CHAPTER 4

RELIABILITY FRAMEWORK FOR OSS

4.1 INTRODUCTION

Open Source Software in general are symbolized with freedom to use the software product as one prefers to, edit it, to fit it to ones needs, redistribute it, obviously by improving it, fixing its bugs and augmenting its functionality. The success and benefits of OSS can be attributed to many factors such as code modification by any party as the needs arise, promotion of software reliability and quality due to peer review and collaboration among many volunteer programmers from different organizations, and the fact that the knowledge-base is not bound to a particular organization, which allows for faster development and the likelihood of the software to be available for different platforms.

A fault or bug is a defect in software that has the potential to cause the software to fail, whereas an error is a measured value or condition that deviates from the correct state of software during operation. A failure is the inability of the software product to deliver one of its services. Therefore, a fault is the cause for an error, and software that has a bug may not encounter an error that leads to a failure. Hence, there is a need to fix bugs that are potential cause for the failure of software.

In order that any open source software may be reused, its reliability factor must be examined such that bugs are not linearly inherited. As software

products have become increasingly complex, software reliability is a growing concern, which is defined as the probability of failure free operation of a computer program in a specified environment for a specified period of time (Musa 1984). Reliability growth modeling has been one approach to address software reliability concern, which dates back to early 1970's (Jelinski 1972) (Shooman 1972). Hence, the demand for the reliability factor among the open source software arises and with this conception the framework is proposed.

4.2 OBJECTIVE

The objective of the chapter is to propose a system that could check the reliability of the open source software such that reliability of the OSS is maintained.

4.3 RESEARCH EXPERIMENTS

In this research, metric suites validation for OSS, fault tolerance and mapping of JAR files are proposed and experimentally substantiated with the use of Rhino, an open source software. These are elaborated in the forthcoming sections.

Metric Suites Validation Using Fault Proneness

The increasing complexity and constraints in the world of programming intensifies the difficulty in producing software without faults such that development & maintenance cost due to software failures and customer's satisfaction is reduced. The fact extraction process, by which different properties of the system are studied, is used in OSS to battle against uncertainty. In the research, metric suites namely Chidamber and Kemerer metrics suite, Robert C Martin metrics suite and McCabe's metric suite are

experimented as to validate them for OSS. Exposing the fault prone components of any Open Source Software would emancipate the time constraints involved in identification of problem areas of the components. The correlation among the fault proneness and the various measurable attributes of code is investigated and results are enlisted. The bug report collected from the software development industry was used in the analysis of fault proneness. From the correlational analysis and verification, the metric suites analyzed were categorized to various metrics effective of them.

Fault Tolerance in OSS

Software-fault tolerance is concerned with all the techniques necessary to enable a system to tolerate software design faults Jie et al (2008). In order to discuss software-fault tolerance, a system is defined to consist of a set of components which interact under the control of a design. Fault tolerance of various class programs, packages, JAR files are calculated. Automated process of counting the errors in the classes or methods or packages is achieved in the research. Thus, the process reduces manual effort and process time. Basing on the results of the automated process, suggestions on the choice of a software is made. Threshold is a point beyond which there is a change in the execution of the program Chandra et.al(2010). A threshold is set for each of the parameters and the conformance of the extracted values are expected with the set threshold values i.e the parameters values are expected to lie within the range of the thresholds. If the value lies within the range then the class satisfies the OO design paradigms. If the parameters lie outside the threshold then appropriate suggestions are provided to the developer or user.

Deducing and Mapping the class path of JAR file

A methodology to compile the package automatically and set up the proper classpath is proposed and implemented in this research. The deducing and mapping the class path is achieved by selecting the software, compiling it and deducing the errors from it. Following the above process, the errors are classified based on the category. If the errors are due to mapping errors then correct path is set. The experiment is conducted and results are analysed. The error rate reduces considerably after implementing the methodology of automating the deducing and mapping of JAR files.

4.4 PROPOSED SYSTEM FOR OSS

The value of measurement is summarised fittingly by the often quoted statement — “You can’t control what you can’t measure” (DeMarco 1982). Various studies (Basili 1996) (Coleman 1995) have complemented the work undertaken to ensure that the metrics defined are mathematically valid and useful. There have also been studies (Hitz 1996) (Cartwright 2000) that have shown the applicability as well as the limitations of software metrics for measuring both size and complexity of a software system.

A framework is proposed in the research comprising of two phases in order to render support for the use and reuse of Open Source Software by enhancing its reliability. In the first phase the empirical validation of metric suite for OSS is carried out and in the second phase the reliability- checking system is constucted.

In the first phase, the methodology adopted involves the following steps as shown in Figure 4.1.

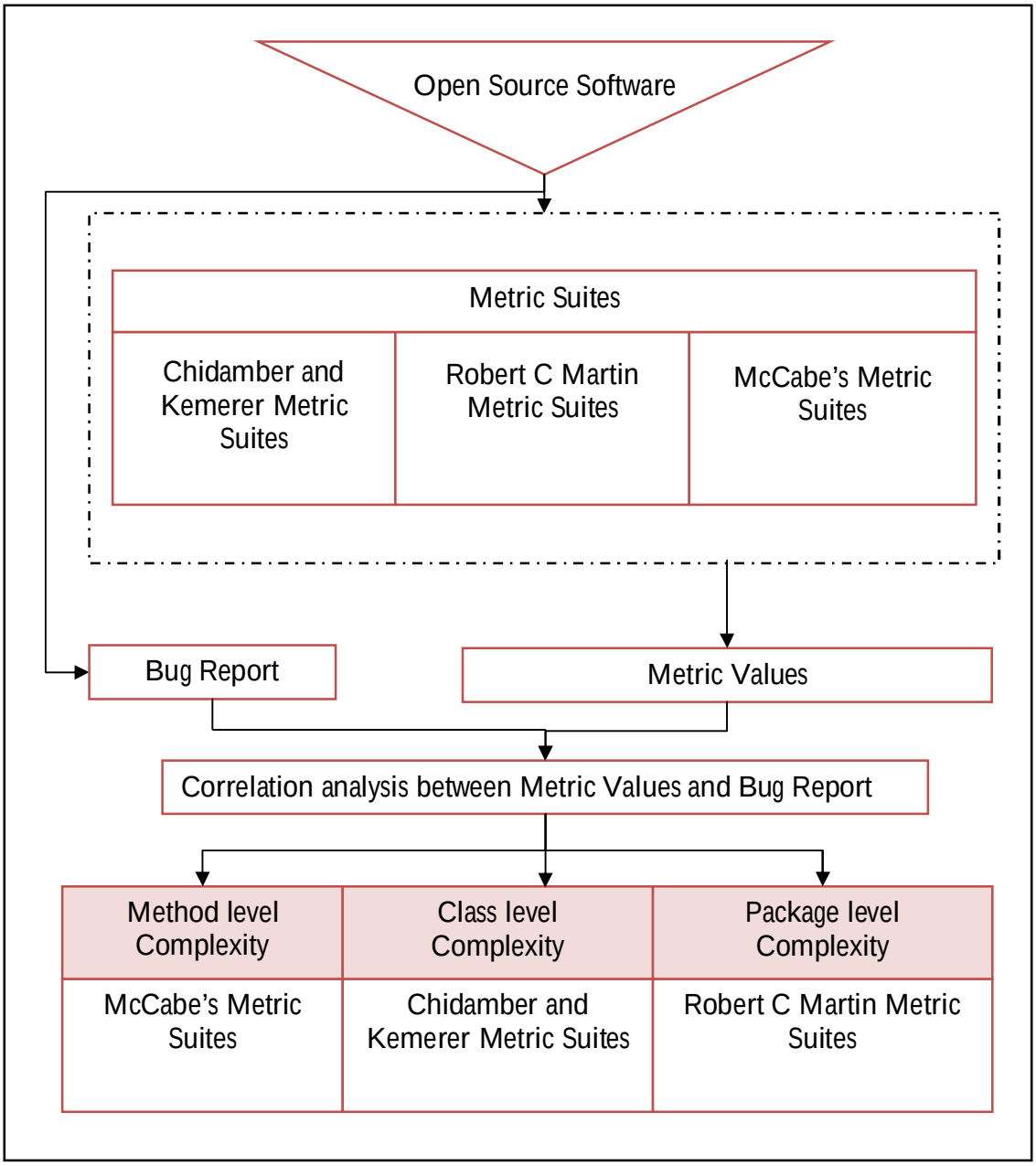

Figure 4.1 Empirical Validation of Metric Suite for OSS

The downloaded Open Source Software is given as an input to the empirical validation system. The software is analysed with the three metric suites taken for analysis namely Chidamber and Kemerer metrics suite, Robert C Martin metrics suite and McCabe's metric suite. The metric values obtained are correlated with the bug report collected from the developers site.

Based on the correlated analysis, measures are appropriated with a metric suite that has a high a correlation value. In the case of Rhino, an OSS, the processes are shown in Figure 4.2. Method level complexity, Class level complexity and Package level complexity are appropriated with McCabe's metric suite, Chidamber and Kemerer metrics suite and Robert C Martin metrics suite respectively.

In the second phase of the Reliability–Checking system for OSS, the metric values are obtained from phase I. The phae II of the framework for reliability checking system is shown in Figure 4.2. The values of complexity, with regard to method, class and package from the previous phase after the selection of highly correlated values, are given as an input to the phase II. Method Complexity, Class Complexity and Package Complexity are taken into consideration for the further analysis. In the case of Rhino Open Source Software which was taken for analysis in the research, it had Chidamber and Kemerer metric suite for the Method level complexity, McCabe metric suite for Class level complexity and Robert C Martin for the Package level complexity.

Threshold Values (TV) of various metric are examined against the metric values obtained from the metric suites. In the case that the metric values are higher that the threshold values then they are considered to be non-reliable OSS and corresponding suggestion is made to the user. And if the metric values are less than the threshold value then the fault proneness value is identified based on the metric values since there is high correlation between fault proneness and metric values.

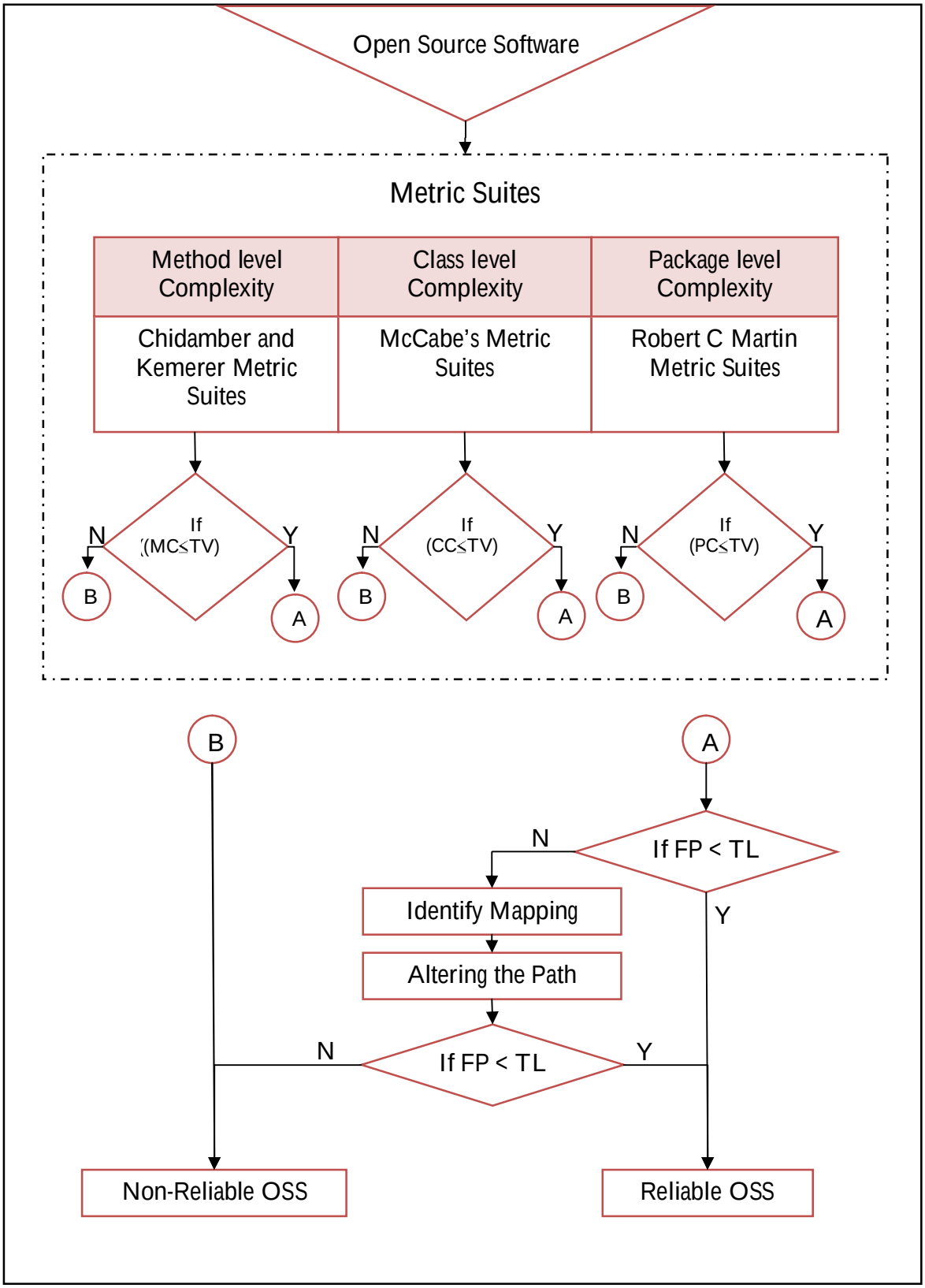

Figure 4.2 Reliability Checking System for OSS

The following are the expansion of the abbreviations used in Figure 4.2:

MC - Method Complexity

CC - Class Complexity

PC - Package Complexity

TV - Threshold Value

TL - Tolerance Level

FP - Fault Proneness

The fault proneness is checked against the Tolerance Level of acceptance. If fault proneness is less than the tolerance value then it is suggested as a reliable open source software. In the case that the values are higher than the tolerance value, then the proposed automated mechanism for mapping correctly is implemented for reducing the deducing and mapping errors in the jar/software. After deducing and mapping correctly if still the fault proneness exceeds the tolerance level, then the OSS is not suggested as a reliable software or else the OSS is suggested as a reliable software.

4.5 CONCLUDING REMARKS

This chapter commences with the brief introduction on Open source software, bugs and the need for reliability that would enhance the usage. The proposed research experiments are presented briefly as to give a thrust to the proposed reliability system. The methodology of the Reliability-Checking system with two phases are put forward with a neat sketch. Finally, the chapter ends with comprehensible explanation such that the proposed system would enhance the usage of open source software by increasing its reliability and suggestions are provided based on the outcome of the system. The elaborations of the proposed research are presented lucidly in the further chapters.