ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 3, March 2015.

Measuring Code Quality

Ms. Dhanshree Kapse[1], Ms. Karishma Kabra[2], Ms. Madhavi Chopade[3], Mr. Kailas Palhal[4]

BE, Department of Computer Engineering, SSBT's COET Bambhori, Jalgaon (M.S.), India[1,2,3,4]

Abstract: Every software Industry requires the quality of code. Formal specifications can help with program testing, optimization, reface, documentation, and, great significance debugging and restore however, they are difficult to do manually, and automatic mining techniques suffer from 90–99% false positive rates. To address those problems this project proposes to temporal-property miner by incorporating code quality metrics. This measure code quality by extracting additional information from the software engineering process, and using this information from code that is more equal to be correct as well as code that is less equal to be correct. When used as a preliminary processing step for an existing specification miner, project technique identifies which input is most correct program, the same number of specifications using only 45% of their original input. As a novel inference technique, this approach has few false positives in practice (63% when balancing precision and recall, 3% when focused on precision), even though finding useful specifications (e.g. find many bugs) on over 1.5 million lines of code.

Keywords: Quality, Errorless, Reliably.

I. INTRODUCTION

Buggy behaviour in software costs up to Rs 4410 billion each year in the software industry. Maintenance of software is consumed up to 90% of the total cost of software projects. It is very hard to repair a coding error [1]. It is hard to imagine software without bugs. Testing is the detection method of bugs [7]. Writing a correct program is more difficult. Verification tool find many errors in programs [3]. More quality of program are hard for humans to construct, and incorrect programs are difficult for humans to debug. This project focuses on a second quality measuring techniques that produce a larger quality of code and make more precise that may be easier to evaluate for correctness [1]. The security of code has become increasingly important in the last decade. More and more software enterprise applications deal with sensitive financial data, which, if compromised, in addition to downtime can mean millions of dollars in damages. It is important to protect these codes from hacker attacks [3]. Method for debugging temporal specifications is found. Given data collected during one or more programs, the miner generates a large number of short scenarios. If some of the modules contain errors (as often happens), some of the scenario traces are also erroneous [6]. Many projects in the past the centre of interest on suffers problems caused by the dangerous nature of C, such as buffer spread over and format string unsafe. However, in recent years, Java has emerged as the language of choice for building large complex systems, in part because of language safety features and eliminates problems such as buffer overruns [3].Contemporary software emphasizes components with clearly specified APIs. Components such as Java library classes have a clearly specified static interface that consists of all the (public) methods, along with the types of input parameters and return values [4].

The rest of this paper is organized as follows. In Section 2, we describe existing techniques. Section 3 presents proposed system that clears our approach. Section 4 describes results and Section 5 describes conclusion.

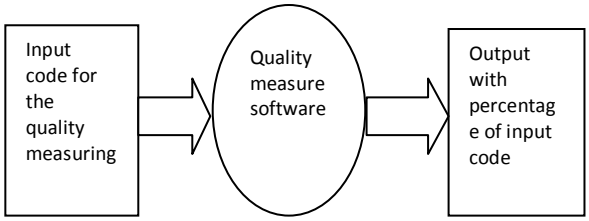

Fig (a) Block diagram

II. EXISTING SYSTEM

There are dozens of existing systems for measuring code quality. Section describes some of existing techniques and compares them with proposed system.

A .Specification Mining

State machines can be observed by a programmer, to remove impurities the specification and identify errors, and can be used by automatic verification tools, to find bugs [7]. It only refines the specification and identifies errors.

B. Debugging Temporal Specifications with Concept Analysis

Short program execution traces that program verification tools generate from specification violations and that specification miners extract from programs. Manually finding by investigation is a straightforward way to debug a specification [6]. But this method is tedious and error-prone because there may be hundreds or thousands of traces to inspect.

C. Finding and Preventing Run Time Error Handling Mistakes

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 3, March 2015.

Dataflow analysis to finding a class of error-handling mistakes: that arises from either failure to release resources or to clean up properly along all paths. Many real-world programs fail to comply such resource safety policies because of incorrect error handling. Flow-sensitive analysis keeps track of outstanding act along program paths and does a marked by exactness and accuracy modelling of control flow in the presence of exceptions [5]. But it found only 800 error handling mistakes almost 4 million lines of Java code.

D. Privately finding specifications

By sharing data, able to discover specifications, and thus find out software bugs, than never share data. However, because sharing data agreement privacy, present a way for unsettled and publish data and yet still discover large specifications and bugs than they never shared data. In aggregate these unsettled traces can be analysed to learn correct specifications of program behaviour. The unsettled traces cannot be analysed to determine that one contributed buggier find by investigation than another. The learned specifications are of benefit to all [2]. But this method finds specifications 85% of the bugs that a no-privacy approach would find.

III. PROPOSED SYSTEM

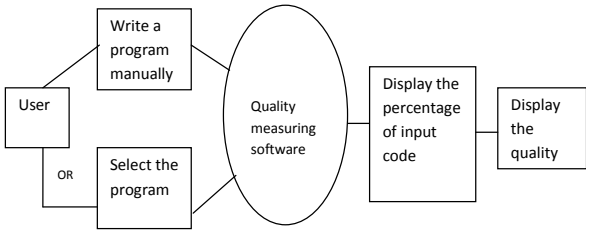

Fig (b) Architecture of System

In above fig(b) can shows that Architecture of system, in that user can give input as a code it may be in the form of manually written program or select available program directly, then this technique can measure the quality of the software .By measuring quality in the form of complied with errorless, use of depreciative API's, use of functions code reliably that includes use of components, use of blank spaces, program well written, security that includes private as well as protected accordingly find the percentage and display the result.

IV. RESULT

Result Prediction	Expected Values	Obtained Values
Compiled with Errorless	5%	5%
Use of depreciative API's	10%	10%
Use of Functions	30%	15%
Code reliably		
Use of Comments	20%	15%
Use of Blank Spaces	10%	10%
Program well written	10%	10%
Security		
Private / Protected	15%	10%
Quality of Code	100%	75%

TABLE 1.1 RESULT OF CODE AS PER MODULES.

The result is displayed in terms of percentage. When the result is between 70- 100% then the quality of code is excellent code. When the result is between 50-70% then quality of code is best, and when the result is below 30-50% then the quality of code is good, and below 30% the quality of code is bad. As shown in table 1.1, the quality of the code is 75%, Hence can conclude that the quality of code is excellent. There are seven modules are described in this project. At the result addition of all modules percentages which are matched with the input code are displayed. From result percentages the quality of code is measured. This project is used to select the best quality code form number of code. As the use of software Indus try is more efficient for finding out the excellent quality code.

V. CONCLUSION

Approach improves the performance of existing trace-based miners by focusing on high-quality traces. This technique can also be used alone basic miner learns more specifications and identifies hundreds more violations than previous miners. A combination of independent, imperfect code quality metrics may prove useful to other automatic static analyses that look at source code to draw conclusions about code or predict faults. Believe that this technique is an important first step towards real-world utility of automated specification mining, as well as to the increased use of quality metrics.

ACKNOWLEDGMENT

We would like to express our deep gratitude and sincere thanks to all who helped us to complete this paper work successfully.
Our sincere thanks to principal **Dr.Prof.K.S.Wani sir**, SSBT COET for having provided facilities for completion of our paper work.
Our deep gratitude goes to **Dr.Prof.G.K.Patnaik sir**, Head of the department, for getting us opportunity to conduct this paper work.
We are sincerely thankful to **Mr.D.D.Puri sir**, guide for his valuable suggestions and guidance needed at time.
Last but not least we thank the Almighty God who makes everything happen.

REFERENCES

[1] Claire Le Goues, Westley Weimer "Measuring Code Quality to Improve Specification Mining", 2012.
[2] Westley Weimer and Nina Mishra "Privately finding specifications" ,2008.
[3] Benjamin Livshits and Monica S. Lam "Finding Security Vulnerabilities in Java Applications with Static Analysis ",2005.
[4] R. Alur, P. Cerny, P. Madhusudan, and W. Nam, "Synthesis of interface specifications for Java classes", in POPL, 2005.

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 3, March 2015.

[5] Westley Weimer George C. Necula "Finding and Preventing Run Time Error Handling Mistakes" 2004.
[6] G. Ammons, D. Mandelin, R. Bodk, and J. R. Larus, "Debugging temporal specifications with concept analysis in Programming Language Design and Implementation",2003, pp. 182195.
[7] Glenn Ammons, Rastislav Bod´ik, James R. Larus "Mining Specifications", 2002.