An Efficient Methodology for Improving Code Quality Using Object oriented approach

Annie Ratna Priya, M. Mythily,
Department of Computer Science and Engineering,
Karunya University, Coimbatore, India

Abstract—

Most of the software development activities require formal specification which can be used as a functional support for finding and fixing the bugs in deployed software. However such specifications are hard to verify since it suffers from high false positive rates. In order to solve this problem a suite of metrics is incorporated along with the specification mining system. The efficiency of the code measured using these metrics. This paper focuses on giving an overview of metrics that are already used in specification mining techniques and thereby stating the need for understanding the object oriented approach and its functionality in molding formal specifications which has been recently concentrated during the maintenance of the system.

***Index Terms*—Metrics, Object, Specification, Temporal property**

I. Introduction

Developing software and maintenance plays central role in the delivery and application of the information technology field, and hence developers are increasingly focusing on process improvement in the software development area. Throughout the software development process, from requirement elicitation to system update the metrics are needed to measure the activities of the software with different environment at each time. Metrics will give a better understanding of the whole life cycle process for people concerned with the different tasks in it [1]. When the input is given to the system, it will analyze the structural and logic components so that it will be helpful for the metrics computation. A relevant study is helpful to investigate on the different types of metrics. Most of the metrics are realized based on procedural and object oriented software. The object-oriented (OO) approach to software development promises better management of system complexity and a likely improvement in project outcomes such as quality and project cycle time.

Generally in object oriented software development process, the system is viewed as collection of objects. The entire working of the system is notified by the relationship among these objects. Whenever, one object depends on another object to do certain functionality, there is a relationship between those two classes. In order to achieve perfect uniqueness of object, objects should rely on the interfaces and support offered by another object without relying on any underlying implementation factors. For example, in case of abstraction, a correct level of abstraction helps build a flexible and scalable application. It is not an easy job to reach the correct level of abstraction and the correct relationship between classes. The possible defects concerning this should be detected as early as possible such as in the design phase so that all the further processes can be corrected without bugs.

In specification mining techniques, the formal specifications have faced difficulty in both verification and validation. Initially most of the automatic mining techniques automatically trace specifications from the program code.

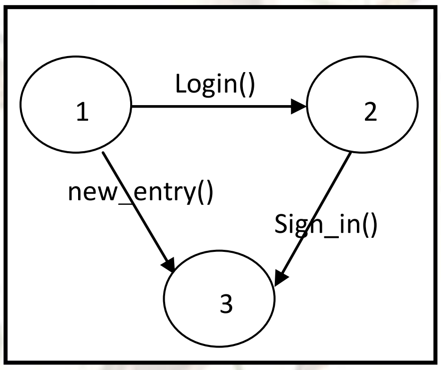

Figure 1. Specification Example

These techniques takes program as input and produce candidate specification as output in the form of finite state machine that will later describes the program correctness. But most of the techniques are imprecise in practice. Initially the decision on choosing the correct behavior itself is inappropriate. A good code is actually a code that works and is free of bugs, and is readable and maintainable [3]. Organizations usually have coding standards which all developers will follow, but every programmer and software engineer has different ideas on what suites best for the efficient functioning of the system.

The code can be formulated by imposing rules on the working of the system. But it is needed to be kept in mind that excessive use of rules can be a barrier for both productivity and creativity. Specifications describe how to manipulate important program resources [4]. They are represented as a finite state machine that encodes valid sequence of events. A program execution adheres to a given specification if and only if it terminates with the corresponding state machine in an accepting state where the machine starts in its start state at program initialization. Otherwise, the program violates the specification and contains an error.

II. Mining characteristics

This section shows the underlying concepts of mining techniques and their limitations which encourage the researchers to step into incorporating code quality metrics.

Specification mining techniques produces specifications but still they have high false positive rates. The Comparison between most of these approaches is provided in the Table 1.

In WN miner [2] the specification mining was motivated by the observations of run-time error handling mistakes. In other approaches examining such mistakes, the code frequently violates simple API specifications in exceptional situations. Despite the proliferation of specification-mining research, there is not much report on issues pertaining to the quality of specification miners. This technique is same as that of Engler et al. but is based on assumptions about run time errors, chooses candidate event pairs differently, presents significantly fewer candidate specifications and ranks presented candidates differently. In a normal execution, events '*a*' and '*b*' may be separated by other events and difficult to discern as a pair. After an error has occurred, however, the cleanup code is usually much less cluttered and contains only operations required for correctness. The candidate specifications are filtered using varied criteria such as exceptional control flow, one error, data path etc.

This highlights the practical importance of the algorithmic assumptions, in particular the use of exceptional control flow. It can serve as a requirement for acceptance. It can even assist inspections by helping to target effort at parts of a program that may need improvement. Though this miner select specifications from software artifacts and finds per-program specifications for error detection, it does not have profound results in bug finding.

Strauss, ECC and WN technique were all good at yielding specifications that found bugs. The WN technique found all bugs reported by other techniques on these benchmarks and did so with the fewest false positives.

Table1. A Comparison study

Miner Used	Features	Remarks
Engler et al.	Use two state temporal properties.	High false positive rates
Whaley et al.	Produces Single multi state specification.	Human intervention
Strauss	Mainly focused on machine learning to learn a Single specification from traces	Use of single specification is not sufficient
JIST	Refines Whaley et al. technique to mainly disregard infeasible paths	Handles only simple subset of Java
WN miner	Selecting specifications from software artifacts	Does not have profound results in finding bugs
Claire approach	Use measurements of trustworthiness of source code to mine specifications	Does not give adequate results over precision.
SMArTC	Provides mining architecture to improve performance	Scalable metrics is at least level

III. Application of object oriented metrics in specification mining

Incorporating the code quality metrics as a model into the existing specification miner is shown as a flow chart diagram in Fig 2. While finding the measurements the additional parameters are also extracted to get a thorough impact over the quality of the code.

After evaluating the relative importance of the quality metrics, the correlation between the metrics is calculated. These metrics implicitly take advantage of previous testing and validation work which should be done manually.

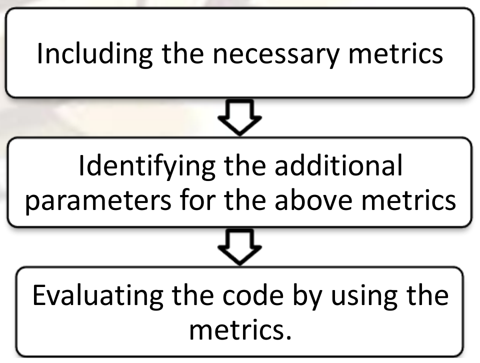
Including the necessary metrics
Identifying the additional parameters for the above metrics
Evaluating the code by using the metrics.

Figure2. Flowchart diagram for metrics evaluation

Object oriented metrics is based mainly on three factors: inheritance, coupling and methods. This can be calculated by calculating the weighted methods per class, number of modules, and response for a class and so on. The quality metrics are the quantitative measures of the degree to which software possesses a given attribute that affects its quality.

Code metrics like LOC and Cyclomatic Complexity examines the internal complexity of a procedure whereas this structure metrics examines the relationship between a section of code and the rest of the system. Process oriented metrics are used through the different phases of the software life cycle. Measurement on quality should concentrate on the early phases in the life cycle to improve the quality of software and decrease of development and maintenance costs.

Defects must be tracked to the release origin which is the portion of the code that contains the defects and at what release the portion was added, changed, or enhanced. When calculating the defect rate of the entire product, all defects are used; when calculating the defect rate for the new and changed code, only defects of the release origin of the new and changed code are included. On the one hand, the process quality metrics simply means tracking defect arrival during formal machine testing for some organizations. On the other hand, some software organizations with well-established software metrics programs cover various parameters in each phase of the development cycle.

The specification language formulation is done by deriving a set of rules which should be followed by the program code which is meant to be the specifications of the particular program behavior. These rules describe conditions that a program must satisfy for it to be considered a valid code. A rule may be thought of as a condition that evaluates a portion of a program to a truth value. These rules are certainly affected by changes to their context elements. However, it is important to understand that rules are also affected by many other model elements not explicitly identified. Since these rules are conditions on a program code, their truth values change only if the application changes or if the condition changes, but this is explored elsewhere.

IV. Trending Metrics

The development of a large software system is a time and resource consuming activity. Even with the increasing automation of software development activities, resources are still scarce. Therefore, it is necessary to be able to provide accurate information and guidelines to managers to help them make decisions, plan and schedule activities, and allocate resources for the different software activities that take place during software development. Software metrics are thereby necessary to identify where the resources are needed. They are a crucial source of information mostly for decision making.

In view of the above approaches a new mechanism can be derived to generate specifications with object oriented metrics since most of the languages generally follow object oriented methodology. The specifications that are generated can be used to check the proper functioning of the system to refer to its flow of execution.

V. Conclusion

This paper clearly illustrates the overview of need for accuracy in specification mining techniques, the possible approaches of deriving specifications at different domains which lists the effectiveness on this mapping to new scope in specifications. The goal of this survey is to support the study on the legacy of generating specifications to the new automatic techniques. It helps to get an insight into this dynamic field of study in Specification Mining. Since the object orientation is emerging in all kinds of applications, it is also welcome in the specification mining process. It is mentioned to be dynamic, as these approaches are under development and it steps higher everyday to achieve efficiency in capturing specifications.

REFERENCES

[1] Claire Le Goues and Westley Weimer, “Measuring Code Quality to Improve Specification Mining”, *IEEE Transactions on Software Engineering*, Vol. 38, No 1 ,Jan/Feb 2012.

[2] W. Weimer and G.C. Necula, “Mining Temporal Specifications for Error Detection,” Proc. Int’l Conf. Tools and Algorithms for the Construction and Analysis of Systems, pp. 461-476, 2005.

[3] D. Engler, B. Chelf, A. Chou, and S. Hallem, “Checking System Rules Using System-Specific, Programmer-Written Compiler Extensions,” Proc. Conf. Symp. Operating System Design and Implementation, 2000.

[4] D.R. Engler, D.Y. Chen, and A. Chou, “Bugs as Inconsistent Behavior: A General Approach to Inferring Errors in Systems Code,” Proc. Symp. Operating System Principles, pp. 57-72, 2001.

[5] G. Ammons, R. Bodik, and J.R. Larus, “Mining Specifications,” Proc. ACM SIGPLAN-SIGACT Symp. Principles of Programming Languages, pp. 4-16, 2002.