenance, or any other activities that result in software products. The primary goal of software development is to deliver high quality software in the least amount of time whenever possible. To achieve these goal, software engineers are increasingly applying data mining algorithms to various software engineering (SE) tasks [1] to improve software productivity and quality.

To deliver high quality software automatic detection of bugs remains one of the most active areas in SE research. Practitioners desire tools that would automatically detect bugs and flag the location of bugs in their current code base so they can fix these bugs. In this direction, much work has been done to develop tools and techniques which analyze large amount of data about a software application such as source code, to uncover the dominant behavior or patterns and to flag variations from that behavior as possible bugs.

2. REVIEW OF RELEVANT RESEARCH AND THEORY

2.1 Rule violation from source code

Rule mining techniques induce set of rules from existing projects which can be used to improve subsequent development or new project development. Some approaches were planned to discover rule-violating flaws. “Engler et al., [2] developed a static verification tool by using compiler extensions called checkers (written in the Metal language) to match rule templates, derived from knowledge of typical programming errors, against a code base”. Proposed tool extracts programming beliefs from acts at different location of source code by exploiting all possible paths between function call and cross check for violated beliefs e.g. a “dereference of a pointer, p, implies a belief that p is non-null, a call to unlock (1) implies that 1 was locked etc.”. Rule template represent general programming rules such as such as “<a>” must be paired with “<b>” and Checkers, match rule templates to find the rules instance and discover code locations where it interrupts a rule that equal a current pattern. Two types of rules categories:

MUST-rules (inferred from acts that imply beliefs code "must" have) and MAY-rules (Inferred from acts that imply beliefs code "may" have) are identified. For MUST rules, internal consistency is checked and contradictions is directly flagged as bugs; for MAY-rules, a statistically based method is used to identify whether a possible rule must hold. Proposed approach applies statistical analysis, founded on how numerous times the rule holds and how numerous it does not to rank deviations from programmer beliefs inferred from source code.

"Li and Zhou [3] proposed a method called PR-Miner (Programming Rule Miner), that uses item-set mining technique to automatically extract general programming rules from software code written in an industrial programming language such as C and detect violations. It transforms a function definition into an item-set by hashing program elements to numbers". In this conversion process, similar program elements are mapped to the same number, which is accomplished by treating identifiers with the same data types as identical elements, regardless of their actual names. "By using the frequent item-set mining algorithm called FPclose PR-Miner [3] extracts rules from possible combination of multiple program elements of different types including functions, variables, data types, etc. that are frequently used together in source code and find association among them". For efficiency, "PR-Miner [3]" generates only closed rules from a mined pattern. "The rules extracted by PR-Miner are in general forms, including both simple pair-wise rules and complex ones with multiple elements of different types. By identifying which elements are used together frequently in the source code, such correlated elements can be considered a programming rule with relatively high confidence [3]".

2.2 Variable used together

"Lu et al,. [6] developed a tool called MUVI to mine variable pairing rules which applied the frequent itemset mining technique to automatically detect two types of bug i.e. (1) multi-variable inconsistent update bugs and (2) multi-variable related concurrency bugs, which may result due to inconsistent update of correlated variables, the variables that need to be accessed together. For example, thd->db_length describes the length of the string thd->db, so whenever thd->db is updated, thd->db_length should be updated consistently. The access together variables are those which appear in the same function with less than maximum distance statement apart, and collected by statically analysis of each function to form Acc-Set. MUVI's applied FPclose algorithm to Acc_Set database, consisting of the Acc_Sets of all functions from the target program and output set of variable accessed more than minimum support number of functions. MUVI only focused on two kinds of variables: global variables and structure/class fields [6]".

2.3 Detecting copy paste code

A tool called Dup is developed which detect two types of matching code that is either exactly the same or name of parameters such as variable and constant are substituted. "It performs the following sub processes: 1) Lines of source files are first divided into tokens by a lexical analyzer, 2) replacement of tokens (identifiers of functions, variables, and types) into a parameter identifier, 3) parameter tokens are encoded using a position index for their occurrence in the line. 4) All prefixes of the resulting sequence of symbols are then represented by a suffix tree, a tree where suffixes share the same set of edges if they have a common prefix. 5) Extraction of matches by a suffix-tree algorithm, if two suffixes have a common prefix, clearly the prefix occurs more than once and can be considered a clone [8]".

"CCFinder is another token based clone detection technique with additional transformation rules to remove minor difference in source code". "It transforms source code into tokens sequence through lexical analyzer to detect clone code portions that have different syntax but have similar meaning and applies rule-based transformation such as regularization of identifiers, identification of structures, context information and parameter replacement of the sequence. It uses source normalizations to remove superficial differences such as changes in statement bracketing (e.g., if (a) b=2; vs. if(a) {b=2;}). Finally, clone pairs, i.e., equivalent substrings in the token sequence, are identified using suffix-tree matching algorithm [7].

"CP-Miner [13] applies data mining to identify copy-paste defect in operating system code". By using frequent subsequence mining and tokenization technique it notices copy-paste-related incorrect variable-name bugs. "It transforms a basic block into number by tokenizing its component such as variable, operators, constants, functions etc. Once all the components of a statement are tokenized, a hash value digest is computed using the "hashpjw"

hash function [13]

2.4 API usage pattern

"Another line of related research is how to write APIs code. A software system cooperates with third-party libraries through various APIs. Using these library APIs often needs to follow certain usage patterns. These patterns aid developers in addressing commonly faced programming problems such as what checks should precede or follow API calls, how to use a given set of APIs for a given task, or what API method sequence should be used to obtain one object from another. "Much research has been conducted to extract API usage rules or patterns from source code by proposing tools and approaches which helps developers to reuse existing frameworks and libraries more easily including [18, 21, 24, 25]".

"In this direction, Michail, [25] described how data mining can be used to discover library reuse patterns in existing applications by developing a tool CodeWeb". "It excavations association rules such as what application classes getting from a specific library class often instantiate another class or one of its children Based on itemset and association-rule mining CodeWeb [25] uncover entities such as components, classes, and functions that occur frequently together in library usages". "Michail [25] explains by browsing generalized association rules, a developer can discover patterns in usage in a way that take into account inheritance relationship".

"Xie and Pei, [21] developed a tool MAPO which mines frequent usage patterns of API through class inheritance". It uses API's usage history to identify methods call in the form of frequent subsequences. "The code search engine receives a query that describes a method, class, or package for an API and then searches open source repositories for source files that are relevant to the query. The code analyzer analyzes the relevant source files returned by the code search engine and produces a set of method call sequences, each of which is a callee sequence for a method defined in the source files. The sequence preprocessor inlines some call sequences into others based on caller-callee relationships and removes some irrelevant call sequences from the set of call sequences according to the given query.

"PARSEWeb [20] analyzes the local source code repository to and constructs a directed acyclic graph". "PARSEWeb [20] identifies nodes that contain the given Source and Destination object types and extracts a Method-Invocation Sequences (MISs) by calculating the shortest path between those Source and Destination nodes".

3. LIMITATIONS IN LITERATURE

3.1 Rule violations

Engler et al. [2] proposed a method based on fixed rule templates that need specific knowledge about the software. "Only one type of pattern analysis that is "function A must be paired with function B". Li and Zhou [3] proposed a method called PR-Miner (Programming Rule Miner). Not using inter-procedural analysis hence rules crossing across multiple function definitions are not detected. Does not consider data flow and control relationship hence result many false negatives. "Some functions may have the same name but different semantics, PR-Miner does not differentiate them which results in false rules. Not detecting violation propagated by copying and pastes the code [3]. Chang et al., [5] proposed an approach to mine implicit condition rules and to detect neglected conditions by applying frequent sub graph mining on C code. Only detects restricted rules including precondition and post condition of function calls and does not detect other kind of rules and violations. Only support C language code

3.2 Variable Used Together

Lu et al., [6] developed a tool called MUVI to mine variable pairing rules. Variables access directly by caller functions are only handled. Only Deals with Variable Used Together pattern.

3.3 Detecting copy paste code

A string based approach to locate code duplication is proposed by Baker [8]. The line-by-line method cannot identify clone in different line structure and also has a weakness in sensing clone code quotas that have different syntax but have similar meaning. CCFinder is another token based clone detection technique with additional

transformation rules to remove minor difference in source code. "CCFinder detects clones that are small in size that is smaller than 60 tokens. It is not scalable to large-scale software because it consumes large amount of computer memory to store the transformed text".

CP-Miner [13] applies data mining to identify copy-paste defect in operating system code. Segments which are same syntax structure but different semantic are also detected as copy paste segment, since " CP- Miner simply follows the program order to compose larger copy-pastes, it is likely that a wrong composition might be chosen". "CP-Miner simply compares the identifiers in a pair of copy-pasted segments in strict order if ordering of statement is changed it report false positive. "CP Miner" cannot tell which segment is original and which is copy-pasted from the original.

3.4 API usage pattern

Holmes et al. [16] developed Strathcona, an Eclipse plug-in, that enables location of relevant code in an example repository. The limitations are: (i) Every empirical is general that it is not adjusted to an exact mission of object method call. This results often in unrelated examples; and (ii) " each heuristic utilizes all defined context, irrespective of whether the context is relevant or not. This over-constraining of the heuristic can result in too few examples or sometimes no examples [16]".

Mandelin et al., [19], developed a tool called Prospector for automatically synthesize the list of candidate jungloid code based on simple query that described the required code in term of input and output. PARSEWeb developed by [20] uses Google code search for collecting relevant code snippets and mines the returned code snippets to find solution jungloids. PARSEWeb suggests only the frequent MISs and code samples, but cannot directly generate compilable code.

4. PROBLEM STATEMENT

After literature survey, we conclude that all techniques are based on single pattern and single technique. There are many studies on this area as discussed in literature review section. There is not anyone system is available that support multiple patterns on single technique. Literature Review shows that there is great depth in this area.

4.1 Research questions

RQ1: What are the source codes patterns identified in literature which deal with the source code Violation?
RQ2: What are the shortcomings of current approaches for Rule violation identification in source code?
RQ3: How can we solve problem Identified in RQ 1 & 2 for detection for rule Violation in Source Code?

5. PROPOSED METHODOLOGY

The most appropriate method for this research is "controlled experiment". The controlled experiment is that in which investigating of cause and effect relation is performed. This means that investigation is done in controlled environment of the lab. In controlled experiment, all possible factors which can affect results are remain in fully observation. Other types of research like case study, systematic literature review, simulation, survey, ethnography, action research, and benchmarking as all these researches are not appropriate for this kind of research because these methods are not fully controlled and chance is that there will be remain biasness during performance and some unpredictable factors may be involved in during performance of any of these methods. Therefore, controlled experiment is appropriate method for this research.

Our method for mining several patterns is more common. We suggest a innovative technique called Integrated Pattern System that uses a Token based method to automatically excerpt common programming patterns from software code written in an developed programming language such as VB and Identify violations with "little effort from programmers". We also offer a resourceful algorithm to identify violations to the mined programming patterns and detect violation. It supports different programming languages like C++, VB, C# for find patterns and violation. In our proposed technique, we integrated four source code patterns.

6. PROPOSED TECHNIQUE

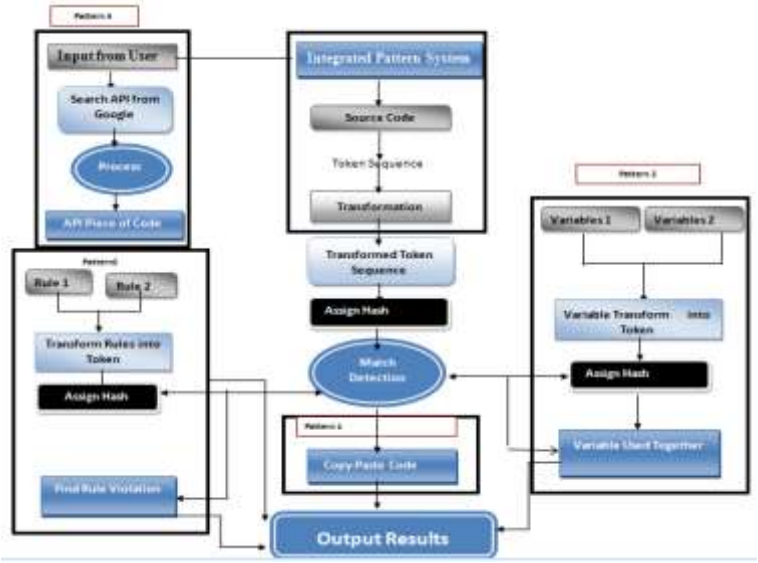

Figure. 1 Proposed technique

- It will be the first technique ("to the best of our knowledge [3]") to automatically
- discovery the commonly existent multiple patterns from large source code.
- First technique that support multiple types of patterns on single platform and single method.
- That can support in improvement of multiple software engineering tasks over different phases of development life cycle e.g. assisting programming in writing code, bug detection and software maintenance.
- Language Compatibility (Support multiple language Source code)
- Platform independent
- No Pre-Requisite required
- No Need of Source code Knowledge
- Develop our own algorithms for pattern matching
- Time and Cost saving.
- Resource Saving

7. CONCLUSION

In this research study, we have proposed a new technique that helps in exploration of common multiple patterns in the source code. The technique is based on a single method and it support several categories of the patterns on a single platform. The technique is helpful in solving multiple software problems and support in high language compatibility. The given technique is time-saving, cost effective and save the resources.

REFERENCES

1. A. Hassan, and T. Xie, "Mining software engineering data," in Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2, pp. 503-504, 2010".
2. D. Engler, D. Chen, S. Hallem et al., "Bugs as deviant behavior: A general approach to inferring errors in systems code," ACM SIGOPS Operating Systems Review, vol. 35, no. 5, pp. 57-72, 2001".
3. Z. Li, and Y. Zhou, "PR-Miner: Automatically extracting implicit programming rules and detecting violations in large software code," in Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering, pp.306-315, 2005".
4. M. Ramanathan, A. Grama, and S. Jagannathan, "Path-sensitive inference of function precedence protocols," in 29th International Conference onSoftware Engineering (ICSE 2007),pp.240-250,

2007".
5. R. Chang, A. Podgurski, and J. Yang, "Finding what's not there: a new approach to revealing neglected conditions in software," in Proceedings of the 2007 international symposium on Software testing and analysis, pp. 163-173, 2007".
6. S. Lu, S. Park, C. Hu et al., "MUVI: automatically inferring multi-variable access correlations and detecting related semantic and concurrency bugs," ACM SIGOPS Operating Systems Review, vol. 41, no. 6, pp. 103-116, 2007".
7. T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: a multilinguistic token-based code clone detection system for large scale source code," IEEE Transactions on Software Engineering, pp. 654-670, 2002".
8. B. Baker, "On finding duplication and near- duplication in large software systems," in Proc. Second IEEE Working Conf. Reverse Eng., pp. 86-95, 1995".
9. I. Baxter, A. Yahin, L. Moura et al., "Anna and L.Bier,"Clone Detection Using Abstract Syntax Trees," Proc. Int" l Conf," Software Maintenance, pp. 368-377, 1998".
10. V. Wahler, D. Seipel, J. Wolff et al., "Clone detection in source code by frequent itemset techniques," in Fourth IEEE International Workshop on Source Code Analysis and Manipulation, pp. 128-135, 2004.
11. J. Krinke, "Identifying similar code with program dependence graphs," in Proceedings of the 8th Working Conference on Reverse Engineering, WCRE 2001, pp. 301-309, 2001".
12. W. Qu, Y. Jia, and M. Jiang, "Pattern mining of cloned codes in software systems," Information Sciences, 2010, 2010".
13. Z. Li, S. Lu, S. Myagmar et al., "CP-Miner: A tool for finding copy-paste and related bugs in operating system code," in Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation-Volume 6, pp. 20, 2004".
14. V. R. Basili, L. C. Briand, and W. L. Melo, "How reuse influences productivity in object-oriented systems," Communications of the ACM, vol. 39, no. 10, pp. 116, 1996".
15. G. T. Heineman, and W. T. Councill, Component- based software engineering: putting the pieces together: Addison-Wesley USA, 2001".
16. R. Holmes, and G. C. Murphy, "Using structural context to recommend source code examples," in Proceedings of the 27th international conference on Software engineering, pp. 117-125, 2005".
17. G. T. Leavens, and M. Sitaraman, Foundations of component-based systems: Cambridge Univ Press,2000".
18. N. Sahavechaphan, and K. Claypool, "XSnippet: mining For sample code," ACM SIGPLAN Notices, vol. 41, no. 10, pp. 413-430, 2006".
19. D. Mandelin, L. Xu, R. Bodík et al., "Jungloid mining: helping to navigate the API jungle," ACM SIGPLAN Notices, vol. 40, no. 6, pp. 48-61, 2005".
20. S. Thummalapenta, and T. 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, pp. 204-213, 2007".
21. T. Xie, and J. Pei, "MAPO: Mining API usages from open source repositories," in Proceedings of the 2006 international workshop on Mining software repositories, pp. 54, 2006"

AUTHORS PROFILE

Rana Muhammad Ashfaq completed his MS in Software Engineering from Department of Computer Science and Software Engineering at International Islamic University, Islamabad, Pakistan. Currently, he is pursuing his PhD in Software Engineering from Department of Computer Science and Software Engineering at International Islamic University, Islamabad, Pakistan. His research interests are in software engineering, software testing, requirements engineering and model driven development.

APPENDIX

List of all possible source Code Patterns			
S#	**Data**	**Patterns**	**Paper title**
1	Source Code	"Relationship Between code Entities Inheritance Relationship"	"Library reuse pattern Michail 2000"
2		"Statement S sequence in basic block"	"CP Miner 58"
3		"Sequence of iterative statements"	"A Framework of Source code search using Program patterns"
4		"Extract Business rules"	"Extract Business rules from source code Harry M.saneed"
5		"Implicit Conditional Rules"	"Revealing Neglected conditions[Change et al.,ISSTA 07]"
6		"Call patterns that occur frequently"	"Deviant patterns as potential bugs"
7		"Element that are frequently used together in source code (Set of functions, variables and data types) (Open conn and close conn)"	"Programming rules PR-Miner [3]"
8		"Mining Rules from source code"	"Bugs as deviant behavior [Engler et al SOSP01]"
9		"API Sequence (Source->Destination)"	"PARSEWEB [20]"
10		"Neglected Condition"	"NEGWEB"
11		"Duplicate Code"	"CLone Detection in source code by Frequent itemset Technique"
12		"Sequence of Function call patterns"	"Comparing approach to mining source code for call usage patterns"
13		Page life cycle stages (initiate,load,unload)	
14		Behavior of control property (Button,Label,Dropdown list)	
15		Behavior of N-Tier/3-Tier Layer (data Logic,Business Lodic and Presentation layer)	
16		Behavior of calling Function and passing values to called functions	
17		"Structural view of program"	"Facilitating program comprehension by mining association rules from source code Christos Tjortjis"

Patterns and ToolsPatterns that exist in Source Code

Techniques →

TOOL/Title	Relationship Between code Entities Inheritance Relationship	Statement sequence in basic block	Sequence of iterative statements	Extract Business rules	Implicit Conditional Rules	Function used together	Duplicate Code Or Copy paste Or Clone	Reuse of API	Neglected Condition	Sequence of Function call patterns	Page life cycle stages	Behavior of control property	Variable used together	Structural view of program
"Duploc [16]"							Y							
"Dup [2]"							Y							
"CCFinder"							Y							
"Gemini"							Y							
"Gemini"							Y							
"CP-Miner"		Y					Y							
"CloneDr"							Y							
"Asta"							Y							
"cpdetector"							Y							
"Deckard [20"]							Y							
"Tairas"							Y							
"CloneDetection"							Y							
"Konto"							Y							
"Covet"							Y							
"Davey [13]"							Y							
"Duplix"							Y							
"KomoRag"							Y							
"GPLAG"							Y							
"PR Miner"					Y	Y				Y				
"Comparing approach"										Y				
"NegWeb"									Y					
"PARSEWEB"								Y						
"Revealing Neglected conditions"					Y		Y							
"Extract Business rules"														
"Library reuse pattern"	Y													

"mining association rules"														
"Static Analysi"s						Y								
"CHRONICLE"										Y				
"MUVI"													Y	
"CCFinder"							Y							
"CloneDetect ion"							Y							
"XSnippet"								Y						
"MAPO"								Y						
"MAC"														
"KClone"							Y							
"Colne Detection Via Structural abstraction"							Y							
"RTF(Repeated token Finder)"							Y							
"A Technique for Just-In-Time Clone Detection in Large Scale system"							Y							
"program Dependence Graph"							Y							
"NiCard Clone Detector"							Y							
"Static Bug Detection Through analysis of Inconsistent Clones"							Y							
"(MeCC)"							Y							
"Code Clone detection using Parsing Action"							Y							
"Clone Manager"							Y							
"The detection of faulty code violating implicit coding rule."					Y									

"Extracting Business Rules from Source Code"				Y										