International Journal of Data Mining Techniques and Applications
Volume: 01 Issue: 02 December 2012 Page No.36-39
ISSN: 2278-2419

Improvement of Software Maintenance and Reliability using Data Mining Techniques

Yethiraj N G

Assistant Professor, Department of Computer Science
Maharani's Science College for Women, Bangalore, India

.

Abstract-Software is ubiquitous in our daily life. It brings us great convenience and a big headache about software reliability as well: Software is never bug-free, and software bugs keep incurring monetary loss of even catastrophes. In the pursuit of better reliability, software engineering researchers found that huge amount of data in various forms can be collected from software systems, and these data, when properly analyzed, can help improve software reliability. Unfortunately, the huge volume of complex data renders the analysis of simple techniques incompetent; consequently, studies have been resorting to data mining for more effective analysis. In the past few years, we have witnessed many studies on mining for software reliability reported in data mining as well as software engineering forums. These studies either develop new or apply existing data mining techniques to tackle reliability problems from different angles. In order to keep data mining researchers abreast of the latest development in this growing research area, we propose this paper on data mining for software reliability. In this paper, we will present a comprehensive overview of this area, examine representative studies, and lay out challenges to data mining researchers.

Key words- Software, Software Reliability, Data Mining, Frequent Item Set, Extracting Rules.

I. INTRODUCTION

The economies of all developed nations are dependent on software. More and More systems are software controlled. Software Engineering is concerned with theories, methods and tools for professional software development. Software Engineering is an engineering discipline which is concerned with all aspects of software production. Software Engineers should adopt a systematic and organized approach to their work and use appropriate tools and techniques depending on the problem to be solved, the development constraints and the resources available. Software reliability, unlike many other quality factors, can be measured directed and estimated using historical and developmental data [1]. *Software reliability* is defined in statistical terms as "the probability of failure-free operation of a computer program in a specified environment for a specific time". Measures of reliability- if we consider a computer-based system, a simple measure of reliability is mean-time-between-failure (MTBF),where MTBF = MTTF + MTTR, the acronym MTTF and MTTR are mean-time-to-failure and mean-time-to-repair respectively [2].Software reliability specification- Reliability is a complex concept that should always be considered at the system rather than the individual component level. Because the components in a system are interdependent, a failure in one component can be propagated through the system and affect the operation of other components. In a computer-based system, we have to consider three dimensions when specifying the overall system reliability:

Hardware reliability- What is the probability of a hardware component failing and how long would it take to repair that component? (ii) *Software reliability*- How likely is it that a software component will produce an incorrect output? Software failures are different from hardware failures in that software does not wear out: It can continue operating correctly after producing an incorrect result. (iii) *Operator reliability* – How likely is it that the operator of a system will make an error? [1].

Following are the basic terminologies that are frequently used for reliability-Table-1

System Failure	When the system does not perform as per the user expectations, then system failure occurs.
System Error	When the system gives the result in an unexpected manner then the system error occurs.
System Fault	It is probability of the system that the failure can lead to system error.
Human Error	It is human activity that makes the system fault to occur.

Mining Software Engineering Data –The main goal is to transform static record – keeping Software Engineering data to active data so that the hidden patterns and trends could be explored.Normally, a Software is "full of bugs", In Windows 2000, containing35 million lines of code, there were 63,000 known bugs at the time of release, 2 per 1000 lines. Software failure costs are becoming very high. A study by the National Institute of Standards and Technology found that software errors cost the U.S. economy about $59.5 billion annually. So testing and debugging are laborious and expensive. "50% of my company employees are testers, and the rest spends 50% of their time testing!" —Bill Gates, in 1995. In general Software is complex for e.g., MySQL has 1.2 millions of LOC and its runtime data is larger and more complex. In fact, finding bugs is challenging which requires specifications/properties, which often don't exist and also substantial human efforts in analysing data are required [3].

Software Reliability Methods are:

- Static Bug Detection - Without running the code, detect bugs in code**,**

International Journal of Data Mining Techniques and Applications
Volume: 01 Issue: 02 December 2012 Page No.36-39
ISSN: 2278-2419

- Dynamic Bug Detection (aka. Testing) - Run the code with some test inputs and detect failures/bugs and
- Debugging - Given known test failures (symptoms), pinpoint the bug locations in the code.

Mining for Soft Reliability is absolutely needed because,

i. Finding bugs is challenging.It requires specifications/properties, which often don't exist and also require substantial human efforts in analyzing data.
ii. We can mine common patterns as likely specifications/properties Detect violations of patterns as likely bugs.
iii. We can mine huge data for patterns or locations to narrow down the scope of human inspection

II. TECHNIQUES

The Software engineering tasks helped by data mining are – (i) programming,(ii)defect detection,(iii)testing,(iv)debugging and**(v)**maintenance.Data mining techniques are(i)Classification, (ii) Association, (iii) Patterns Detection, (iv) Clustering [4].

Software engineering data
Considered are- (i) Code bases, (ii)
change history, (iii) program states,(iv)
structural entities and (v) bug reports [5].

III. ANALYSIS

Data Mining for Software Bug Detection needs frequent pattern mining then automated Debugging in Software Programs is carried out from frequent patterns to software bugs and statistical debugging. Further, automated Debugging in computer **s**ystems is carried out from (i) Automated diagnosis of system misconfigurations and (ii) performance debugging [6].

A. Software Bug Detection

Common approach: mining rules/patterns from source code/revision histories and detecting bugs as rule/pattern violations.

B. Mining rules from source code

i. Bugs as deviant behaviour [Engler et al., SOSP'01]
ii. Mining programming rules with PR-Miner [Li et al., FSE'05]
iii. Mining function precedence protocols [Ramanathan et al., ICSE'07]
iv. Revealing neglected conditions [Chang et al., ISSTA'07]

C. Mining rules from revision histories

i. DynaMine [Livshits& Zimmermann, FSE'05]

D. Mining copy-paste patterns from source code

ii. CP-Miner [Li et al., OSDI'04] to find copy-paste bugs [7].

Bugs as Deviant Behaviour
Static verification tools need rules to check against program code.To find errors without knowing the truth

- Contradiction in belief. To find lies: cross-examine one witness or many witness. Any contradiction is an error (internal consistency)

- Deviation from common behaviour. To infer correct behaviour: if 1 person does X, might be right or a coincidence. If 1000s do X and 1 does Y, probably an error (statistical analysis)

IV. A BRIEF METHODOLOGY: SOFTWARE BUG DETECTION

Based on the discussion presented in the previous section, the following steps for software bug detection are presented.

Step 1:Mining rules from source code [8]

- Bugs as deviant 37ehaviour [Engler et al., SOSP'01]
- Mining techniques: Statistical analysis
- Mining programming rules with PR-Miner [Li et al., FSE'05]
- Mining function precedence protocols [Ramanathan et al., ICSE'07]
- Revealing neglected conditions [Chang et al., ISSTA'07]

Step 2:Mining copy-paste patterns from source code

- CP-Miner [Li et al., OSDI'04] to find copy-paste bugs

An Overview of Extracting Rules –
Observation: elements are usually used
together.Idea: finding association among elements that are*frequently used together*in source code Implies frequent item set mining [9].Examples:spin_lock_irqsave and spin_unlock_irqrestore
appear together within the same function more than 3600 times.

Step 3: Mining Programming Patterns and Generation of Rules –
Parsing Source Code – Purpose: building an item set database. Element: function call, variable, data type, etc. are mapped to a number. The Source code is mapped to an item set database.A frequent sub-item set corresponds to a programming pattern and application of frequent item set mining algorithm on the item set database.

E.g., {39, 68, 36, 92}:27 corresponds to pattern
{Scsi_Host, host_alloc, add_host, scan_host}

- Tradeoff: consider order or not

Step 4: Generating Programming Rules_Programming patterns - programming rules
E.g., Patterns: {*a, b, d*} : 3,
{*a*} : 4

International Journal of Data Mining Techniques and Applications
Volume: 01 Issue: 02 December 2012 Page No.36-39
ISSN: 2278-2419

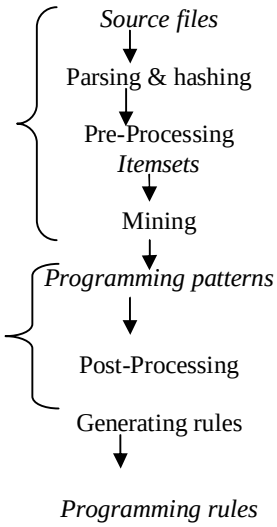


Fig.1 Flowchart of Extracting Rules

Rules:
{*a*} => {*b*,*d*} with confidence = ¾ =75%
{*b*} => {*a*,*d*} with confidence = 100%
{*d*} => {*a*,*b*} with confidence = 100%
{*a*,*b*} => {*d*} with confidence = 100%
{*a*,*d*} => {*b*} with confidence = 100%
{*b*,*d*} => {*a*} with confidence = 100%

Rule Explosion Problem
- Exponential number of rules
- Solution: closed mining

Example:
{*a*,*b*,*d*}:3, {*a*}:4
{*a*,*b*}:3, {*a*,*d*}:3, {*b*,*d*}:3 are not closed
- Close rules

{*a*,*b*,*d*}:3 | {*a*}:4

Detection of Violations

For violations of a programming rule

(i) The rule holds for most cases
- Confidence > threshold

(ii) The rule is violated for a few cases
- Confidence < 100%

Example: Detecting Violations

Step 5:Programming patterns:
{Scsi_Host, host_alloc, add_host, scan_host}: 27
{Scsi_Host, host_alloc, add_host}: 29

Programming rule:

{Scsi_Host, host_alloc, add_host}=>
{scan_host}
with confidence 27/29 = 93%

Missing

Table 2: Some Results of Bug Detection

Software	#C files	LOC	#functions
Linux	3,538	3,037,403	73,607
Postgre SQL	409	381,192	6,964
Apache	160	84,724	1,912

Table-3

Software	Inspected (top 60)		
	Bugs	Anomalies	False Positives
Linux	16	20	24
Postgre SQL	6	9	45
Apache	1	0	6

V. LIMITATIONS OF PR-MINER

Rules across multiple functions
- Not using inter-procedural analysis

False negatives of violations in control paths
- Not using sophisticated analysis techniques
- Inter-procedural, path-sensitive inference of function precedence protocols to address the limitations [Ramanathan et al., ICSE'07] [10].

We shall now discuss Mining Function Precedence Protocols

```
fp = fopen(…);
fclose(…);
```

a) Definition:-Precedence protocol:
A call tofcloseis always preceded by a call tofopen

b) Definition:-Successor protocol :
A call tofopenis always succeeded by a call tofclose

c) Violation of Precedence Protocols

```
fp = fopen(…);
if(fp == NULL)
exit(-1);
fclose(…);
```

d) Tool Implementation/Evaluation
CHRONICLER – tool implemented in C has the following features:

- Tested on open source C programs
- Apache, linux, openssh, gimp,postgresql
- Lines of code varies from 66K to 2M
- Number of call-sites varies from 10K to 110K

e) Some Results of Precedence-Related Bug Detection

Case Study: Linux

Hardware Bug
- Difficult to detect using traditional testing techniques
- Platform dependent error
- Transparently identified using CHRONICLER

International Journal of Data Mining Techniques and Applications
Volume: 01 Issue: 02 December 2012 Page No.36-39
ISSN: 2278-2419

Performance Bug

- Cache lookup operation was absent
- Not easily specified as a bug for testing
- Deviation delays data write flushes [11].

f) Limitation of Precedence-Related Bug Detection

- Does not take data flow or data dependency into account
- A new approach to discovering neglected conditions [Chang et al., ISSTA'07] addresses the issue
- Based on dependence analysis, frequent item set, and frequent sub graph mining

g) Crucial Observation

Things that are frequently changed together often form a pattern...also known as co-changeCo-changed items = patterns

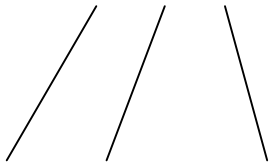
h) Finding Patterns

Find "frequent itemsets" (with Apriori)
o.enterAlignment()
o.exitAlignment()
o.redoAlignment()
iter.hasNext()
iter.next()
{enterAlignment (), exitAlignment(),
redoAlignment()}

i) Ranking Patterns
Support count = #occurrences of a pattern
Confidence count= Strength of a pattern, P (A|B)

j) Pattern classification

Post-process
v validations, e violations



Usage patterns | error patterns | unlikely patterns

e<v/10 | v/10<=e<=2v | otherwise

Fig. 2

Results of Mining Patterns
Usage pattern – 15
Error Pattern- 8
Unlikely Pattern – 11
Not Hit – 24
Total – 56 Patterns

Mining into Computer SystemsHuge volume of data from computer systems

Persistent state interactions, event logs, network logs, CPU usage …
Mining system data for …

Reliability
Performance
Manageability …

VI. CONCLUSION

Challenges in data mining-Statistical modelling of computer systemsOnline, scalability, interpretability …Data Mining for Software Bug DetectionFrequent pattern mining.Automated Debugging in Software Programs-From frequent patterns to software bugs.Statistical debugging-Automated Debugging in Computer Systems.Automated diagnosis of system misconfigurations.Limitations of Bugs as Deviant Behaviour Fixed rule templates.Need specific knowledge about the software.2 elements.PR-Miner [Li et al., FSE'05] (mining implicit programming rules) developed to address the limitations.General method (No prior knowledge; No templates).General rules (Different types: function, variable, data type, etc.;Multiple elements)Ubiquitous computing demands reliable software- Mining for software reliability.Mining program source code/version histories to find bugs.Mining program runtime data to locate why an execution fails.Mining system snapshots to diagnose misconfigurations and performance problems.An active and rewarding research area.International Workshop on Mining Software Repositories since 2004.SIGCOMM Workshop on Mining Network Data since 2005.Systems and Machine Learning Workshop since 2006.Workshop on Statistical Learning Techniques for Solving Systems.Problems, co-located with NIPS

REFERENCES

[1] Ian Sommerville, Software Engineering 8th edition, Pearson Education Publications, 2007.
[2] Roger S. Pressman, Software Engineering: A Practitioner's Approach, 6th edition McGraw-Hill International edition Publications, 2005.
[3] James S. Peters &WitoldPedrycz, Software Engineering an Engineering Approach, Wiley Publications, 2000.
[4] Jiawei Han &MichelineKamber, Data Mining: Concepts and Techniques, 2nd edition,, Elsevier Publications, March 2006.
[5] Chai Liu, Long Fei, Xifang Yan, Jiawei Han and Samuel Midkiff, Statistical Debugging: A Hypothesis Testing-based approach, IEEETSE 2006.
[6] Dawson Engler, David Yu Chen, Seth Hallem, Andy Chou and Benjamin Chelf, Bugs as Deviant Behaviour: A General approach to inferring errors in systems code, SOSP 2001.
[7] Zhenmin Li, Shan Lu, SuvdaMyagmar and Yuanyan Zhou, CP-Miner: A tool for finding copy-paste and related bugs in operating system code, OSPI 2004.
[8] Prof. S. Chitra&Dr. M. Rajaram, A Software Reliability Estimation tool using Artificial Immune Recognition System: Proceedings of the International Multiconference of Engineers and computer scientists 2008 vol 1, IMECS 2008, pp. 19-21 March 2008, Hong Kong.
[9] Leon Wu, BoyiXie, Gail Kaiser & Rebecca Passonneau, Department of Computer Science, Columbia University, Newyork NY 10027 USA, BUGMINER: Software Reliability Analysis via Data Mining of Bug Reports2007.
[10] Swapna S. Gokhale, Member, IEEE, A Simulation Approach to structured-based software reliability analysis, IEEE transactions on Software Engineering, vol 31, No. 8, August 2005.
[11] Simon P. Wilson and Francisco J. Samaniego,Nonparametric Analysis of the order-statistic model in software reliability, IEEE transactions on software engineering, vol 33, No. 3, March 2007.