Chapter 4

**METHODOLOGY AND IMPLEMENTATION**

Statistical validation for the proposed dynamic object-oriented metric suite was conducted next. Four object-oriented systems from Educational Institution Project (EIP), written in Java 1.5, were taken as the test cases for this research. A new Java-based software tool called DynaMetrics [Singh and Singh(2008)] was designed and developed for this purpose using two open source Integrated Development Environments (IDEs) called NetBeans 5.5 [NB] and Eclipse 3.2 [EI]. The next subsection describes the test cases used for this study. It would be followed by the description of methodology adopted to conduct the metric validation.

4.1 Test Case

The test case considered for the empirical study was the Educational Institution Project (EIP) developed at National Institute of Technology Jalandhar. The project developers were the students of an undergraduate level course offered by the Department of Computer Science and Engineering. EIP project was one of the final semester major projects in the course. The objective of this class was to learn *Software Engineering* and *Object-Oriented System Design*.

The students did not require any previous experience or training in the application development or object-oriented methodology. All students had some experience with Java Programming versions 1.4 and 1.5, and therefore had the basic skills necessary for such a study.

4.1.1 The Development Process

Students were divided into 4 teams, each consisting of 2 students. Team1 was asked to develop the System1 implementing the Library Issue Record (LIR) system related to students and employees. The system comprised of 5 classes. Team 2 was assigned to enhance the system built by Team2. Team 2 modified System1 to build System2 comprising of 10 classes. Team3 further modified System2 to develop System3 consisting of 30 classes. Finally Team4 modified System3 to develop System4 having 47 classes.
First two systems implemented the Library Issue Record system whereas System3 included student information related classes and staff information related classes. These classes included implementations related to result evaluation, hostel allotments, general club inventory and employee records etc. System4 in addition to classes in system3 added, implementation of various student clubs, engineering departments, employee hostels and central store.

Such an application domain had the advantage of being easily comprehensible and, therefore, we could make sure that system requirements could be easily interpreted by students regardless of their educational background.

4.1.2 System Description

The detailed modular structures of the four test case systems are demonstrated in Figure 10-13 respectively. The modules of each system are mentioned below:

a) System1
   - Library
   - Employee

b) System2
   - Library
   - Employee

c) System3
   - Library
   - Employee
   - Student

d) System4
   - Library
   - Employee
   - Student
   - Department
   - Student Club
Chapter 4 – Methodology and Implementation

**Figure 10: Modular structure - System1**

**Figure 11: Modular structure – System2**
Figure 12: Modular structure – System3
Figure 13: Modular structure – System4
4.2 Methodology

The methodology followed for this research is described as follows.

4.2.1 Static Analysis

Static analysis of the test cases is carried out to evaluate the metric values before runtime. JVMTI is used for this purpose to instrument the bytecode of the application systems.

4.2.2 Creating the runtime logs

Runtime logs of the test cases are created. The logs contain all the information about the messages passed among object methods at runtime in plain text log files. For Example, a log file(s) may contain entries in the form

\begin{verbatim}
Class1.Method1 CALLS Class2.Method2
or
ENTERS: Class1.Method1
ENTERS: Class2.Method2
EXITs: Class2.Method2
EXITs: Class1.Method1
\end{verbatim}

4.2.3 Metric Evaluation

The log files are further used to access information required to evaluate the value of a runtime metric.

4.2.4 Result Analysis

The results obtained from the static and dynamic analysis are statistically analyzed to find out any relationship/dependencies among metrics.

We developed a metric evaluation tool called DynaMetrics to perform the first three steps of the methodology. The next section explains the working of DynaMetrics tool in detail.

4.3 DynaMetrics –Features and Working

DynaMetrics is a runtime metric evaluation and analysis tool that calculates the static and dynamic metric values for Java/C++ programs. DynaMetrics has been completely written in java. Currently it supports java programming language based test cases but it is being enhanced to support the object-oriented systems written in other programming languages.
such as C++. Figure 14 shows a brief snapshot of DynaMetrics tool at work. Appendix-A shows more DynaMetrics metric evaluation snapshots in Figures 75-99.

4.3.1 Features

DynaMetrics has the following key features/operations:

- Interactive Graphical User Interface.
- It can evaluate the class-level, object-level, method-level and system-level runtime metrics for Java and C++ programs.
- It statistically analyzes the metric values and helps in relating them to software’s quality attributes like reuse, maintainability, fault-tolerance etc.

![DynaMetrics Snapshot](image)

**Figure 14: DynaMetrics snapshot**

- It has multi-platform support. It can be installed on all the platforms that have Java 2 or higher runtime environment (Windows 9x/ME/NT/2000/XP, Unix, Linux).
- It compares the runtime and static values of a metric, and find the following conclusions:
  - which metric values vary a lot from static to runtime evaluation, indicating the usefulness of the dynamic version.
  - what can be the best metric combination comprising of static and dynamic metrics for each object-oriented software systems.
• It is the first tool to process some of the lately proposed dynamic metrics suites [Mitchell and Power(2003b&2004b), Zaidman and Demeyer(2004), Arisholm(2002), Yacoub et al.(1999)].

4.3.2 JVMTI Technology [JVMTI]

The Java Virtual Machine Tool Interface (JVMTI) is a programming interface used by development and monitoring tools. It provides both a way to inspect the state and to control the execution of applications running in the Java Virtual Machine (JVM). JVMTI was introduced in J2SE 5.0 to provide an Application Programming Interface (API) to support the full breadth of tools that need access to JVM state, including but not limited to: profiling, debugging, monitoring, thread analysis, and coverage analysis tools.

JVMTI was defined through the Java Community Process by JSR-163. The JVMTI replaces the JVMPI (Java Virtual Machine Profiling Interface) and the JVMDI (Java Virtual Machine Debug Interface). The JVMPI and the JVMDI are declared as being deprecated in J2SE 5.0 and were removed in Java SE6. JVMTI is the lowest level of the JPDA (Java Platform Debugger Architecture). JVMTI may not be available in all implementations of the JVM.

4.3.2.1 Using JVMTI

The JVMTI is a two way native interface of the JVM. A library, written in C or C++, is loaded during the initialization of the JVM. The library has access to the JVM state by calling JVMTI and JNI (Java Native Interface) functions and can register to receive JVMTI events using event handler functions that are called by the JVM when such an event occurs. Figure 15 demonstrates this process.

A client of JVMTI, hereafter called an agent, can be notified of interesting occurrences through events. These agents, or "native" libraries often form a basis for the Java technology-level tool APIs, such as the Java Debugger Interface (JDI) that comes with the Java Development Kit (JDK). Profiler tool vendors will often need to create an agent library that uses JVMTI. JVMTI can query and control the application through many functions, either in response to events or independent of them to carry out BCI (Byte Code Instrumentation) and Runtime Events Instrumentation of the application. Agents run in the same process with and communicate directly with the virtual machine executing the application being examined. This communication is through a native interface (JVMTI).
The native in-process interface allows maximal control with minimal intrusion on the part of a tool. Typically, agents are relatively compact. They can be controlled by a separate process which implements the bulk of a tool's function without interfering with the target application's normal execution.

Figure 15: JVMTI for BCI and JVM events monitoring

A number of tools are based on this interface, such as Hyades, JProfiler, and Ariadna. DynaMetrics is written in Java 5.0 using JVMTI. Tools can be written directly to JVMTI or indirectly through higher level interfaces. The JPDA includes JVMTI, but also contains higher-level, out-of-process debugger interfaces.

4.3.3 Working

DynaMetrics is a runtime metric-based collection tool that works over the data collected at runtime to evaluate the metric values. Designed and developed in Java, DynaMetrics works over event log files created at runtime for getting the runtime information about particular java program. The working structure of DynaMetrics is shown in Figure 16. DynaMetrics sits between the JVM/JVMTI (Java Virtual Machine) and the Java/C++ code to intercept and collect the useful information. DynaMetrics has five main components as shown in Figure 16. Each component is implemented through a class as described below:
• **Graphical User Interface (GUI)**

   DynaMetricUI class implements the GUI shown in Figure 14.

• **Static Analyzer**

   This class parses the source code of application software to extract the useful static information. It further stores this information in a log file.

• **Runtime Tracer**

   This class traces the runtime events like objects created, methods called etc. and stores the information in a log file. It uses Java Virtual Machine Tool Interface (JVMTI) [JVMTI] which is one of the debugging interfaces provided by Java Platform Debugging Architecture (JPDA) [JPDA]. The C++ program evaluation is being implemented by embedding tcc compiler in JVMTI agent.

---

![Diagram of DynaMetrics workflow](image)

**Figure 16: DynaMetrics – Work flow**

• **Metric Evaluator**

   The information collected by **Static Parser** or **Runtime Tracer** is then fed to **Metric Evaluator**. Metric Evaluator class then processes this information to provide the required static and dynamic metric values.
• *Metric Interpreter*

Metric Interpreter works with the values created by the *Evaluator*. It analyzes and compares the static and dynamic values for the required metric using bar charts, line charts, pie charts.

### 4.3.4 Tools Used

a) **Eclipse IDE 3.2**

Eclipse[EI] is an open-source Integrated Development Environment (IDE) for Java/C++ developers, consisting of the Java Development Tools (JDT) and the Eclipse Compiler for Java (ECJ). Figure 17 shows the development snapshot of DynaMetrics in Eclipse. All the components except GUI were implemented in Eclipse.

![Figure 17: DynaMetrics development view – Eclipse 3.2 & NetBeans 5.5](image)

b) **NetBeans IDE 5.5**

NetBeans[NB] refers to an IDE for the development of Java desktop applications. NetBeans 5.5 was used to build the GUI (Graphical User Interface) for DynaMetrics as shown in Figure 17.
4.4 SPSS Statistical Tool

SPSS (Statistical Package for the Social Sciences) Statistics 17.0 [SPSS] is a comprehensive system for analyzing data. SPSS was released in its first version in 1968 after being developed by Norman H. Nie and C. Hadlai Hull. SPSS Statistics can take data from almost any type of file and use them to generate tabulated reports, charts, and plots of distributions and trends, descriptive statistics, and complex statistical analysis. SPSS is among the most widely used programs for statistical analysis in social science. It is used by market researchers, health researchers, survey companies, government, education researchers, marketing organizations and others. SPSS Statistics makes statistical analysis more accessible for the beginner and more convenient for the experienced user. Simple menus and dialog box selections make it possible to perform complex analysis without typing a single line of command syntax. The Data Editor offers a simple and efficient spreadsheet-like facility for entering data and browsing the working data file.

4.4.1 SPSS Statistics Options

Statistics included in the base software:

- **Descriptive statistics:** Cross tabulation, Frequencies, Descriptives, Explore, Descriptive Ratio Statistics.
- **Bivariate statistics:** Means, t-test, ANOVA, Correlation (bivariate, partial, distances), Nonparametric tests.
- **Prediction for numerical outcomes:** Linear regression.
- **Prediction for identifying groups:** Factor analysis, cluster analysis (two-step, K-means, hierarchical), Discriminant.

The following options are available as add-on enhancements to the full (not Student Version) SPSS Statistics Base system:

- **Regression** provides techniques for analyzing data that do not fit traditional linear statistical models. It includes procedures for probit analysis, logistic regression, weight estimation, two-stage least-squares regression, and general nonlinear regression.
- **Advanced Statistics** focuses on techniques often used in sophisticated experimental and biomedical research. It includes procedures for general linear models (GLM), linear mixed models, variance components analysis, loglinear analysis, ordinal
regression, actuarial life tables, Kaplan-Meier survival analysis, and basic and extended Cox regression.

- **Custom Tables** creates a variety of presentation-quality tabular reports, including complex stub-and-banner tables and displays of multiple response data.

- **Forecasting** performs comprehensive forecasting and time series analyses with multiple curve-fitting models, smoothing models, and methods for estimating autoregressive functions.

- **Categories** perform optimal scaling procedures, including correspondence analysis.

- **Conjoint** provides a realistic way to measure how individual product attributes affect consumer and citizen preferences. With Conjoint, you can easily measure the trade-off effect of each product attribute in the context of a set of product attributes—as consumers do when making purchasing decisions.

- **Exact Tests** calculates exact $p$ values for statistical tests when small or very unevenly distributed samples could make the usual tests inaccurate. This option is available only on Windows operating systems.

- **Missing Values** describes patterns of missing data, estimates means and other statistics, and imputes values for missing observations.

- **Complex Samples** allows survey, market, health, and public opinion researchers, as well as social scientists who use sample survey methodology, to incorporate their complex sample designs into data analysis.

- **Decision Trees** creates a tree-based classification model. It classifies cases into groups or predicts values of a dependent (target) variable based on values of independent (predictor) variables. The procedure provides validation tools for exploratory and confirmatory classification analysis.

- **Data Preparation** provides a quick visual snapshot of your data. It provides the ability to apply validation rules that identify invalid data values. You can create rules that flag out-of-range values, missing values, or blank values. You can also save variables that record individual rule violations and the total number of rule violations per case. A limited set of predefined rules that you can copy or modify is provided.

- **Neural Networks** can be used to make business decisions by forecasting demand for a product as a function of price and other variables, or by categorizing customers based on buying habits and demographic characteristics. Neural networks are non-
linear data modeling tools. They can be used to model complex relationships between inputs and outputs or to find patterns in data.

- **EZ RFM** performs RFM (recently, frequency, monetary) analysis on transaction data files and customer data files.
- **Amos (Analysis of moment structures)** uses structural equation modeling to confirm and explain conceptual models that involve attitudes, perceptions, and other factors that drive behavior.

SPSS places constraints on internal file structure, data types, data processing and matching files, which together considerably simplify programming. SPSS datasets have a 2-dimensional table structure where the rows typically represent cases (such as individuals or households) and the columns represent measurements (such as age, sex or household income). Only 2 data types are defined: numeric and text (or "string"). All data processing occurs sequentially case-by-case through the file. Files can be matched one-to-one and one-to-many, but not many-to-many.

The graphical user interface (Figure 18) has two views which can be toggled by clicking on one of the two tabs in the bottom left of the SPSS window. The 'Data View' shows a spreadsheet view of the cases (rows) and variables (columns). Unlike spreadsheets, the data cells can only contain numbers or text and formulas cannot be stored in these cells. The 'Variable View' displays the metadata dictionary where each row represents a variable and shows the variable name, variable label, value label(s), print width, measurement type and a variety of other characteristics. Cells in both views can be manually edited, defining the file structure and allowing data entry without using command syntax. This may be sufficient for small datasets. Larger datasets such as statistical surveys are more often created in data entry software, or entered during computer-assisted personal interviewing, by scanning and using optical character recognition and optical mark recognition software, or by direct capture from online questionnaires. These datasets are then read into SPSS.

SPSS can read and write data from ASCII text files (including hierarchical files), other statistics packages, spreadsheets and databases. SPSS can read and write to external relational database tables via ODBC (Open Database Connectivity) and SQL (Structured Query Language).

Statistical output is to a proprietary file format (*.spv file, supporting pivot tables) for which, in addition to the in-package viewer, a stand-alone reader can be downloaded.
The proprietary output can be exported to text or Microsoft Word. Alternatively, output can be captured as data (using the OMS command), as text, tab-delimited text, PDF, XLS, HTML, XML, SPSS dataset or a variety of graphic image formats (JPEG, PNG, BMP and EMF).

**Chapter Summary**

Chapter 4 explains the methodology followed to solve the research problem. This chapter initially focuses on the test cases considered for metric validation. Various modules of the Educational Institution Project (EIP), developed in java, are discussed. The test case description is followed by the stepwise methodology followed to validate the metrics. A new tool, called DynaMetrics, that was designed and developed in this research for static and dynamic metric evaluation, is briefly explained next. DynaMetrics was written in java.
using Eclipse 3.2 and NetBeans 5.5. Finally the description of SPSS Statistics 17.0 tool that is used for statistical analysis of metric data concludes the chapter. Next chapter includes the analysis of metric evaluation results, obtained from the test case execution, for validating the proposed dynamic metric suite