ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 9, September 2015

Measuring Code Quality to Improve Specification Mining

Roshan Omprakash Deshmukh, Prof. Umesh Kulkarni

M.E. Student, Dept. of Computer Engineering, ARMIET, Shahapur, India

Assistant Professor, Dept. of CMPN, VIT, Wadala, India

ABSTRACT: Every software Industry requires the quality of code. Formal specifications are mathematically based techniques whose purposes are to help with the implementation of systems and software. They are used to describe a system, to analyze its behavior, and to aid in its design by verifying key properties of interest through rigorous and effective reasoning tools. These specifications are formal in the sense that they have syntax, their semantics fall within one domain, and they are able to be used to infer useful information.

Measuring Code Quality to Improve Specification Mining is used to create a set of design principles for code modularization and produce set of metrics that characterize software in relation to those principles. Some metrics are structural, architectural, and notions. The structural metrics refer to inter module-coupling based notions. The architectural metrics refer the horizontal layering of modules in large software systems. Here we are using three types of contributions coupling, cohesion, and complexity of metrics to modularize the software.

These contributions measure were primarily at the level of how the individual classes were designed from the standpoint of how many methods were packed into the classes, the depth of the inheritance tree, the inheritance fan-out, coupling between objects created by one object invoking a method on another object.

Other contributions that have also used function call dependencies to characterize software modularization. Modularization algorithm is based on the combination of coupling and cohesion metrics. This is used to find modularization quality.

KEYWORDS: Specification mining, machine learning, software modularization, code metrics, program understanding

I. INTRODUCTION

Incorrect and buggy behavior in deployed software costs up to $70 billion each year in the US [7]. Thus debugging, testing, maintaining, optimizing, refactoring, and documenting software, while time-consuming, remain critically important.

Such maintenance is reported to consume up to 90% of the total cost of software projects.A key maintenance concern is incomplete documentation up to 60% of maintenance time is spent studying existing software. Human processes and especially tool support for finding and fixing errors in deployed software often require formal specifications of correct program behavior; it is difficult to repair a coding error without a clear notion of what "correct" program behavior entails. Unfortunately,

while low-level program annotations are becoming more and more prevalent, comprehensive formal specifications remain rare.

Many large, preexisting software projects are not yet formally specified. Formal program specifications are difficult for humans to construct .and incorrect specifications are difficult for humans to debug and modify. Accordingly, researchers have developed techniques to automatically infer specifications from program source code or execution traces [2]. These techniques typically produce specifications in the form of finite state machines that describe legal sequences of program behaviors.

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 9, September 2015

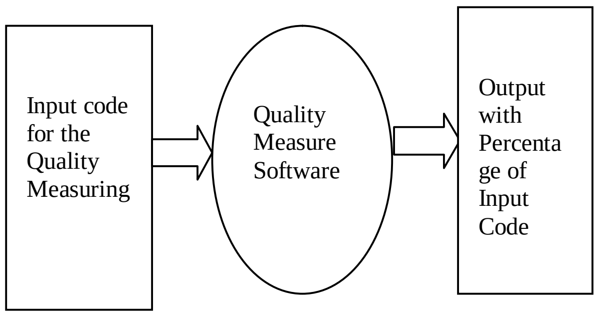

Input code for the Quality Measuring
Quality Measure Software
Output with Percentage of Input Code

Fig.1.Block Diagram

A.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 [6] 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 Table1. A Comparison study 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.

B.QUALITY METRICS:

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.

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.

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.

II. LITERATURE SURVEY

Whaley *et al.* propose a static miner [1] that produces a single multi-state specification for library code. The JIST[2] miner refines Whaley *et al.*'s static approach by using techniques from software model checking to rule out infeasible paths. Gabel and Su [3] extend Perracotta using BDDs, and show both that two-state mining is NP-complete and some specifications cannot be created by composing two-state specifications. Lo *et al.* use learned temporal properties, such

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 9, September 2015

as those mined in this article, to *steer* the learning of finite state machine behavior models [4].Shoham *et al.*[5] mine by using abstract interpretation, where the abstract values are specifications

III. PROPOSED WORK:

Proposed system is developed using Object oriented software system. Create a set design principles for code modularization and produce set of metrics. Modularization quality is calculating using metrics such as structural, architectural and notions.
There are three contributions such as coupling, cohesion and complexity metrics to modularize the software. Our proposed metrics seek to characterize a body of software according to the enunciated principles. Provide two types of experiments to validate the metrics

FEATURES OF PROPOSED SYSTEM

The proposed system is having the following features

*Easy to identify the project quality.
*The quality is based on the metrics such as structural, architectural and notion.
*Reorganization is not difficult.
*The total number of line, function call, and module are easy to calculate.

A. IMPLEMENTATION:

MODULES

Solution strategy defines the way being used for solving the problem. The project is having four major modules:

*Source Code Import And Partition
*Module Count and Function Call Calculation
*Metric Calculation
*Report Generation

Source Code Import & Partition

User or tester will import file/project to our tool. The tool will partition the source code by its self.

Module Count & function Call Calculation

In this module the tool will find the size/total number of lines in the project. After that calculate what are the functions/methods are involved in this project. How many methods call from other modules, how many modules call other modules and what are all the functions from other module and find how many classes and modules in a given file.

Metrics Validation

This module is heart of our project. Here we are going to calculate the quality of software based on the modules, function and size. There are three types of metrics used to calculate the quality of software. Each metric is given various output/result. Using these outputs we can draw a graph. Finally the graph will denote the quality.

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 9, September 2015

Report Generation

In this module we are going to generate a report for our testing result. Using this report we give some suggestion to developer. There are two type reports available. One is normal report another one is graph report.

B.ANTICIPATED ADVANTAGES :

*Proposed system is used for both object and non object oriented software system.
*Well efficient system.
*Reorganization is easy.
*Less time consumption.
*It test for object oriented and non object oriented software system
*Three types of metrics is used to identify the project accuracy.

IV. **RESULTS**

Fig.2 shows the log-in page,upon log-in user is redirected to the registration page where user is requested to register (Fig.3) the project details of a client for future reference,which includes project code,platform,booking date,delivery date, in-charge name.

Once the user is registered then they will be redirected to directory path (Fig.4) to analyze the path frequency & path density.Next step is of code analyzation.Once the code analysis is done,the metric count & metric calculations are done.After metric calculation,report generation is done as shown in fig.5.

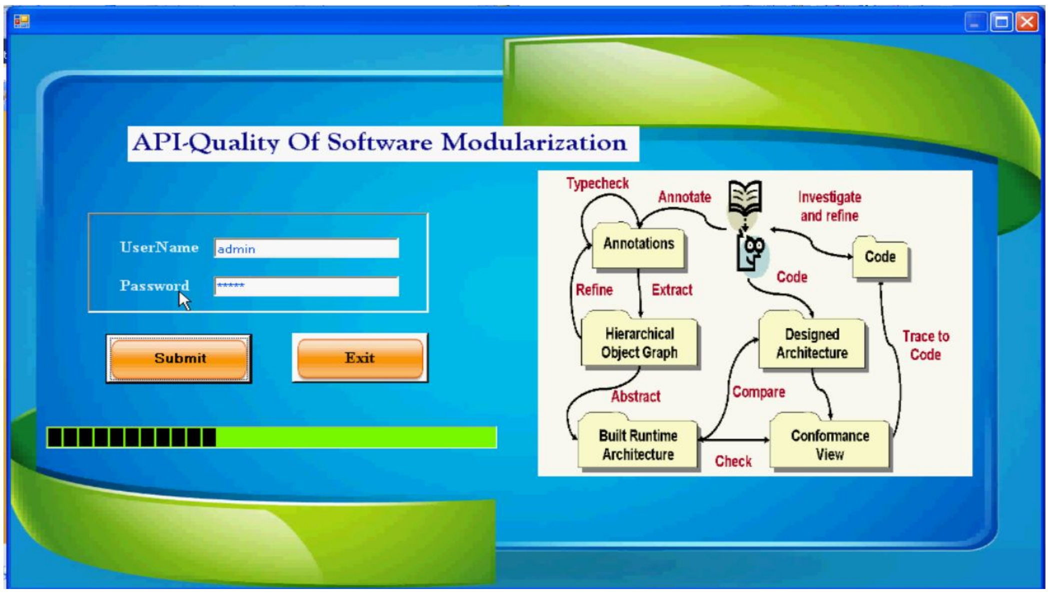


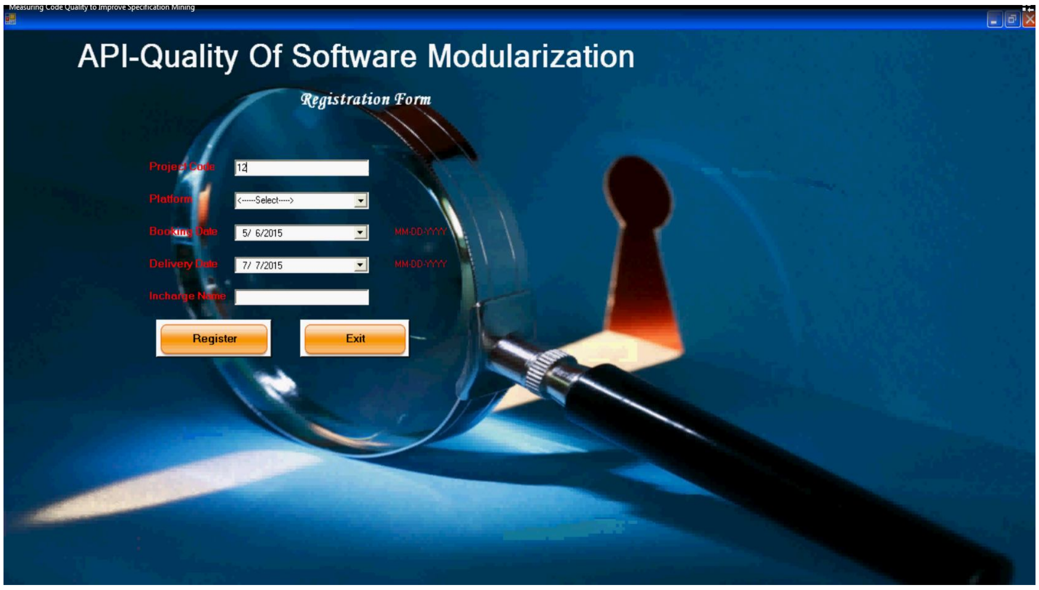
Fig.2. Log-in page

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 9, September 2015

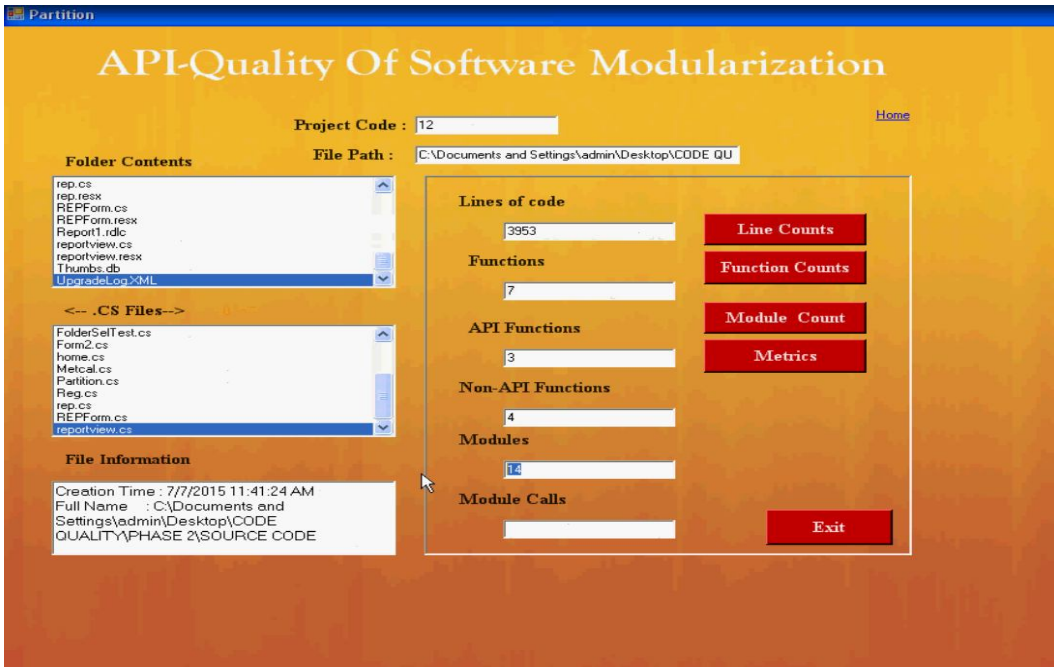

Measuring Code Quality to Improve Specification Mining

API-Quality Of Software Modularization

Registration Form

Project Code: 12
Platform: <-----Select----->
Booking Date: 5/ 6/2015 MM-DD-YYYY
Delivery Date: 7/ 7/2015 MM-DD-YYYY
Incharge Name:

Fig.3.Registration Form


Partition

API-Quality Of Software Modularization

Project Code : 12
File Path : C:\Documents and Settings\admin\Desktop\CODE QU

Folder Contents

```
rep.cs
rep.resx
REPForm.cs
REPForm.resx
Report1.rdlc
reportview.cs
reportview.resx
Thumbs.db
UpgradeLog.XML
```

<-- .CS Files-->

```
FolderSelTest.cs
Form2.cs
home.cs
Metcal.cs
Partition.cs
Reg.cs
rep.cs
REPForm.cs
reportview.cs
```

File Information

Creation Time : 7/7/2015 11:41:24 AM
Full Name : C:\Documents and Settings\admin\Desktop\CODE QUALITY\PHASE 2\SOURCE CODE

Lines of code: 3953
Functions: 7
API Functions: 3
Non-API Functions: 4
Modules: 14
Module Calls:

Fig.4. Code Selection

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 9, September 2015

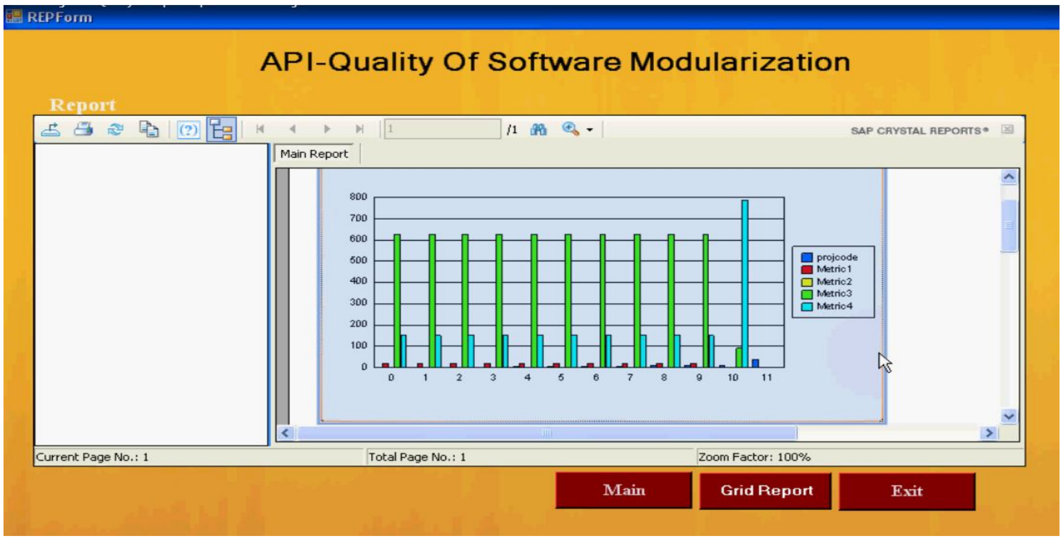


Fig.5. Report Generation

V. CONCLUSION

We have enunciated a set of design principles for code modularization and proposed a set of metrics that characterize software in relation to those principles. Although many of the principles carry intuitive plausibility, several of them are supported by the research literature published to date. Our proposed metrics seek to characterize a body of software according to the enunciated principles. The structural metrics are driven by the notion of API—a notion central to modern software development. Other metrics based on notions such as size-boundedness, size-uniformity, operational efficiency in layered architectures, and similarity of purpose play important supporting roles. These supporting metrics are essential since otherwise it would be possible to declare a malformed software system as being well-modularized. As an extreme case in point, putting all of the code in a single module would yield high values for some of the API-based metrics, since the modularization achieved would be functionally correct. We reported on two types of experiments to validate the metrics. In one type, we applied the metrics to two different versions of the same software system. Our experiments confirmed that our metrics were able to detect the improvement in modularization in keeping with the opinions expressed in the literature as to which version is considered to be better.

The other type of experimental validation consisted of randomizing a well-modularized body of software and seeing how the value of the metrics changed. This randomization very roughly simulated what sometimes can happen to a large industrial software system as new feature are added to it and as it evolves to meet the changing hardware requirements. For these experiments, we chose open-source software systems. For these systems, we took for modularization the directory structures created by the developers of the software. It was interesting to see how the changes in the values of the metrics confirmed this process of code disorganization. Theoretical validation implies conformance to a set of agreed-upon principles that are usually stated in the form of a theoretical framework.

REFERENCES

1. J. Whaley, M. C. Martin, and M. S. Lam, “Automati extraction of object-oriented component interfaces,” in *ISSTA*, 2002.
2. R. Alur, P. Cerny, P. Madhusudan, and W. Nam, “Synthesis ofinterface specifications for Java classes,” in *POPL*, 2005.
3. M. Gabel and Z. Su, “Symbolic mining of temporal specifications,”in *ICSE*, 2008, pp. 51–60.
4. D. Lo, L. Mariani, and M. Pezz`e, “Automatic Steering of Behavioral Model Inference,” in *FSE*. ACM, 2009, pp. 345–354.
5. S. Shoham, E. Yahav, S. Fink, and M. Pistoia, “Static specification mining using automata-based abstractions,” in *International Symposium on Software Testing and Analysis*, 2007, pp. 174–184.
6. W. Weimer and G.C. Necula, “Mining Temporal Specifications for Error Detection,” Proc. Int‟l Conf. Tools and Algorithms for the
7. Construction and Analysis of Systems, pp. 461-476, 2005.

National Institute of Standards and Technology, “The economicimpacts of inadequate infrastructure for software testing,” Tech.Rep. 02-3, May 2002.