VOL. 2, NO. 11, October 2011 ISSN 2079-8407
Journal of Emerging Trends in Computing and Information Sciences
©2009-2011 CIS Journal. All rights reserved.

http://www.cisjournal.org

A Study of Mining Software Engineering Data and Software Testing

T.Murali Krishna[1], Devara Vasumathi[2]

1. *Lecturer, Department of Computer Science, College of Engineering & Technology, Jimma University, Jimma, Ethiopia*
2. *Associate. Professor, Department of Computer Science &Engineering, College of Engineering & Technology, Jawaharlal Nehru Technological University, Hyderabad, India.*

{murali2007tel@yahoo.com[1], vasukumar_devara@yahoo.co.in[2] }

ABSTRACT

The primary goal of software development is to deliver Optimal Software, i.e., software produced at low cost, high quality & productivity and scheduled with in time. In order to achieve this optimal software, programmers generally reuse the existing libraries, rather than developing similar code products right from the scratch. While reusing the libraries, programmers are facing several changes such as many existing libraries are not properly documented and many libraries contain large number of program interfaces (PIs) through which libraries expose their functionality.
These challenges lead to certain problems that affect in producing optimal software. The problems such as reuse of existing libraries consumes more time, lack of knowledge on reusage of program interfaces and we can't generate effective test inputs during white box testing. The first two problems reduce the software productivity where as last one affect on software testing.
To resolve these problems, we propose a general framework called Netminer. Netminer contains a code search engine. With the help of code search engine, we can search the available open source code over the internet. In the analysis phase, Netminer automatically compares the specifications of program interfaces with relevant code examples that are available in the internet. In the next phase, Netminer applies data mining techniques on code examples that are collected and identify common patterns. The common patterns represent exact usage of program interfaces.
We propose some more approaches based on Netminer. Some approaches help programmers in effectively reusing program interfaces provided by existing libraries. Some approaches identify defects under analysis from the mined specifications and some approaches help in generating test inputs by the use of static and dynamic test generation. Our research study shows that Netminer framework can be effectively used in software engineering for achieving optimal software.

Keywords: *Software Engineering, Data Mining, Program Interface, Netminer, Algorithms.*

1. INTRODUCTION

What affects software productivity and how do we improve it? This is a concern near and dear to those who are responsible for researching and developing large software system. The main aim of software development is to produce optimal software efficiently and effectively. In order to attain the optimal software, programmers reuse the existing libraries, rather than developing similar code from the scratch. These libraries include open source libraries such as Eclipse or C#. From 1995 onwards, there is a rapid growth in not only open source libraries but also in reuse of these open source libraries.

It is observed from earlier researches, that more than 40% of source files among the projects under analysis include the code from open source libraries. A new programming methodology by reusing libraries is called Opportunistic Software Systems Development (OSSD). Using OSSD, programmers develop systems from readymade components by combining each other. Rather than developing similar code right from the beginning, OSSD reuses the existing libraries. Reuse of existing libraries helps in reducing effort during software maintenance and also increased software productivity.

For the reuse of libraries, we considered Object Oriented libraries where inheritance plays a vital role. The functionality of Object Oriented Libraries handled through an interface called Program Interface (PI). In object oriented languages PI is used to represent a set of classes and methods provided by libraries. For effective reusing of existing libraries, programmers need the knowledge of how to use PIs. The following two challenges are faced by programmers in understanding the usage of PIs.

First, large number of existing libraries contains many PIs. For e.g., C# library provides around 9000 PI classes. These classes provide different functionalities such as Stack, Queue, and Linked List etc. Out of all these PIs some are important and some are more important compared to others. Programmers should have proper knowledge about libraries. Otherwise they will face problems such as from where and how to start using library.

Second, many existing libraries are outdated and or not properly documented. For example, programmer reusing Eclipse library. Programming task is to write code and convert into parse code in an editor.

e.g. IEditorPart obj1 = …….

IEditorInput obj2 = obj1.getEditorInput ();

IWorkingCopyManager wcm = JavaUI. GetWorkingCopyManager ();

ICompilationUnit cu = wcm.getWorkingCopy (obj2);

A programmer unfamiliar to Eclipse may take long time in reusing the existing libraries.

2. PROBLEM STATEMENT

Because of the above two factors , while in reusing the existing libraries programmers face three major problems that affect in producing optimal software.

While using PIs, programmers introduce defects due to lack of knowledge on how to reuse PIs. The main reason for such defects is programmers to follow implicit programming rules. For e.g., both *next* and *has next* methods form implicit programming in Java utility package.

Proper documentation and sufficient examples play a vital role in understanding PIs. If programmers spend more time in understanding PIs, software productivity will be reduced which in turn affects producing optimal software.

While reusing libraries, programmers face challenges in generating test inputs for the client code during white box generation technique. Test inputs need method sequences and exercise the code under test. These sequences go for branch testing that covers True or False. In general existing libraries contain sequences, which include multiple classes. Hence it's a challenging practice to programmers to automatically generate method sequences with multiple classes.

3.

3. TECHNIQUES FOR SOLVING THE PROBLEM

In order to solve above problems and to produce optimal software, we propose a general frame work called Netminer. This Netminer contain new techniques such as information retrieval, program analysis, and software testing. Netminer helps programmer by giving open source code on the internet in reusing existing PIs. This is done in analysis phase. Netminer then applies data mining techniques on the relevant code examples that are collected known as "Mining Software Engineering Data" (MSED).

Fig. 1: Phase 1 of Netminer

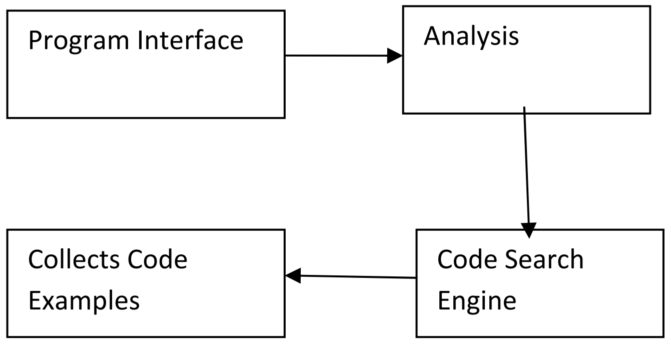
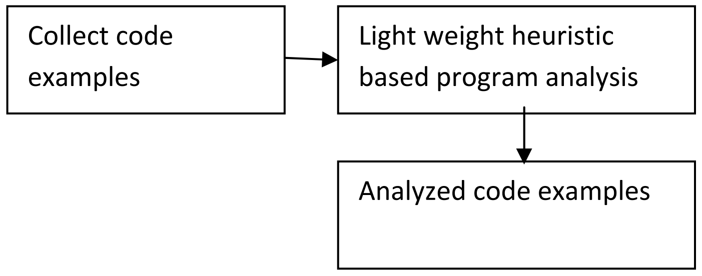
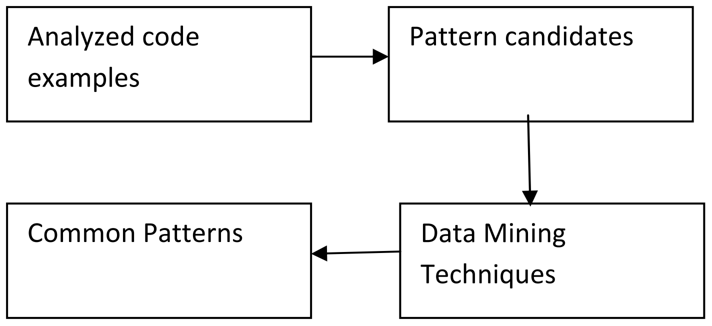


Fig. 2: Phase 2 of Netminer



With the help of light weight analysis we can handle large number of code examples.

Fig. 3: Phase 3 of Netminer



These common patterns represent likely uses of PIs among all pattern candidates. In the last phase, Netminer uses mined PI specifications for getting optimal software. Netminer helps programmer in understanding how to use PIs by suggesting related PI specifications.

In order to develop optimal software, Netminer uses the approaches such as PARSE web, spot web, static verification and dynamic test generation.

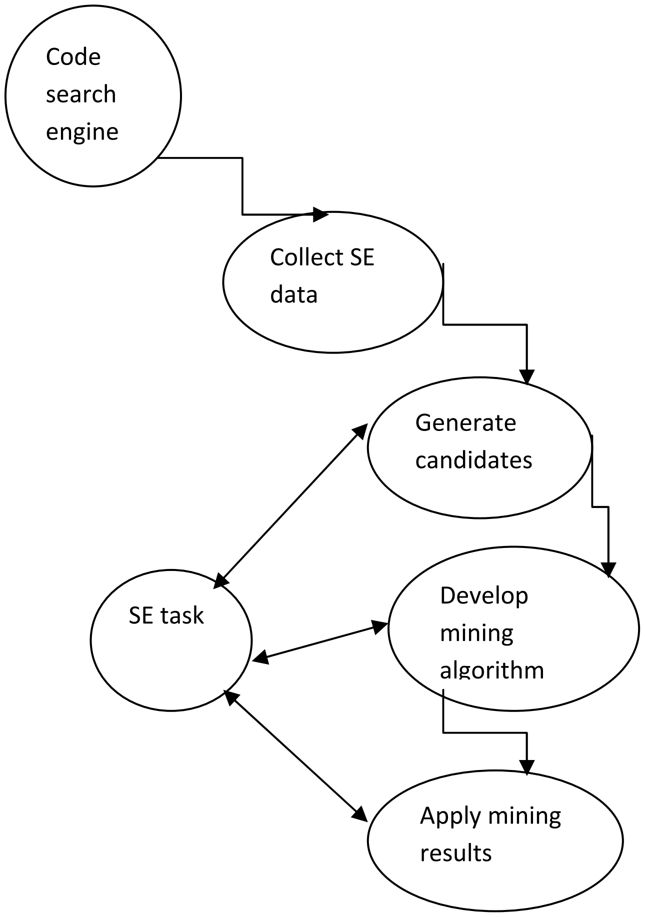


Fig. 4: FOUR PHASES OF NETMINER FRAMEWORK

3.1 APPROACHES FOR BETTER PRODUCTIVITY

To help programmers in effectively reusing existing libraries, we developed two approaches, called PARSEWeb and SpotWeb based on NetMiner framework. These two approaches will help us to increase programmer's productivity.

PARSEWeb: It accepts queries in the form of "source→destination" and mines frequent method sequences that accept an object of source type and produce an object of destination type. While programmers are writing code using existing libraries, the above two method sequences are helpful to them.

Eg Consider a programmer, who uses open source implementation of Java Message Service PI1.1 specification. Programmer has an object of type **QueueConnectionFactory** and does not know how to write code to get a **QueueSender** object, which required for sending messages. PARSEWeb approach helps the programmer in the following manner. The programmer first translates the problem into a "**QueueConnectionFactory → QueueSender".**

Methods Sequences suggested by PARSEWeb

01:FileName: O_UserBean.java MethodName: ingest

02:QueueConnectionFactory,createQueueConnection() Returntype: QueueConnection

03:QueueConnection,createQueueSession(Boolean,Session.AUTO_ACKNOWLEDGE) Returntype: QueueSession

04:QueueSession,CreateSender(Queue) Returntype:QueueSender

Equivalent Java code for method sequence suggested by PARSEWeb

01:QueueConnectionFactory objqcf;

02:QueueConnection objqucon = objqcf.createQueueConnection();

03:QueueSession objqs = objqucon.createQueueSession(true,Session,AUTO_ACKNOWLEDGE);

04:QueueSender objqusen = objqs.createSender(new Queue());

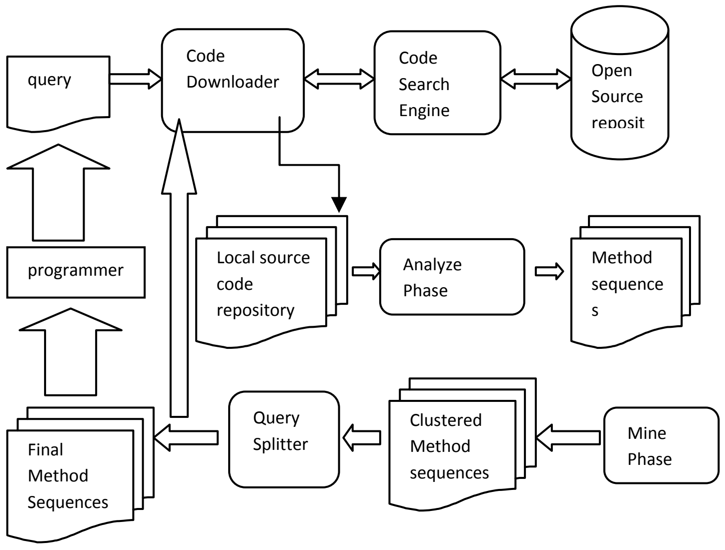


Fig. 5: OVERVIEW OF PARSEWeb Approach

SpotWeb: It helps programmers in reusing PI classes and methods of an existing library by detecting coldspots and hotspots of the library. Coldspots are PI classes and methods that are very rarely used. Hotspots are PI classes and methods that are frequently reused. They are also used as starting points for programmers in reusing and understanding the library.

In general, users reuse certain areas of libraries which are flexible. For effective reuse of libraries, programmers must be aware of these flexible areas, which are known as hotspots. Hotspots are built upon Open-Closed principle. The "Open" parts(also known as hooks) represent areas that are variant and flexible. The "Closed" parts (known as templates) represent areas that are immutable in the given library. A hotspot is a combination of both templates and hooks.

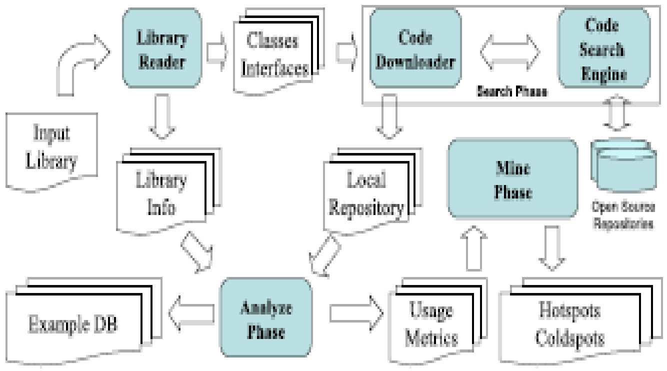


Fig. 6: OVERVIEW OF SPOTWEB

Both these approaches are based on NetMiner framework that collects code examples on demand. These two approaches are independent of any specific set of libraries.

3.2 APPROACHES FOR BETTER QUALITY THROUGH STATIC VERIFICATION

To mine PI patterns and detect defects in client code from mined patterns, we use two approaches called Alattin and CAR-Miner. Alattin focuses on reducing false positives among detected violations, where as CAR-Miner focuses on reducing false negatives by detecting new defects. False positives indicate those violations that do not represent real defects. False negatives represent the defects that exist in applications under analysis.

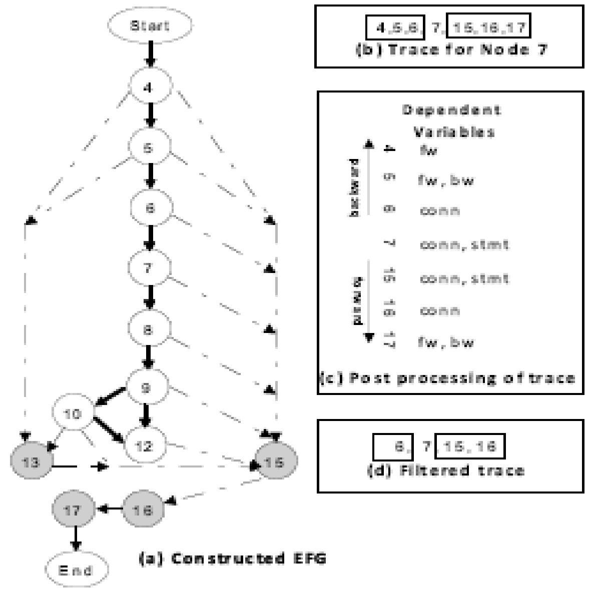


Fig. 7: EXAMPLE OF CAR-MINER APPROACH

CAR-Miner accepts an application under analysis and mines exception-handling rules for all function calls. CAR-Miner next detects violations of the mined exception-handling rules.

3.3 APPROACHES FOR BETTER QUALITY THROUGH DYNAMIC TEST GENERATION

To assure high quality of developed software, we adopt unit testing, which helps to detect defects at an early stage. For effective generation of test inputs that achieve high structural coverage of the code under test, we use two approaches called MSeqGen and DyGen that mine static and dynamic traces respectively. MSeqGen statically extracts method sequences from existing code bases and assists random & dynamic symbolic execution. DyGen automatically generates regression tests from dynamic traces.

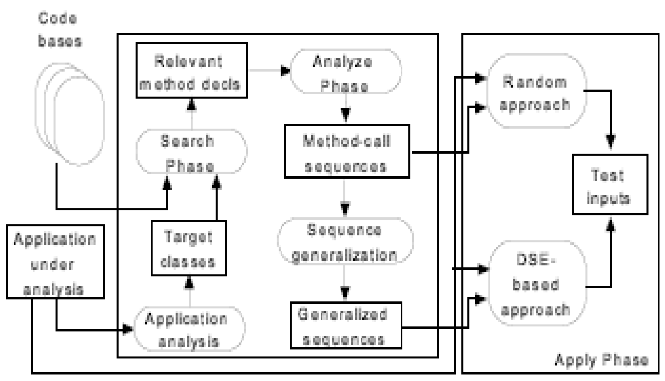


Fig. 8: OVERVIEW OF MSEQGEN APPROACH

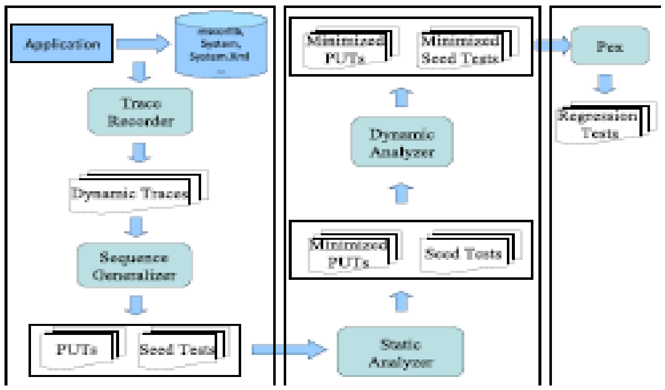


Fig. 9: HIGH-LEVEL OVERVIEW OF DYGEN

4. FUTURE WORK

In our research work we used the general frame work Netminer that combines code searching and mining to achieve Software Engineering problems such as detecting defects under analysis.

We focused on mining source code which is in the form of Structured SE data. With in the internet, there may be other form of SE data which is in Natural Language (e.g. BCEL and other technical blogs) called Unstructured SE data. Hence in future, we plan to develop new frame work Netminer ++, that can influence both structured and unstructured SE information available on the internet. Netminer ++ includes additional techniques and approaches based on Natural Language Processing (NLP)

In future, we mainly concentrate on various aspects such as what kind of unstructured SE data are available on the internet, the analyzation & understanding the unstructured SE data, Indexing the analyzed data and search for relevant data in unstructured SE data.

The present research mainly focuses on mining source code syntactically. In future, we can extend from syntactic code analysis to semantic analysis.

For getting optimal software, we used static verification and dynamic test generation in the present research. In future, even we can integrate both methodologies for better results.

5. CONCLUSION

Mining software engineering data (MSED) mainly applies data mining techniques on SE artifacts. In this research work, we advanced by expanding the data scope to large amount of source code available on the internet, using Netminer. The effectiveness of Netminer explained by various approaches such as PARSE web, spot web, MSeqGen, Dygen etc. Some can helpful in developing optimal software where as some assist in test generation.

ACKNOWLEDGEMENT

We thank Dr. Balaji Dhanasekaran for giving valuable suggestions and comments. We also thank Dr.Senthil and Mr. Niraj Kumar Rai, IISC, Banglore for interesting discussions. We also thank Jimma University for providing us not only financial support but also for providing all the reference publications, manuals and journals.

REFERENCES

[1] Mithun Acharya. *Mining API Specifications from Source Code for Software Reliability*. PhD thesis, North Carolina State University, 2009.

[2] Eclipse, 2010. http://www.eclipse.org/.

[3]Eclipse developer forum, 2010. http://www.eclipse.org/forums/

[4] J. Hammond. What developers think, 2010. http://www.drdobbs.com/architect/222301141/

[5] Parastoo Mohagheghi, Reidar Conradi, Ole M. Killi, and Henrik Schwarz. *An empirical study of software reuse vs. defect-density and stability.* In Proceedings of the 26th International Conference on Software Engineering (ICSE), pages 282-292, 2004.

[6] Cornelius Ncube, Patricia Oberndorf, and Anatol W. Kark. *Opportunistic software systems development: Making systems from what's available*. Software, IEEE, 25(6):38–41, Nov. 2008.

[7] Martin P. Robillard. *What makes APIs hard to learn?* Answers from developers. IEEE Software, 26(6):26–34, 2009.

[8] Westley Weimer and George Necula. *Mining temporal specifications for error detection.* In Proceedings of 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), pages 461–476, 2005.

[9] *Principles of Data Mining* - D.J.Hand, Heikki Mannila, Padhraic Smyth

[10] J.D. Anjou, S. Fairbrother, D. Kehn, J. Kellerman, and P. McCarthy. The Java Developer's Guide to Eclipse. Addison-Wesley Professional, 2004.

[11] Google Code Search Engine, 2006. http://www.google.com/codesearch.

[12] Viljamaa Jukka. Reverse engineering framework reuse interfaces. In Proceedings of the 9th European Software Engineering Conference held jointly with 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE), pages 217–226, 2003.

[13] JUnit, 2001. http://www.junit.org.

[14] Timothy C. Lethbridge, Janice Singer, and Andrew Forward. How software engineers use documentation: The state of the practice. In IEEE Software, pages 35–39, 2003.

[15] Wolfgang Pree. Meta patterns - a means for capturing the essentials of reusable object oriented design. In Proceedings of the 8th European Conference on Object-Oriented Programming (ECOOP), pages 150–162, 1994.

[16] Naiyana Sahavechaphan and Kajal Claypool. XSnippet: Mining for sample code. In ACM SIGPLAN symposium on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA), pages 413–430,

[17] Nikolai Tillmann and Jonathan de Halleux. Pex white box test generation for .NET. In Proceedings of the 2nd International Conference on Tests and Proofs (TAP), pages

134–153, 2008.2006.

[18] Jianyong Wang and Jiawei Han. BIDE: Efficient mining of frequent closed sequences. In Proceedings of 20th International Conference on Data Engineering (ICDE), pages 79 – 88, 2004.

[19] http://java.sun.com/products/jms