University of Mosul
College of Computer Sciences
And Mathematics

Constructing a Tool for Measuring a Quality of Object Oriented Design to Enterprise Architect v7.5

Khalil Ahmed Ibrahim Al-Hadidi

M.Sc./Thesis
Software Engineering

Supervised By

Dr. Laheeb Mohammed Ibrahim Al-Zobaidy
Assistant Professor

2013 A.D. **1434 A.H.**

Constructing a Tool for Measuring a Quality of Object Oriented Design to Enterprise Architect v7.5

A Thesis Submitted by

Khalil Ahmed Ibrahim Al-Hadidi

To
The Council of the College of
Computer Sciences and Mathematics
University of Mosul

in Partial Fulfillment of the Requirements
for The Degree of Master of Sciences
In
Software Engineering

Supervised By

Dr. Laheeb Mohammed Ibrahim Al-Zobaidy
Assistant Professor

2013A.D **1434 A.H**

بسم الله الرحمن الرحيم

(رَبِّ أَوْزِعْنِي أَنْ أَشْكُرَ نِعْمَتَكَ الَّتِي أَنْعَمْتَ عَلَيَّ
وَعَلَى وَالِدَيَّ وَأَنْ أَعْمَلَ صَالِحًا تَرْضَاهُ وَأَدْخِلْنِي
بِرَحْمَتِكَ فِي عِبَادِكَ الصَّالِحِينَ)

سورة النمل

الآية (١٩)

صدق الله العظيم

Acknowledgments

It has been a long journey, and I could not have completed it all on my own. I have many people to thank, who have accompanied me, enlightened me, supported me, encouraged me, and helped me. To all of them, I am deeply grateful.

First, I wish to express my thanks and gratitude to Allah, the most gracious the most merciful for uncountable help guidance.

I wish to express my thanks and gratitude to my supervisor **Dr. Laheeb Mohammed Ibrahim** for her valuable guidance and advice. She inspired me greatly to work on this project. Her willingness to motivate me contributed tremendously to this project.

A special thanks goes to Dr. Dujan for her help at the beginning of this journey.

I would like to present my thanks to the head and the staff of Software Engineering Department, college of computer science, university of Mosul for their help.

I am indebted to all my friends and colleagues especially Saad Ahmed for his ideas, encouragement and moral support. Also a special thanks goes to my colleague Tawfeq for his help in doing the questionnaire.

I would not be who I am or be able to complete this difficult task without the influence and education I received from my family, I thank my father, my mother, my brothers and sisters, especially Ibrahim and Sumaya for their technical and coding advice ,and for their encouragement throughout the way.

Finally, to the many people who have influenced me and my work, and are not mentioned by name, I thank you.

Khalil Ahmed Ibrahim

Abstract

Software developers are always thankful for tools that do not interfere too much with their work habits. Software metrics can offer some feedback on the quality and results of design decisions, and help developers move towards an engineering approach to software. Metrics are a useful resource to help determine whether software meets desired quality characteristics. Although many types of metrics have been devised, design metrics for object orientation have reached one of the highest levels of maturity because their relationship to lower maintenance efforts, lower defect rates, and other benefits have been demonstrated repeatedly in studies. As such, basic design concepts such as encapsulation, inheritance, polymorphism, are known indicators of software quality.

As UML (Unified Modeling Language) becomes a de facto standard notation for software documentation, effective design metrics must increasingly work with UML diagrams as input. Some tools have the ability to calculate design metrics from UML diagrams and others cannot.

In this study, three tools are developed. First, is “Khalil Design Metrics tool (KDM)” as an add-in that works inside Enterprise Architect v7.5 which is a well-known, powerful CASE (Computer Aided Software Engineering) tool. KDM tool takes the XMI (XML Metadata Interchange) document for the UML class diagram exported by EA as input, processes it, calculates and visualize metrics, provides recommendations about design naming conventions and exports metrics as XML (Extensible Markup Language) document in order to communicate with other tools namely KRS (Khalil Reporting Service) and KDB (Khalil Database).

Two object oriented design metrics models are used, namely MOOD (Metrics for Object Oriented Design) which measures Encapsulation, Inheritance,

Polymorphism and Coupling, and MEMOOD (Maintainability Estimation Model for Object Oriented software in Design phase) which measures Understandability, Modifiability and Maintainability. Both models are validated theoretically and empirically.

These measurements allow designers to access the software early in process, make changes that will reduce complexity and improve the design.

A Second tool is “KRS” which takes XML document generated by KDM tool as input, parses it and gives a report. The report helps the project manager or the team leader to monitor the progress and to document the metrics. Hence KRS tool is integrated with Enterprise Architect.

A Third tool is “KDB” which takes the same XML document generated by KDM tool as input, parses it and stores metrics in the database to be used as a historical data. KDB tool is also integrated with Enterprise Architect.

All three tools were developed using C# programming language with the aid of Microsoft Visual Studio 2010 as integrated development environment under Windows 7 operating system with minimum 4 GB of RAM and Core-i3 of CPU.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Symbol	Meaning
AHF	Attribute Hiding Factor
AIF	Attribute Inheritance Factor
ANSI	American National Standard Institute
API	Application Programming Interface
CASE	Computer Aided Software Engineering
CC	Cyclomatic Complexity
CF	Coupling Factor
CHM	Microsoft Compiled HTML Help
CK	Chidember and Kemrer
COM	Component Object Model
DLL	Dynamic Link Library
DOM	Document Object Model
EA	Enterprise Architect
EMOOSE	Extended Metrics for Object Oriented Software Engineering
ER	Entity Relationship
FP	Function Points
GQM	Goal Question Metrics
GUI	Graphical User Interface
HKCR	HKEY_CLASSES_ROOT
HKCU	HKEY_CURRENT_USER
HMS	Hospital Management System
HOM	Hotel Management System
HTML	Hyper Text Markup Language
IDE	Integrated Development Environment
IPSE	Integrated Project Support Environment
KDB	Khalil Database
KDM	Khalil Design Metrics
KRS	Khalil Reporting Service
LOC	Line of Code
MDG	Model Driven Generation
MEMOOD	Maintainability Estimation Model for Object Oriented Software in Design Phase
MHF	Method Hiding Factor
MI	Maintainability Index
MIF	Method Inheritance Factor
MOOD	Metrics for Object Oriented Design
MOOSE	Metrics for Object Oriented Software Engineering
OLE	Object Linking and Embedding
OMG	Object Management Group
OO	Object Oriented
OOD	Object Oriented Design
POF	Polymorphism Factor
QMOOD	Quality Model for Object Oriented Design

SDK	Software Development Kit
SDLC	Software Development Life Cycle
UML	Unified Modeling Loanguage
URL	Uniform Resource Locator
XMI	XML Metadata Interchange
XML	Extensible Markup Language

Chapter One

Introduction

Introduction

1.1 Preface

Software is becoming the foundation of modern civilization. It affects almost all aspects of our lives and our everyday activities. With software engineering, many tasks/activities are defined, including requirement, design, coding, testing, deployment and support, maintenance, configuration, and project management [84]. Software design (object oriented) is the stage in the software engineering process where the executable software system is developed. So, it plays a pivotal role in software development since it determines the structure of the software solution. Once the design has been implemented, it is difficult and expensive to change. Therefore, high design quality is vital for reducing software cost [62] [6] [49]. Below is a list of points signifying the importance of design [36]:

1. Design provides the basic framework that guides how the program codes are to be written and how personnel are to be assigned to tasks.
2. Design errors outweigh coding errors. They take more time to detect and correct, and are therefore costlier, than coding.
3. Design provides a basis for monitoring the progress.
4. A poorly designed software product is often unreliable, inflexible, inefficient, and not maintainable, because it is made up of an accumulation of uncoordinated, poorly tested, and, sometimes, undocumented pieces.
5. The larger the system and the larger the number of developers involved, the more important the design becomes.

Quality assurance plays an important role in monitoring software process (Software process means any software engineering activity in the development process i.e. analysis, design, or coding phase) in the form of umbrella activities (Umbrella activities are applied throughout a software project and help a software team manage and control progress, quality, change, and risk [47]) and in the form of measurement or metrics. Without measurements (or metrics), it is impossible to detect problems early in the software process, before they get out of hand. Metrics therefore can evaluate the process and serve as an early warning system for potential problems [56].

Many object oriented design metrics have been proposed specifically for the purpose of assessing the design of a software system suck as MOOD, CK (Chidamber and Kemrer) , Lorenz and Kids metrics [35]. Some of these metrics (or models) are supported by CASE tools due to their importance in evaluating or assessing the design of the software system.

Enterprise Architect is a well-known CASE tool that is used in over 130 countries for designing and constructing software systems. Enterprise Architect differentiates from other tools in that it supports a comprehensive UML modeling, have a built-in requirements management, test management, extensive project management support, Code engineering, and .. Etc. But Enterprise Architect does not support metrics on software design [65].

Any UML modeling tool uses XMI as a common format to save and load UML models. Without XMI, no UML model can be described or represented and no model details can be exchanged between different UML tools [64] [37].

1.2 Related Works

Many researchers have worked on object oriented design by means of quality assurance. Some of them propose tools that calculate metrics, other have made surveys about quality models, following a brief explanation about their works:

Dewar et al. (2002) [42] demonstrated the potential for deriving a suite of object-oriented design metrics by the XSLT (Extensible Style Sheet Language Transformation) processing of XMI representations of UML class diagram models. They propose a tool that extracts metrics like number of classes.

James M. Hogan et al.(2002) [9] proposed “Java Metrics Reporter” which is a tool that uses java source code as input to calculate metrics like cyclomatic complexity, polymorphism factor.

Moheb R. et al. (2009) [22] proposed a tool that automates the computation of the important metrics that are applicable to the UML class diagrams. The tool

collects information by parsing the XMI format of the class diagram, and then uses the data to calculate the metrics like CK, MOOD.

Poornima (2011) [45] stated that quality metrics are helpful for the designers in measuring solution architecture for better products. By understanding the solution domain of object oriented systems and measuring the quality of the design using metrics yields to future enhancements.

EvoJava is a tool for extracting static software metrics from Java source code repository, proposed by Oosterman et al. (2011) [41].

Rakesh Kumar and Gurvider Kaur (2011) carried out a comparative study on the complexity of object oriented design metrics proposed by Chidamber et al. (1994) [32].

Salem and Qureshi (2011) have discussed the inconsistencies in CK and Morris metrics in the literature with examples on complexity and cohesion metric of a project [52].

Mago and Kaur (2012) [34] proposed a model based on fuzzy logic which serves as an integrated means to provide an interpretation of the object oriented design metrics and also surveyed MOOD metrics with other metrics.

Rani et al. (2012) [48] proposed a tool (SD-Metrics) that measures the complexity of a class diagram using class metrics from XMI files from ArgoUML.

Kalia et al. (2012) [59] reviewed quality metrics suites namely, MOOD, CK and Lorenz & Kidd, selected some metrics and discarded others based on the definition and capability of the metrics.

Hilera and Fernández (2012) made a web service for calculating the metrics of UML class diagrams from XMI document. They stated that as UML becomes a standard format for specifying classes, it is useful to have a web service that quickly runs metrics on the diagram and gives developers feedback on the class quality [27].

Jassim and Altaan (2013), the main goal was to predict factors of MOOD metrics for object oriented design using a statistical approach. They also used a

linear regression model to find out the relationship between factors of MOOD and their influence on object oriented software measurements [30].

Sewisy et al. (2013) [4], proposed a hybrid metrics suite for evaluating the design of object oriented software early in UML design phase. A metrics extraction tool was developed which operated on UML design models and corresponded XMI files to assure independency results.

All studies state that design metrics are important to access the software design early in process and make changes that will improve the design. None of the above mentioned studies fully automate MOOD metrics. In this study, all MOOD metrics were fully automated and another model (MEMOOD) is used as an add-in inside Enterprise Architect (EA). None of the above studies integrates or improves an existing CASE tool.

1.3 Problem Description

Software design is very important stage in software engineering since it lies in the middle of the software development life cycle and costs can be reduced if corrections or improvements made in design phase. Some of the existing CASE tools do not have the ability to correct or improve software design like EA v7.5.

1.4 Objectives of the Research

The present study aims to:

1. construct a CASE tool that helps software engineers in design phase by assessing or evaluating the quality of that design using object oriented design metrics,
2. use two metrics models. MOOD (Metrics for Object Oriented Design) which measures Encapsulation, Inheritance, Polymorphism and Coupling, and MEMOOD (Maintainability Estimation Model for Object Oriented Systems in Design phase) which measures understandability, modifiability and maintainability, and
3. use the developed CASE tool as add-in to work inside Enterprise Architect since it has no support for design metrics. So, this study may be

considered as an evolvement of such a well-known CASE tool like the Enterprise Architect.

1.5 Thesis Layout

The chapters of this thesis are organized as follows:

Chapter Two (Software Engineering and Quality Assurance)

This chapter talks about software engineering and its relationship with quality assurance, measurement and metrics, and the importance of measurement. It also presents a detailed explanation about MOOD and MEMOOD models.

Chapter Three (UML and Enterprise Architect)

Reviews UML, class diagrams and their relationships, XMI, Enterprise Architect, and CASE tools integration.

Chapter Four (Analysis and Design of KDM, KRS and KDB Tool)

This chapter is about analyzing the proposed tools, algorithms and UML diagrams.

Chapter Five (Implementation and Testing of KDM, KRS and KDB Tools)

The way of using the proposed tools is explained in this chapter. A case study is presented followed by a discussion of the results.

Chapter Six (Conclusion and Suggestions for Future Work)

This chapter states the conclusion of this study and gives some suggestions for future work.

Appendix A (LINQ to XML and Lambda Expressions). This appendix reviews the techniques used to build the XMI parser.

Appendix B (JAVA Design Naming Conventions). This appendix describes the naming convention of java object oriented design.

Appendix C (Case Studies). This appendix represents the explanation of two case studies.

Appendix D (Sequence and Class Diagrams). This appendix contains sequence diagrams and class diagrams for the algorithms in chapter four.

Appendix E (MEMOOD Experiments). This appendix contains experiments of MEMOOD model with some conclusions about it.

Appendix F (Questionary Form). This appendix contains the form of the questionary used in this study.

Chapter Two

Software Engineering and Quality Assurance

Software Engineering and Quality Assurance

2.1 Software

Many people think that software is simply another word for computer programs. However, in software engineering, software is not just the programs themselves but also all associated documentation and configuration data that are required to make these programs operate correctly [62].

So the Definition of ***Software*** will be "Computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system" [29].

2.2 Software Engineering

According to [29], ***Software Engineering*** can be defined as the "application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software".

Also ***Software engineering*** is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification through to maintaining the system after it has gone into use. In this definition, there are two key phrases [62]:

1. Engineering discipline

   Engineers make things work. They apply theories, methods, and tools where these are appropriate. Also, they use them selectively and always try to discover solutions to problems even when there are no applicable theories and methods. Engineers also recognize that they must work to organizational and financial constraints so they look for solutions within these constraints.

2. All aspects of software production

   Software engineering is not just concerned with the technical processes of software development. It also includes activities such as software project management and the development of tools, methods, and theories to support software production.

2.3 Software Development Life Cycle (SDLC)

Building an information system using the SDLC follows a similar set of phases see figure (2-1): requirements phase, design phase, implementation phase, test phase, installation/checkout phase, and operation/maintenance phase [66], which are as follow:

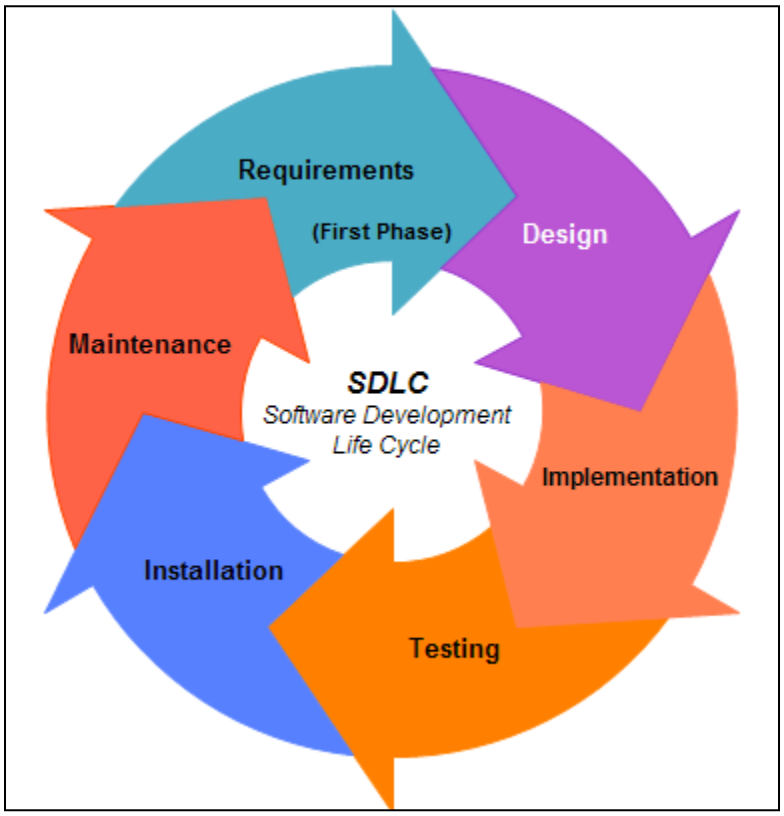


Figure (2-1) Software Development Life Cycle

1. **Requirements and Analysis Phase**

   Requirements capture is about discovering what to achieve with the new software. It has two aspects namely [39][66]:

   - *Business modeling* which, involves understanding the context in which the software will operate.
   - *System requirements modeling* (or functional specification) which means deciding what capabilities the new software will have and writing down those capabilities. It should be clear in the case of what

the software will do and what it will not do, so that the development does not veer off into irrelevant areas and then it will be evident if the software has been finished and whether it is successful or not.

Analysis means understanding what we are dealing with. Before a solution can be designed, some things should be verified like the relevant entities, their properties and their inter-relationships. This can involve customers and end users, since they are likely to be subject-matter experts [39].

2. **Design Phase**

Design phase, concerns the way a problem is solved. In other words, making decisions, based on experience, estimation and intuition, about what software will be written and how it will be deployed [39].

Also in the design phase, decisions are made about how the system will operate in terms of the hardware, software, and network infrastructure that will be in place; the user interface, forms and reports that will be used; the steps in the design phase determine exactly how the system will operate [82].

3. **Implementation Phase**

In this phase, the system is actually built; writing pieces of code that work together to form subsystems, which in turn collaborate to form the whole system. This phase usually gets most attention because for most systems it is the longest and most expensive single part of the development process [82][39].

4. **Test Phase**

In this phase of the software life cycle, components of a software product are evaluated and integrated based on whether or not software requirements have been satisfied; test data and associated test procedures exercise program paths to verify compliance to specified software requirements [66].

5. **Installation and Checkout Phase**

In this phase, the software product is integrated into an operational environment and tested in the environment to ensure that the software performs as required .This phase can be considered as "Deployment" since we are concerned with getting the hardware and software to the end users, along with manuals and training materials. This may be a complex process that involves a gradual, planned transition from the old way of working to the new [66][39].

6. **Operation and Maintenance Phase**

When the system is deployed, it has only just been born. A long life stretches before it, during which it has to stand up to everyday use. This is where the real testing happens.

Software Engineers (developers) are normally interested in maintenance because of the faults (bugs) that are found in the software. So, these faults must be found and removed as quickly as possible to keep the end user happy. Also, faults may enable users to discover deficiencies (things that the system should do but does not) and extra requirements (things that would improve the system) [39].

2.4 Object Orientation (OO)

The object-oriented approach was invented (or, rather, it evolved) because of the difficulties. People were trying to get good quality systems produced on time and within budget, especially for large systems with many people involved [39].

OO has taken the software world by storm, and rightfully so as a way of creating programs. OO has a number of advantages. It fosters a component-based approach to software development so that it begins by first creating a system by making a set of classes. Then the system is expanded by adding capabilities to components that are already built or by adding new components. Finally, the classes that are created in building the new system are reused, cutting down substantially on system development time [57].

Here are some of the justifications typically given for object orientation [39]:

1. **Objects are easier for people to understand;** because objects are derived from the business that software engineers are trying to automate, rather than being influenced too early by computer-based procedures or data storage requirements.
2. **Specialists can communicate better over time,** software industry has constructed career ladders that newcomers are expected to climb gradually as their knowledge and experience increase, with the object-oriented approach, and everyone is dealing with the same concepts and notations. Moreover, there are generally fewer concepts and fewer notations to deal with in the first place.
3. **Data and processes are not artificially separated;** in traditional methods, the data that needs to be stored are separated early from the algorithms that operate on that data and they are then developed independently. With object-oriented development, data and processes are kept together in small, easy-to-manage packages; data are never separated from the algorithms.
4. **Code can be reused more easily;** with the traditional approach, it starts with the problem that needs to be solved and allows that problem to drive the entire development, but with object-oriented development; software engineers are constantly looking for objects that would be useful in similar systems. Even when a new system has minor differences, it is much more likely to be able to change existing code to fit.

So for the reasons listed above; it can be said that OO systems are easier to change than systems developed using functional approaches. Objects include both data and operations to manipulate that data; they may, therefore, be understood and modified as standalone entities; changing the implementation of an object or adding services should not affect other system objects because objects are associated with things. Also, there is often a clear mapping between real-world entities (such as hardware components) and their controlling objects in the system. This improves the understandability, and hence the maintainability of the design [62].

UML plays a role in all of this by allowing software engineers to build object oriented systems that are easy-to-use and easy-to-understand models of objects [57].

2.4.1 Object Oriented Design (OOD)

OOD processes involve designing classes and the relationships between these classes. These classes define the objects in the system and their interactions. When the design is realized as an executing program, the objects are created dynamically from these class definitions [62].

The unique nature of OOD lies in its ability to build upon four important software design concepts namely abstraction, information hiding, functional independence, and modularity [46].So, by applying OOD, it allows to create software that is flexible to change and write with economy of expression [8].
Also it must be said that OOD is always about UML. So, any object oriented system will use UML in order to be developed or built.

2.5 Software Quality

Quality must be defined and measured if improvement is to be achieved. Yet, a major problem in quality engineering and management is that the term 'Quality' is ambiguous, so it is commonly misunderstood. The confusion may be attributed to several reasons. First, quality is not a single idea, but rather a multidimensional concept. Second, for any concept there are levels of abstraction; when people talk about quality, one party could be referring to it in its wide sense, whereas another might be referring to its specific meaning. Third, the term quality is a part of the daily language; the popular and professional uses of it may be very different [31]. So, just to be clear, ***Quality*** can be defined as [29]:

1. The degree to which a system, component, or process meets specified requirements.
2. The degree to which a system, component, or process meets a customer or user's needs or expectations.

2.5.1 Cost of Quality

The cost of quality includes all costs incurred in the pursuit of quality or in performing quality-related activities and the downstream costs of lack of quality. To understand these costs, an organization must collect metrics to provide a baseline for the current cost of quality, identify opportunities for reducing these costs, and provide a normalized basis of comparison [47].

The cost of quality can be classified into costs associated with prevention, appraisal (estimated), and failure [47].

1. ***Prevention costs*** include:
    a) Cost of management activities required to plan and coordinate all quality control and quality assurance activities.
    b) Cost of added technical activities to develop complete requirements and design models.
    c) Test planning costs, and the cost of all training associated with these activities.

2. ***Appraisal costs*** such as the cost of conducting technical reviews, Cost of data collection and metrics evaluation, and of testing and debugging.

3. ***Failure costs*** are those that would disappear if no errors appeared before or after shipping a product to customers. Failure costs may be subdivided into internal failure costs and external failure costs. Internal failure costs are incurred when an error is detected in a product prior to shipment. Internal failure costs include those required performing rework (repair) to correct an error and costs that occur when rework inadvertently generates side effects that must be mitigated while External failure costs are associated with defects found after the product has been shipped to the customer such as complaint resolution, product return and replacement and help line support. Also a poor reputation and the resulting loss of business form another external failure cost that is difficult to quantify but nonetheless very real. Bad things happen when low-quality software is produced [47].As expected, the relative costs to find and repair an error or defect increase dramatically

as we go from prevention to detection to internal failure to external failure costs. See figure (2-2) based on data collected by Boehm and Basili.

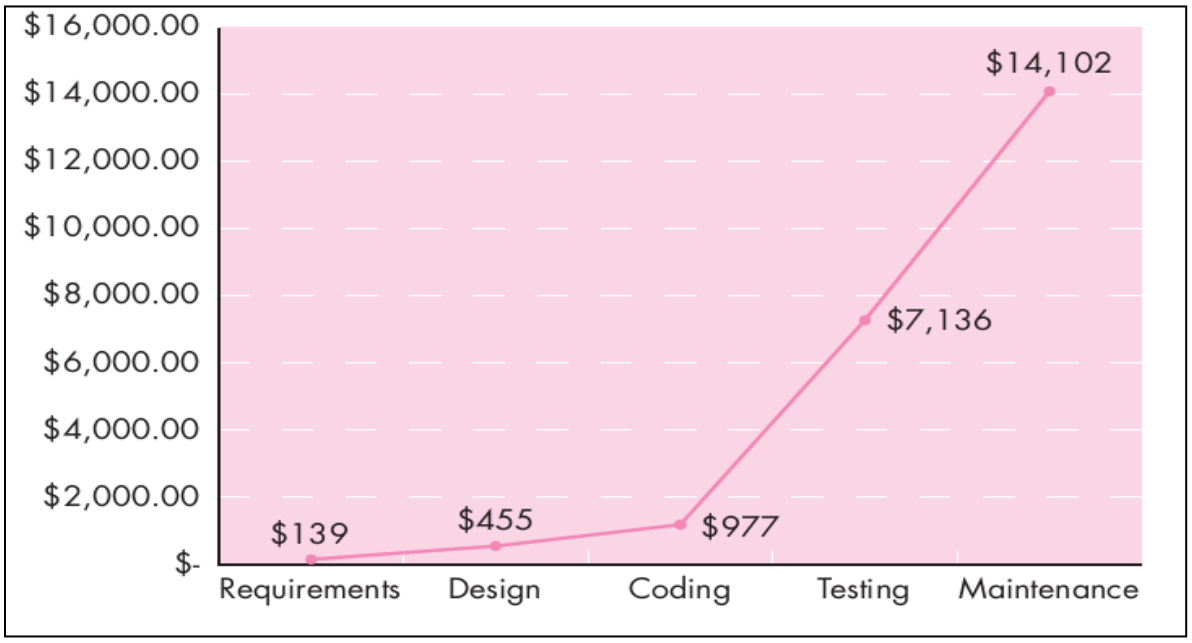


Figure (2-2) Relative cost of correcting errors and defects [47]

2.6 Measures, Metrics and Indicators

A key element of any engineering process is measurement. Using measures allows for better understanding of the attributes of the models that will be created and assessing the quality of the engineered products or systems to be built [47].

Measure is defined as a quantitative indication of the extent, amount, dimension, capacity, or size of some attribute of a product or process whereas ***Measurement*** is the act of determining a measure [47].

Metric is "a quantitative measure of the degree to which a system, component, or process possesses a given attribute [29].

When a single data point has been collected (e.g., the number of errors uncovered within a single software component), a measure has been established. Measurement occurs as the result of the collection of one or more data points.

Software metric relates the individual measures in some way (e.g., the average number of errors found per review) [47].

The software engineer collects measures and develops metrics so that ***Indicators*** will be obtained. An ***Indicator*** is a metric or combination of metrics that provides insight into the software process, a software project, or the product itself [47].

2.6.1 Importance of Measurement

There are four reasons for measuring software processes, products, and resources namely characterization, evaluation, prediction, and to improvement [46].

To **characterize** means to gain understanding of processes, products, resources, and environments. It also means to establish baselines for comparisons with future assessments.

To **evaluate** means to determine status with respect to plans. Measures are the sensors that allow the software development team or the quality assurance (QA) team to know when the projects and processes are drifting off track, so that it can be brought back under control. It also means to assess the achievement of quality goals and the impacts of technology and process improvements on products and processes.

To **predict** means to plan. Measuring for prediction involves gain understanding of relationships among processes and products and building models of these relationships, so that the values for some attributes can be used to predict others. This is done in order to establish achievable goals for cost, schedule, and quality so that appropriate resources can be applied.

To **measure** means to **improve** when gathering quantitative information that helps to identify roadblocks, root causes, inefficiencies, and other opportunities for improving product quality and process performance.

2.6.2 Benefits of Software Metrics

Software metrics can be very helpful during software development or after the software is completed. Some of the benefits of software metrics are [51]:

1. Software metrics can evaluate the quality of software and provide cost estimation of software project. So, it becomes easier to estimate and plan new activates based on them.
2. Software metrics can help to determine the effect of object technology, especially reusing technology applied in software development according to some quantitative evaluation such as productivity, quality, maintainability, reusability etc.
3. Software metrics can help to fully understand both the design and architecture information of the system. Additionally, it can help to understand the process of development by applying the process evaluation during software development.
4. Software metrics can also assist in the task of software testing.
5. Software metrics (Design) can aid to discover the underlying errors in software design at the early stage of software development life cycle.

   From all the benefits above, it can be said that better decisions are made when using software metrics and yet a better software.

2.6.3 Classification of Software Metrics

Software metrics can be classified into three categories namely product metrics, process metrics, and project metrics.

1. **Product Metrics**

   Product metrics describe the attributes of the software product at any phase of its development. Product metrics may measure the size of the program, performance, portability, maintainability, and product scale. Product metrics are used to measure the medium or the final product [14].

2. **Process Metrics**

Process metrics highlight the process of software development. It mainly aims at process duration, cost incurred and type of methodology used. Process metrics can also be used to improve software development and maintenance. Examples include the effectiveness of defect removal during development, and the response time of the fix process [31].

3. **Project Metrics**

Project metrics are used to monitor project situation and status. Project metrics prevent the problems or potential risks by calibrating the project and help to optimize the software development plan. Project metrics describe the project characteristics and execution. Examples include the number of software developers, the staffing pattern over the life cycle of the software, cost, schedule, and productivity [14].

2.7 OOD Metrics

OOD quality models (The phrases (Object Oriented Design Metrics), (Object Oriented Design Quality Models), or (Metrics) used in this thesis, mean the same thing as (metrics for OOD)) are means for judging OOD.OOD quality models work by establishing relationships between desirable design attributes, such as maintainability and reusability.

The quality of the software component, of any computerized information system, influences the quality of the information system, as a whole, up to a very large extent. Analogously, software design quality affects the software end product (i.e. implemented applications) quality extensively. So, there is an intense need to guarantee the quality of the design. To achieve a high quality design, there must be means to judge designs against established desirable attributes (e.g. maintainability). This judgment should be quantitative and objective to be applicable [15].

The reasons for not using conventional metrics for OO systems are [51]:

- Traditional software metrics aims at the procedure-oriented software development. So, it cannot fulfill the requirement of object oriented software.
- There are no metrics for the concepts like encapsulation, inheritance, coupling, cohesion etc. This means that conventional metrics cannot be used for OO paradigm.

Because of these limitations, a set of new software metrics adapted to the characteristics of object technology was proposed. Accordingly, metrics then became an essential part of object technology as well as good software engineering [51].

2.7.1 Benefits of OOD Metrics

In order to improve object oriented design, software measures or metrics are needed. The main objective of object oriented design metrics is to understand the quality of product to assess the effectiveness of the process and to improve the quality of work in the project. Application of design metrics in software development saves the time and money on redesign [61].

Another prominent benefit of utilizing object oriented design quality models is predicting estimations about the quality of the information system software component as a whole. In general, quality model consists of four parts [15]:

1. *Objectives*: which are the quality characteristics that the model claims to assess (e.g. Maintainability).
2. *Metrics*: which are a set of quantitative design properties, that should be computable for a given design diagram (e.g. depth of the inheritance tree).
3. *Relationships:* Ways to express the association between the objectives and the metrics (e.g. classes deep in the class's hierarchy are harder to maintain).
4. *(Optional) Thresholds:* which set upper/lower limits for metrics to fall within, to express a given trait (e.g. for better maintainability the Depth of inheritance tree should not exceed 6).

2.7.2 Metrics Hierarchy

Some of the software quality metrics for the development of the software development can be seen in figure (2-3).

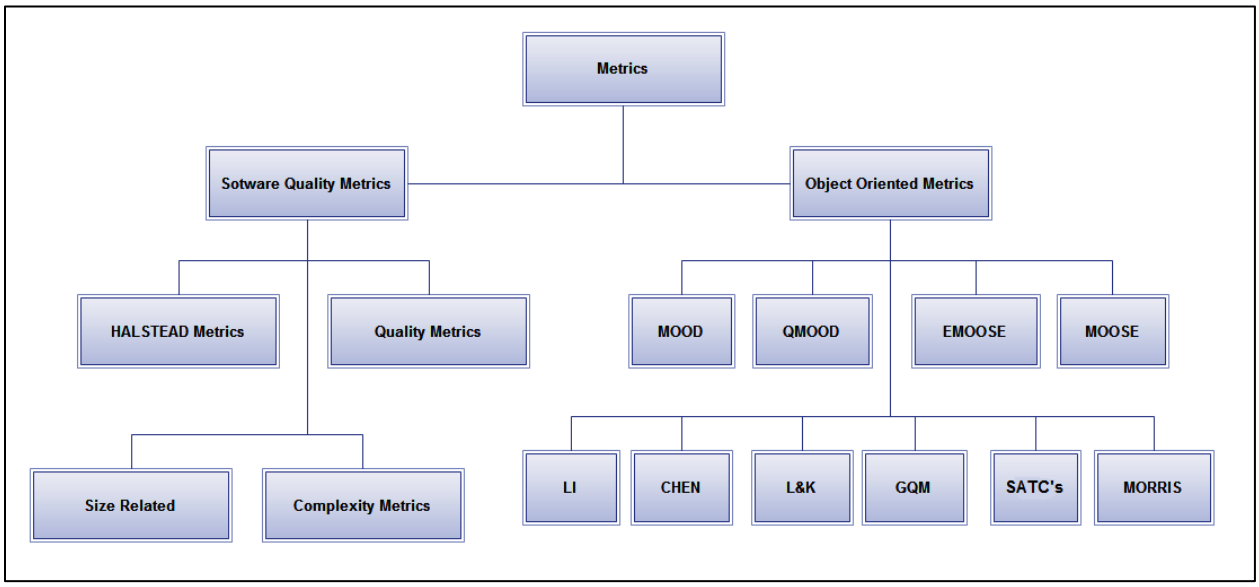


Figure (2-3) Metrics Hierarchy [58]

2.7.2.1 Software Quality Metrics

Software quality metrics can be classified into four categorizes [58]:

1. **Size Related**

   These are the metrics which can help to quantify the software size. There are three types of software metrics which are used to measure the software size [58] [53]:

   a) *Line of Code (LOC):* It is one of the earliest and simpler metrics for calculating the size of computer program. It is generally used in calculating and comparing the productivity of the programmers [21].

   b) *Function point Metrics (FP):* The metric can be used effectively as a means for measuring the functionality delivered by a system [47].

c) *Bang:* Dc. Marco defined bang as a function metrics. It can be calculated from a certain algorithm and data primitives available from the set of formal specification for the software and give the measures of total functionality delivered to the user.

2. Complexity Metrics

Cyclomatic Complexity (CC) is a metric value introduced by Thomas McCabe. It measures the number of possible paths through a source code item. It is widely considered a broad measure of soundness and confidence for a program, and also gives an indication of the minimum number of test cases that are necessary to properly test each item of the program [68].

3. Halstead Metrics

In1976, Halstead proposed the software science theory; the main aim of this software science theory is to find out the overall software production effort. Where it contains some vocabulary, length and volume.

4. Quality Metrics

In the quality metrics, there are some of the few quality metrics:

a) *Defect Metrics*: There is no effective procedure for counting the defects in the program, number of design change, number of intended errors and the error detected by the code inspections. The number of program test may be treated as an alternative measure to the defects.
b) *Reliability Metrics*: Internal product quality is usually measured by the number of bugs in the software or by how long the software can run before the encountering in a crash.
c) *Maintainability Index*: Dr. Paul W. Oman defined this metric to predict software maintainability.

2.7.2.2 OO Metrics

Some OO metrics for the OO software development are [58]:

Also see figure (2-3).

1. **Chen Metrics**
   Chen proposed software metrics, through which it can define "what is the behavior of the metrics in object-oriented design" such as class coupling metric and operating complexity metric.

2. **Morris Metrics**
   Morris proposed a metrics suite for the object oriented metrics systems and they define the system in the form of the tree structure.

3. **Lorenz & Kidd Metrics (L&K)**
   In their fundamental book about software quality named "Object Oriented Software Metrics" Lorenz and Kidd (1994) introduced many metrics to quantify software quality assessment. These metrics can be grouped in four categories namely size, inheritance, internal and external [6][58].

4. **Metrics for Object-Oriented Software Engineering (MOOSE)**
   This metric suite introduced by Chidamber and Kemerer, offers informative insight into whether developers are following object oriented principles in their design. They claim that using several of their metrics collectively help managers and designers to make better design decision [6].

5. **Extended Metrics for Object-Oriented Software Engineering (EMOOSE)**

   Li proposed this metrics of the MOOSE model. This model has three metrics (number of methods, data abstraction coupling and message pass coupling).

6. **Metrics for Object-Oriented Design (MOOD)**:
   These will be discussed in details, in paragraph (2.8) page (23).

7. **Goal Question Metrics (GQM)**
   This paradigm has been developed by Basili and Weiss as a technique for identifying meaningful metrics for any part of the software process [47].

8. **Quality Model for Object-Oriented Design (QMOOD)** is a comprehensive quality model that establishes a clearly defined and empirically validated model to assess OOD quality attributes such as understandability and reusability, and relates it through mathematical formulas [15].

9. **LI W. Metrics**
   Li proposed six metrics, viz. number of ancestor classes, number of local methods, class method complexity, number of descendent classes, coupling through abstract data type and coupling through message passing.

10. **SATC's Metrics**:
    Rosenberg Linda proposed selecting object oriented metrics that support the goal of measuring the code, quality, result. Many object-oriented metrics have been proposed due to lack of theoretical basis and that can be validated.

In order to use any OOD Quality Model, the model must be validated theoretically and empirically.

Empirical validation works, in general, by assessing the quality characteristics of a given design in two ways. One way is using the proposed model. The other way is using human judgment [15], using proven statistical and experimental techniques. In this way, the usefulness and relevance of any new metrics in the field can be assessed. The theoretical approach to the validation of metrics requires clarifying what attributes of software will be measured, and how to measure those attributes. A metric must measure what it purports to measure [26].

Models without validation are always in doubt concerning their correctness [15].

2.7.3 Used Models

In this thesis, two models for object oriented design are used. In the next section they will be discussed in details, and they are:

1. *MOOD* which stands for Metrics for Object Oriented Design, validated in [1][19][25].
2. *MEMOOD* which stands for Maintainability Estimation Model for Object Oriented software in Design phase, validated in [50].

The above models will be used because:

1. Both models are empirically and theoretically validated.
2. MOOD as will be discussed next is directly involved with OO concepts.
3. MEMOOD gives the ability to calculate three quality attributes namely understandability, modifiability and maintainability.

2.8 MOOD

The person who sets the MOOD metrics was Fernando B. Abreu [2]. MOOD refers to a structural model of the object oriented paradigm like *encapsulation* as (Method Hiding Factor (MHF) and Attribute Hiding Factor (AHF)), *inheritance* as (Method Inheritance Factor (MIF) and Attribute Inheritance Factor (AHF)), *polymorphism* as (Polymorphism Factor (POF)), and *message passing* as (Coupling Factor (CF)). Each of the metrics was expressed to measure where the numerator defined the actual use of any one of the feature for a particular design. In MOOD model, there are two main features, namely methods and attributes [6].

Attributes are used to represent the status of object in the system and methods are used to maintain or modify several kinds of status of the objects [58][6].

MOOD metrics are designed to meet a particular set of criteria. They were also proposed by the MOOD project team. The criteria are listed here [6][55]:

1. Non-size metrics should be system size independent.

2. Metrics should be dimensionless or expressed in some consistent unit system.

3. Metrics should be easily computable.

4. Metrics should be language independent.

2.8.1 Encapsulation

Encapsulation is the process of hiding all the details of an object that do not contribute to its essential characteristics. MHF and AHF were proposed together as measure of encapsulation. Both represent the average amount of hiding between all classes in the system [6].

1. Method Hiding Factor

This metric measures the invisibilities of the methods in the class. The invisibility of a method is the percentage of the total classes from which the method is not visible. The MHF is a fraction where the numerator is the sum of the invisibilities of all methods defined in all classes. The denominator is the total number of methods defined in the project. Methods should be encapsulated (hidden) within a class and not available for use to other objects. Method hiding increases reusability, but in other applications it decreases complexity and also reduces modifications to the code [60]. See equation (2.1).

$$MHF = \frac{\sum_{i=1}^{Tc}[\sum_{m=1}^{Md(Ci)}(1-V(Mmi))]}{\sum_{i=1}^{Tc} Md(Ci)} \quad \ldots\ldots\ldots (2.1)[55]$$

Where:

- TC = total number of classes. Summation occurs over i=1 to TC.
- Ci = class with index i (current class).
- Md (Ci) = the number of methods defined in class Ci
- V (Mmi) = Visibility value of a member (method or attribute), i.e. a value between 0-1 where public members = 1, private members = 0, and semi-public (e.g. protected) members are calculated as the number of classes that can access the member / total classes in the system (if working with different packages at the same time then the protected member is calculated. Otherwise it is considered the same as public in which it is equal to 1).

If the value of MHF is high (100%), it means all methods are private, they indicates very little functionality. Thus, it is not possible to reuse methods with high MHF.

MHF with low (0%) value indicates that all methods are public. This means that most of the methods are unprotected [55][6].

Standard value for MHF is in between (12.7% and 21.8) according to Fernando B. Abreu [2].

2. Attribute Hiding Factor

It measures the invisibilities of attributes in classes. The invisibility of an attribute is the percentage of the total classes from which the attribute is not visible. An attribute is called visible if it can be accessed by another class or object. Attributes should be "hidden" within a class by being declared as private. The numerator is the sum of the invisibilities of all attributes defined in all classes. The denominator is the total number of attributes defined in the project. It is desirable for the AHF to have a large value [60]. See equation (2.2).

$$AHF = \frac{\sum_{i=1}^{Tc}[\sum_{m=1}^{Ad(Ci)}(1-V(Ami))]}{\sum_{i=1}^{Tc} Ad(Ci)} \quad \ldots\ldots\ldots (2.2)[55]$$

Where:

- TC = total number of classes. Summation occurs over i=1 to TC.
- Ci = class with index i (current class).
- Ad (Ci) = the number of attributes defined in class Ci .
- V(Ami) is the same as V(Mmi) except it is for the attribute not for the method.

If the value of AHF is high (100%), it means all attributes are private. This indicates very little functionality. Thus it is not possible to reuse attributes with high MHF. If the value is low (0%), it indicates that all attributes are public and this means most of the attributes are unprotected [55][6].

The standard value for AHF is greater or equal to (75.2%) [2].

2.8.2 Inheritance

Inheritance is the process by which objects of one class acquire the properties of objects of another class. Inherited features in a class are those which are inherited and not overridden in that class. MIF and AIF are used to measure inheritance [6][54][4].

1. Method Inheritance Factor

MIF is defined as the ratio of the sum of the inherited methods in all classes of the system under consideration to the total number of available methods (locally defined plus inherited) for all classes [58]. See equation (2.3)[55].

$$MIF = \frac{\sum_{i=1}^{TC} Mi(Ci)}{\sum_{i=1}^{TC} Ma(Ci)} \quad \ldots\ldots\ldots (2.3)$$

Where:

- TC = total number of classes. Summation occurs over i=1 to TC.
- Ci = class with index i (current class).
- Mi is the number of inherited methods in Ci
- Ma is the number of available methods defined in Ci
- Md is the number of declared methods and not inherited in Ci
- Ma = Md + Mi of class Ci

Standard value for MIF is in between (66.4% and 78.5%) [2].

2. Attribute Inheritance Factor

AIF is defined as the ratio of the sum of inherited attributes in all classes of the system under consideration to the total number of available attributes (locally defined plus inherited) for all classes [58][6]. See equation (2.4)[55].

$$AIF = \frac{\sum_{i=1}^{TC} Ai(Ci)}{\sum_{i=1}^{TC} Aa(Ci)} \quad \ldots\ldots\ldots (2.4)$$

Where:

- TC = total number of classes. Summation occurs over i=1 to TC.
- Ci = class with index i (current class).
- Ai is the number of inherited Attributes in Class Ci
- Aa is the number of available attributes defined in class Ci
- Ad is the number of attributes declared in the class Ci
- Aa = Ad + Ai of class Ci

The standard value for AIF is in between (52.7% and 66.3%) [2].

2.8.3 Polymorphism

Polymorphism means the ability to take more than one form. Polymorphism is an important characteristic in object oriented paradigm and it measures the degree of overriding in the class inheritance tree [6][54].

1. Polymorphism Factor

POF represents the actual number of possible different polymorphic situations with respect to the maximum number of possible distinct polymorphic situations. POF is computed by dividing the total number of overridden methods in all classes by the result of multiplying the number of new methods times the number of descendants for all classes, respectively. The definition of POF means that it can only be applied to complete hierarchies [6][55]. See equation (2.5)[55].

$$POF = \frac{\sum_{i=1}^{Tc} Mo(Ci)}{\sum_{i=1}^{TC}[Mn(Ci)*DC(Ci)]} \qquad \ldots\ldots\ldots\ldots (2.5)$$

Where:

- TC = total number of classes. Summation occurs over i=1 to TC.
- Ci = class with index i (current class).
- Mo(Ci) is the overridden methods for Class Ci
- Mn(Ci) is the new methods defined in Class Ci
- DC(Ci) is the descendant counts (number of subclasses) for Class Ci .

The standard value for POF is in between (2.7% and 9.6%) [2].

2.8.4 Coupling

Coupling shows the relationship between models. A class is coupled to another class if it calls a method of another class [54].

1. Coupling Factor

CF of a class represents the percentage of couplings among classes, not computable to inheritance. CF is computed by dividing the number of associations, not related to inheritance between all classes by the number of classes squared minus the number of classes [6][54] . See equation (2.6)[55].

$$CF = \frac{\sum_{i=1}^{TC}[\sum_{j=1}^{TC} is\_Client(Ci,Cj)]}{TC^2 - TC} \qquad \ldots\ldots\ldots (2.6)$$

Here:

- TC = total number of classes. Summation occurs over i=1 to TC.
- Ci = class with index i (current class).
- is_client(Ci,Cj)=1 if Ci contains at least one non inheritance reference to a method or attribute of a class and Cj=0 otherwise.

This metric is intended to count all client-supplier relationships in a system. The two main relationships in an OO system are 'is -a' and 'has-a' relationships. The former describes relationships based on inheritance and the latter describes client – supplier relationships, e.g. one class's use of another class as an instance variable (association relationship)[6][55].

Pressman [46] argues that although many factors affect software complexity, understandability, and maintainability, it is reasonable to conclude that as "the CF value" increases, the complexity of object oriented design will also increase, and as a result, the understandability, maintainability, and the potential for reuse may suffer.

The standard value for CF is in between (4.0% and 11.2%) [2].

Table (2-1) summarizes the standard value for MOOD model, under 95% confidence interval.

Table (2-1) Standard intervals for MOOD Model [2]

Metrics	Minimum Value	Maximum Value
MHF	**12.7%**	**21.8%**
AHF	**75.2%**	**100%**
MIF	**66.4%**	**78.5%**
AIF	**52.7%**	**66.3%**
POF	**2.7%**	**9.6%**
CF	**4.0%**	**11.2%**

2.9 Maintainability Estimation Model for Object Oriented Software in Design Phase (MEMOOD)

The ever changing world makes maintainability a strong quality requirement for the majority of software systems. The maintainability measurement during the development phases of object oriented system estimates the maintenance effort. It also evaluates the likelihood that the software product will be easy to maintain. Despite the fact that software maintenance is an expensive and challenging task, it is not properly managed and often ignored. One reason for this poor management is the lack of proven measures for software maintainability [50].

Maintainability is defined as "the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment" [29].

As class diagrams play a key role in the design phase of object-oriented software, early estimation of their maintainability may help designers to incorporate required enhancements and corrections in order to improve their maintainability and consequently the maintainability of the final software to be delivered in future. Two quality attributes of class diagram, namely understandability and modifiability are focused to estimate their maintainability [50].

Metrics in table (2-2) have been selected for quantifying understandability and modifiability of class diagram. It had already been empirically validated that these metrics are correlated with understandability and modifiability of class diagram [69][10].

Table (2-2) Size and Structural Complexity Metrics for UML Class Diagrams

Metric Name	Metrics Definition
Number of classes (NC)	The total number of classes
Number of attributes (NA)	The total number of attributes
Number of methods (NM)	The total number of methods
Number of associations (NAssoc)	The total number of associations
Number of aggregation (NAgg)	The total number of aggregation relationships within a class diagram (each whole-part pair in an aggregation relationship)
Number of dependencies (NDep)	The total number of dependency relationships

Number of generalizations (NGen)	The total number of generalization relationships within a class diagram (each parent-child pair in a generalization relationship)
Number of aggregations Hierarchies (NAggH)	The total number of aggregation hierarchies (whole-part structures) within a class diagram
Number of generalizations Hierarchies (NGenH)	The total number of generalization hierarchies within a class diagram
Maximum depth of inheritance (MaxDIT)	It is the maximum of the DIT (Depth of Inheritance Tree) values obtained for each class of the class diagram. The DIT value for a class within a generalization hierarchy is the longest path from the class to the root of the hierarchy
Maximum aggregation hierarchy (MaxHAgg)	It is the maximum of the HAgg values obtained for each class of the class diagram. The HAgg value for a class within an aggregation hierarchy is the longest path from the class to the Leaves.

In order to calculate the Maintainability Estimation model see figure (2-4) [50], Both the Understandability and the Modifiability of the design are used.

Understandability in our context means the extent of users (software engineer or programmer) capability with different backgrounds to understand the software design. Understandability of the design can be calculated as in equation (2.7)[50].

$$Understandability = 1.166 + 0.256 * NC - 0.394 * NGenH \quad \ldots. (2.7)$$

Where:

- NC is the total number of classes
- NGenH is the number of generalization hierarchies.

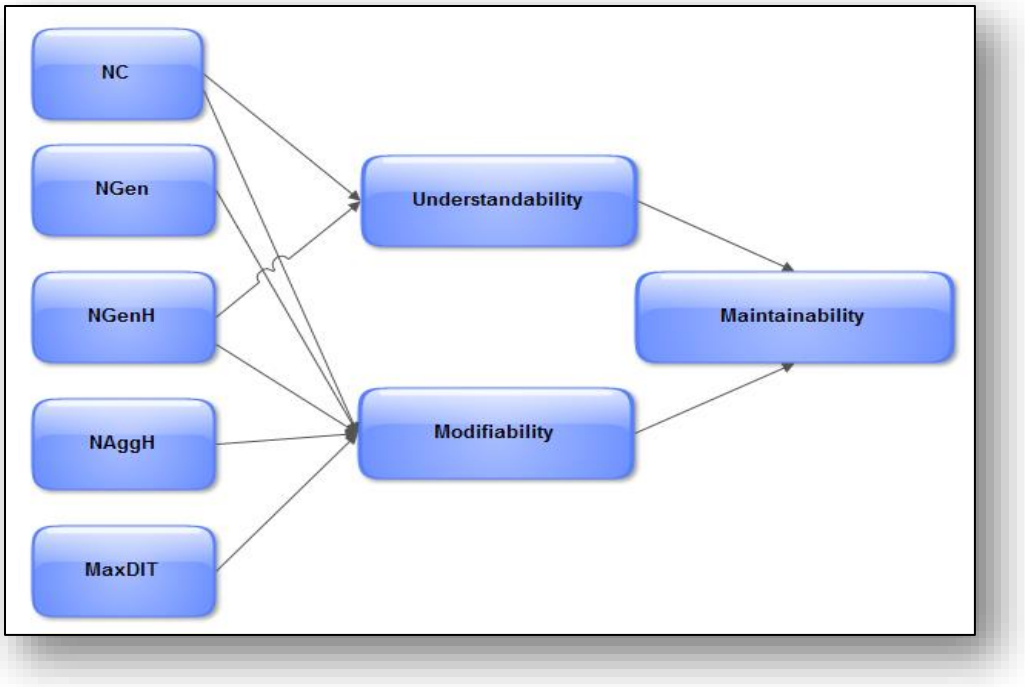


Figure (2-4) Maintainability Estimation Model (MEMOOD) [50]

Modifiability in our context is the capability to modify the design without affecting the overall system. See equation (2.8)[50].

$$Modifiability = 0.629 + 0.471 * NC - 0.173 * NGen - 0.616 * NAggH - 0.696 * NGenH + 0.396 * MaxDIT \quad \ldots\ldots\ldots \text{(2.8)[50]}$$

Where:

- NC is the total number of classes
- NGen is the number of generalization relationship (inheritance relationship between super class and sub class)
- NAggH is the number of aggregation relationship hierarchies
- NGenH is the number of generalization hierarchies in the design
- MaxDIT is the maximum depth of the inheritance in the design

After calculating understandability and modifiability quality attributes it is possible now to find the maintainability of software design. See equation (2.9)[50].

$$Maintainability = -0.126 + 0.645 * Understadability + 0.502 * Modifiability \quad \ldots\ldots \quad (2.9)$$

The values of understandability, modifiability and maintainability are of immediate use in the software development process. These values may help software designers to review the design and take appropriate corrective measures, early in the development life cycle, in order to control or at least reduce future maintenance cost [50].

Chapter Three

UML and Enterprise Architect

UML and Enterprise Architect

3.1 UML

A modeling language can be made up of pseudo-code, actual code, pictures, diagrams, or long passages of description. In fact, it is anything that can help in describing the system. The elements that make up a modeling language are called its notation [24].

So ***UML*** can be defined as a family of graphical notations that help in describing and designing software systems, particularly software systems built using the OO style [16].

The primary goals of the design of the UML are [12]:

1. It provides users with a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models.
2. It provides extensibility and specialization mechanisms to extend the core concepts. For example, the UML provides stereotypes which allow new elements to be defined by extending and refining the semantics of existing elements. A stereotype is enclosed in double chevrons (<< … >>).
3. It is independent of particular programming languages and development processes.
4. It provides a formal basis for understanding the modeling language.
5. It supports higher-level development concepts such as collaborations, frameworks, patterns, and components.
6. It integrates best practices.

3.1.1 History of UML

The UML is the brainchild of Grady Booch, James Rumbaugh, and Ivar Jacobson. They were called "the Three Amigos", and they worked in separate organizations through the 1980s and early 1990s, each devising his own methodology for OO analysis and design. Their methodologies achieved preeminence over those of numerous competitors. By the mid-1990s, they began to borrow ideas from each other. So, they decided to evolve their work together. In

1994 Rumbaugh joined Rational Corporation, where Booch was already working. Jacobson enlisted at Rational a year later and UML was born [57].

In 1997, UML 1.0 was submitted to the Object Management Group (OMG), a nonprofit consortium involved in maintaining specifications for use by the computer industry. UML versions can be as the following [47] [78]:

1. UML 1.0 was revised to UML 1.1 and adopted later that year.
2. UML 2.0 was adopted by the OMG in 2005.
3. Versions 2.1.1 and 2.1.2 appeared in 2007.
4. UML version 2.2 in February 2009.
5. UML 2.3 was formally released in May 2010.
6. UML 2.4.1 was formally released in August 2011.
7. UML 2.5 was released in October 2012 as an "In process" version and has yet to become formally released.

3.1.2 Ways of Using UML

UML can be used according to the ways that people tend to use UML [16][24], and as follows:

1. **UML as a sketch** UML is used to make brief sketches to convey key points. These are throwaway sketches that could be written on a whiteboard or on any piece of paper.
2. **UML as a blueprint** a detailed specification of a system with UML diagrams is provided. These diagrams would not be disposable but would be generated with a UML tool.
3. **UML as a programming language i**n this environment, developers draw UML diagrams that are compiled directly to executable code, and the UML becomes the source code obviously. This usage of UML demands particularly sophisticated tooling.

The approach used depends on the type of application that will be built, and how rigorously the design will be reviewed. So, whether developing a software system, and, if it is software, the software development process that will be used, also UML can be used in certain industries, such as medicine and defense. Software

projects tend to lean toward UML as a blueprint because a high level of quality is demanded [24].

3.1.3 Reasons of UML Modeling

The benefits of using UML as a modeling tool can lead to both higher productivity and also better quality of overall system.

UML is necessary because before the advent of UML, system development was often a hit-or-miss proposition. So, system analysts would try to assess the needs of their clients and generate a requirement analysis in some notation that the analyst understands (but not always the client). And because system development involves communication among people, the potential for error lurked at every stage of the process. The analyst may misunderstand the client and produce a document the client cannot comprehend. Thus, the results of the analysis may not be clear to the programmers who subsequently may create a program that is difficult to use and does not solve the client's original problem [57]. So, here comes the UML which allows for better understanding and better communication along either the development team or with the customer. See figure (3-1) which shows the benefits of UML modeling.

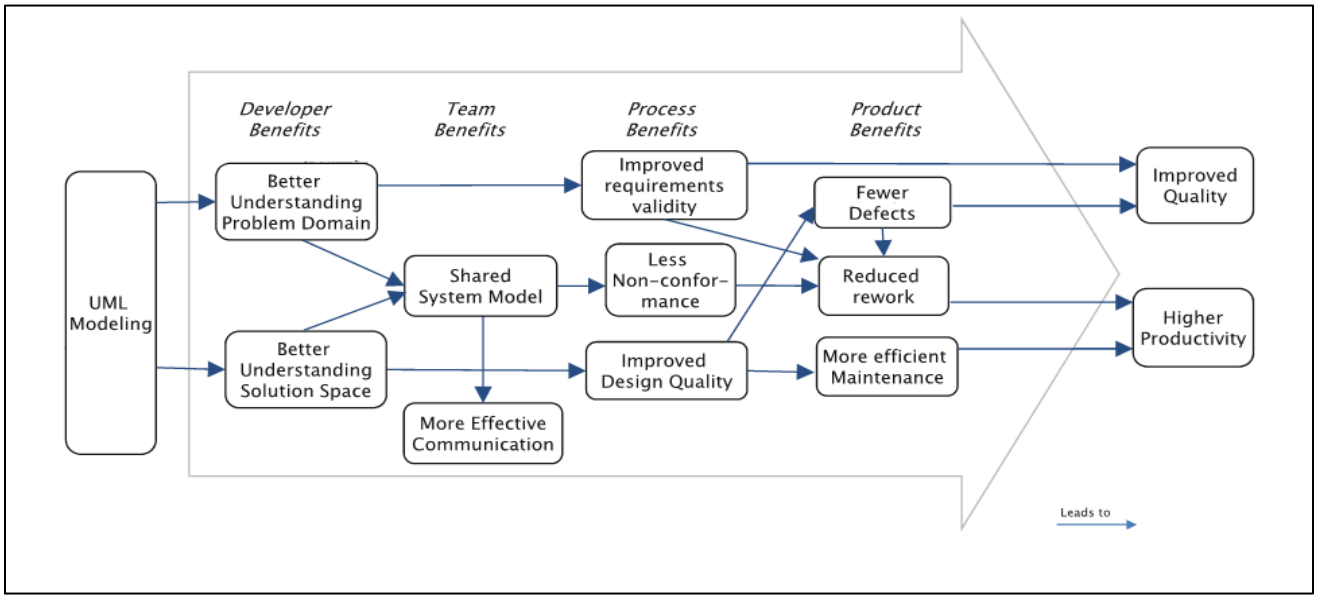


Figure (3-1) The benefits of UML modeling [38]

3.2 UML Diagrams

UML diagrams fall into two major groupings: one for modeling the structure of a system and one for modeling behavior.

Structure diagrams are used for representing the data and static relationships that are in the information system. ***Behavior diagrams*** provide the analyst with a way to describe the dynamic relationships among the instances or objects that represent the business information system. They also allow the modeling of dynamic behavior of individual objects throughout their lifetime. Furthermore, they support the analyst in modeling the functional requirements of an evolving information system [82]. Table (3-1) shows UML diagrams along with their using.

Table (3-1): UML diagram summary [82]

Diagram Name	Used to	Primary Phase
Structureal Diagrams		
Class	Illustrates the relationships between classes modeled in the system	Analysis, Design
Object	Illustrates the relationships between objects modeled in the system	Analysis, Design
Package	Groups other UML elements together to form higher level constructs	Analysis, Design, Implementation
Deployment	Shows the physical architecture of the system. Also shows software components being deployed onto the physical architecture	Physical Design, Implementation
Component	Illustrates the physical relationships among the software components	Physical Design, Implementation
Composite Structure	Illustrates the internal structure of a class—i.e., the relationships among the parts of a class	Analysis, Design
Profile	Auxiliary UML diagram which allows defining custom stereotypes, tagged values, and constraints [80]	Analysis, Design, Implementation
Behavioral Diagrams		
Activity	Illustrates business work flows independent of classes, the flow of activities in a use case or detailed design of a method	Analysis, Design
Sequence	Models the behavior of objects within a use case. Focuses on the time-based ordering of an activity	Analysis, Design
Communication	Models the behavior of objects within a use case. Focuses on the communication among a set of collaborating objects of an activity	Analysis, Design

Interaction Overview	Illustrates an overview of the flow control of a process	Analysis, Design
Timing	Illustrates the interaction that takes place among a set of objects and the state changes that they go through along a time axis	Analysis, Design
State Machine	Examines the behavior of one class	Analysis, Design
Use Case	Captures business requirements for the system and illustrates the interaction between the system and its environment	Analysis

The focus will be only on class diagram because all available metrics are applicable to class diagram which will be discussed in detail.

3.2.1 Class Diagram

Class diagram is a static model that shows the classes and the relationships among classes that remain constant in the system over time. The class diagram depicts classes, which include both behaviors and states, with the relationships between the classes [13].

3.2.1.1 Class Diagram Elements

The main building block of a class diagram is the *class* which stores and manages information in the system during analysis. Classes refer to the people, places, events, and things about which the system will capture information. Later, during design and implementation, classes can refer to implementation-specific artifacts like windows, forms, and other objects used to build the system [13].

Each class is drawn by using three part-rectangles with the class's name at the top, attributes in the middle, and methods (also called operations) at the bottom [82]. See figure (3-2).

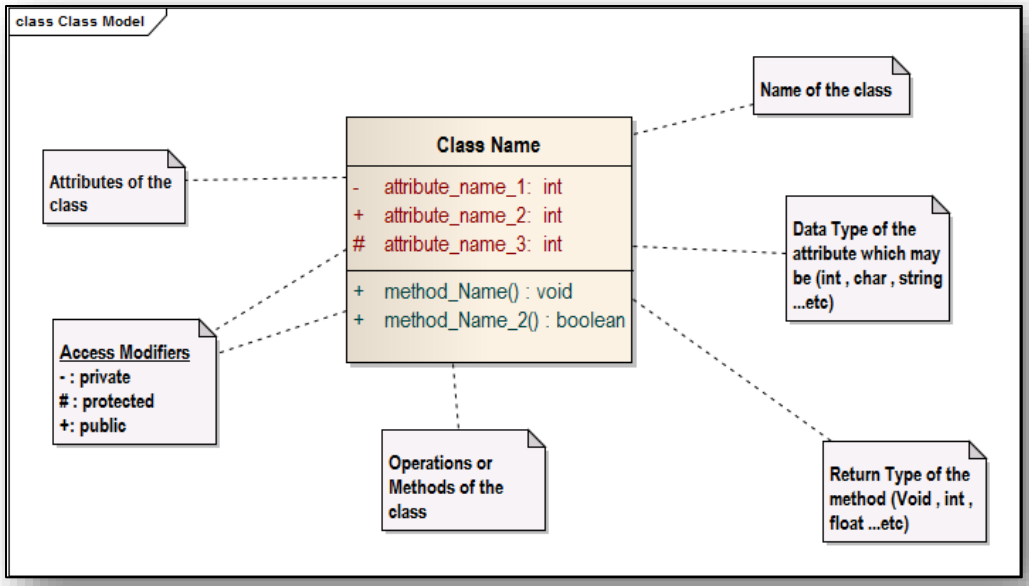


Figure (3-2) Class Diagram Syntax

Any Class Diagram consists of a number of class elements and relationships between those elements [82].

1. ***Class Element***

   Class element can represent a kind of person, place, or thing about which the system must capture and store information. The class element has the following properties:

   1. A name typed in bold and centered in its top section.
   2. A list of attributes in its middle section.
   3. A list of operations in its bottom section.
   4. Implicit operations available to all classes.

   Also class elements have *Attributes* which represent properties that describe the state of an object, whereas the *Methods* have the following properties

   1. Representing the actions or functions that a class can perform.
   2. Can be classified as a constructor, query, or update operation.
   3. Including parentheses that may contain special parameters or information needed to perform the operation.

Each member (Attribute or Method) of the class element has a visibility. Visibility of any member relates to the level of information hiding to be enforced for that member. The visibility of a member can either be public (+), protected (#), or private (-). A public member is one that is not hidden from any other object. As such, other objects can modify its value. A protected member is the one that is hidden from all other classes except its immediate subclasses. A private member is one that is hidden from all other classes. Normally, the default visibility for any member is private [82].

2. *Relationships*

A primary purpose of the class diagram is to show the relationships or associations that classes have with one another. These relationships may contain multiplicity symbols, which represent the minimum and maximum times a class instance can be associated with the related class instance [13]. There are many of relations that could be seen in class diagram, namely [57]:

a) Association

When classes are connected together conceptually, that connection is called an association. For instance, a class named “Player” has an association with a class named “Team”, see figure (3-3).

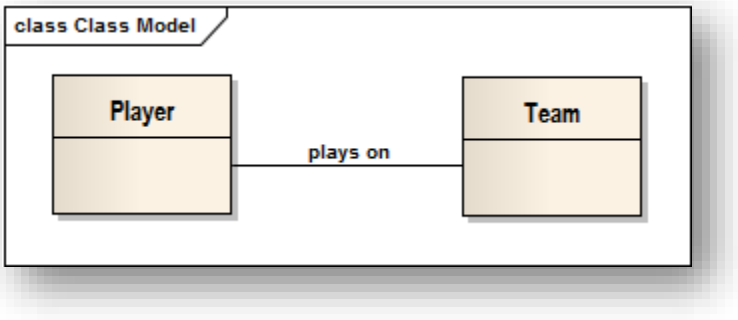


Figure (3-3) Association relationship

b) Generalization

This relationship indicates inheritance where one class (the child class or subclass) can inherit the attributes and the operations from another (the

parent class or super class). The parent class is more general than child class. See figure (3-4).

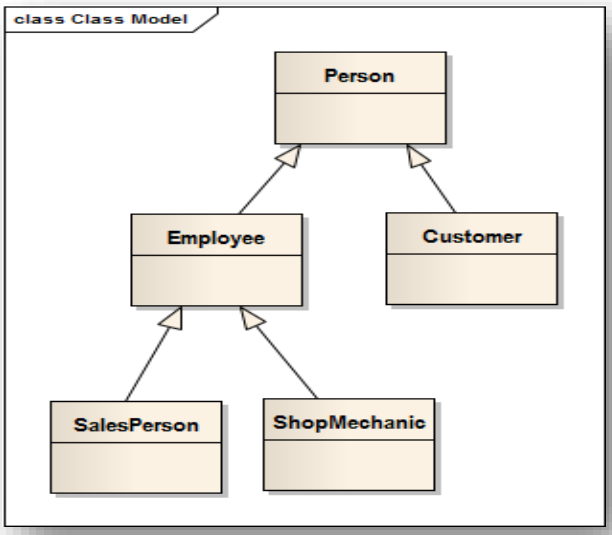


Figure (3-4) Generalization hierarchy example

c) Aggregation

Sometimes a class consists of a number of component classes. This is a special type of relationship called an aggregation. The components and the class they constitute are in a part-whole association. For example, a computer system is an aggregation that consists of a CPU box, a keyboard, a mouse, a monitor, a CD-ROM drive, … etc. See figure (3-5).

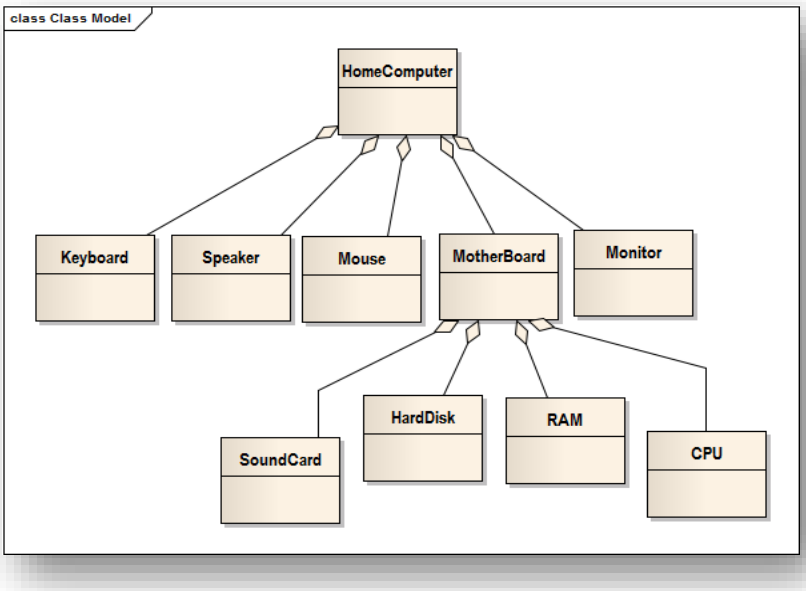


Figure (3-5) Aggregation relationship

d) Composition

A composite is a strong type of aggregation. Each component in a composite can belong to just one whole. For example, the components of a coffee table: the tabletop and the legs make up a composite see figure (3-6).

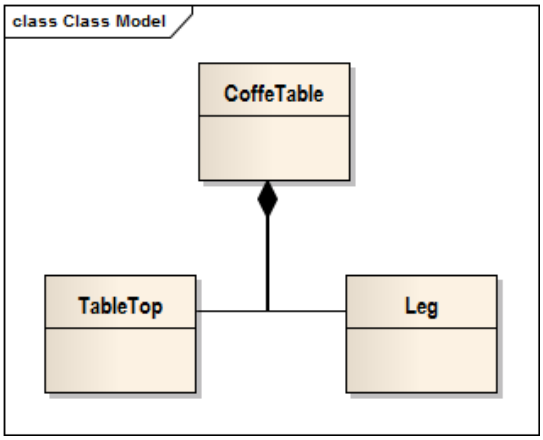


Figure (3-6) Composition relationship

There are also a number of other relationships (i.e. dependency relationship and realization relationship) but they are not needed in our context, so they will not be discussed.

3.3 XML Meta Interchange (XMI)

XML stands for (Extensible Markup Language) which is a simple language for data storage and a good cross-platform technology [33]. Hence, it is also a universal format for representing data on the World Wide Web [23].

XMI is an open standard file format that enables the interchange of model information between models and tools. XMI is defined by the OMG and is based on XML [64].

XMI provides metadata information exchange for programmers and other users. The purpose of XMI is to help exchange data model for programmers using UML and the different languages and development tools [33].

XMI is used in this work as an input to one of the proposed tools (KDM tool). In addition to that, XMI is a way of saving UML diagrams as XML which is the main reason for using it as an input.

3.3.1 Benefits of XMI

The primary benefits of XMI are as follows [23]:

1. XMI provides a standard representation of objects in XML, thus enabling the effective exchange of objects using XML.
2. XMI specifies how to create XML schemas from models (XML Schemas also called XSD (XML Schema Definition)). It is a way of describing the structure and constraints on a class of XML documents.
3. XMI enables to create simple XML documents and make them more advanced as the applications evolve.

3.3.2 Importance of XMI

XMI is necessary because XML is not OO. So, it is necessary to map objects to XML and there is more than one way to do this mapping due to the flexibility of XML. However, the flexibility is also a drawback when it comes to exchanging XML documents. If one tool maps objects to XML one way, and another tool maps objects to XML a different way, it is unlikely that the two tools will be able to properly interpret each other's XML documents. A tool that uses XMI can exchange objects with other tools that also use XMI [23].

3.4 CASE Tools

CASE stands for (Computer Aided Software Engineering). ***CASE tools*** are computerized software development tools that support the developer when performing one or more phases of the software life cycle and/or support software maintenance. The definition's generality allows compilers, interactive debugging systems, configuration management systems and automated testing systems to be considered as CASE tools. In other words, well-established computerized software development support tools (such as interactive debuggers, compilers and project progress control systems) can readily be considered *classic* CASE tools, whereas

the new tools that support the developer for a succession (series) of several development phases of a development project are referred to as *real* CASE tools [17].

CASE provides the software engineer with the ability to automate manual activities and to improve engineering insight. Like computer-aided engineering and design tools that are used by engineers in other disciplines, CASE tools help to ensure that quality is designed in before the product is built [46].

The difference between CASE tool and tool, is that the tool is any program that cannot be used in any software engineering activity (i.e. paint, calculator …etc).

3.4.1 Categories of CASE Tools

CASE tools are divided into two categories namely Vertical and Horizontal CASE tools. *Vertical CASE tools* are the tools that provide support for certain activities within a single phase of the software life cycle, while *Horizontal CASE Tools* are those tools which support the automated transfer of information between the phases of a life cycle. These tools include project management, configuration-management tools, and integration services.

The above two categories of CASE tools can further be divided into the following [3][47]:

1. Upper or Front-end CASE Tools

   These CASE tools are designed to support the analysis and design phases of the SDLC. All analysis, design, and specification tools are front-end tools. These tools also include computer aided diagramming tools oriented toward a particular programming design methodology, and more recently including OOD.

   The general types of Upper CASE tools are listed below:

   a) Diagramming tools: tools that enable system process, data, and control structures to be represented graphically. They strongly support analysis and documentation of application requirements.
   b) Form and report generator tools: They support the creation of system forms and reports in order to show how systems will “look and feel” to users.

c) Analysis tools: tools that enable automatic checking for incomplete, inconsistent, or incorrect specifications in diagrams, forms, and reports.

2. Lower or Back-end Tools

These CASE tools are designed to support the implementation and maintenance phases of the SDLC. All generator, translation, and testing tools are back-end tool. See figure (3-7).

3. Cross life-cycle CASE or Integrated Tools

These CASE tools are used to support activities that occur across multiple phases of the SDLC. While such tools include both front-end and back-end capabilities, they also facilitate design, management, and maintenance of code. In addition, they provide an efficient environment for the creation, storage, manipulation, and documentation of systems.

4. Reverse-engineering Tools.

These tools build bridges from lower CASE tools to upper CASE tools. They help to understand the internal design structure of complex program. In most cases, reverse engineering tools accept source code as input and produce a variety of structural, procedural, data, and behavioral design representations.

3.4.2 Benefits of CASE Tools

The benefits of using CASE tools are numerous. Some of them are [82][3]:

1. Tasks are much faster to complete and alter.
2. Development information is centralized; and information is illustrated through diagrams, that are typically easier to understand.
3. Reduction of maintenance costs.
4. Improved coordination among staff members who are working on a large software project.
5. An increase in project control through better planning, monitoring, and communication.

Whereas the benefits of CASE tools in this work are:

1. Improving software design quality and assessing overall quality by calculating metrics.
2. Helping software engineers in documentation.
3. Reducing or eliminating rework.
4. Helping software engineer by following naming conventions.
5. Storing metrics value to be used as a historical data in order to be compared with other similar projects in the future.

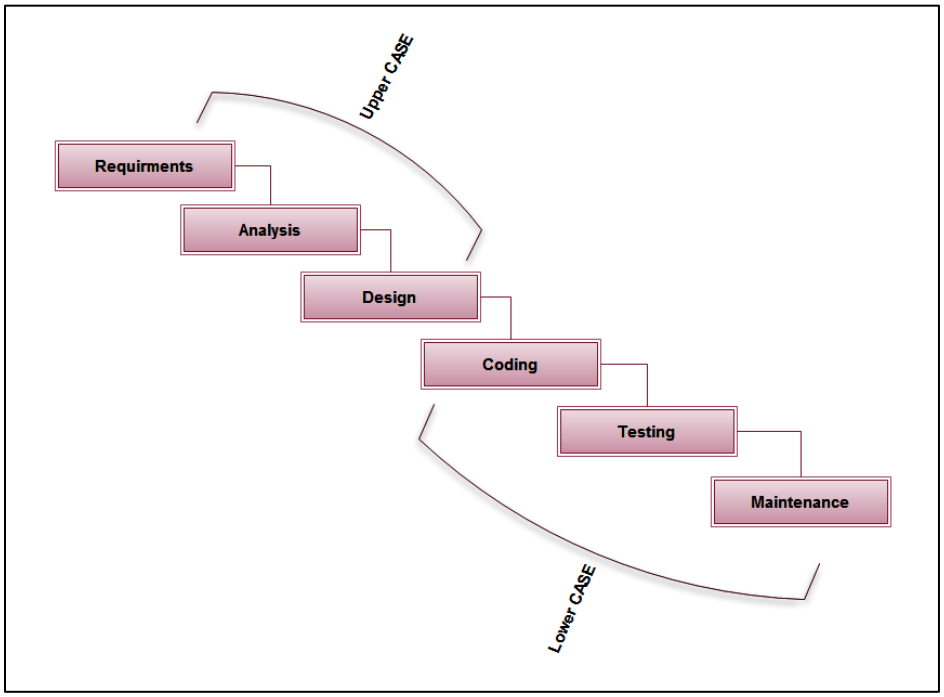


Figure (3-7) Upper CASE and Lower CASE tools

3.5 Enterprise Architect (EA)

EA is a CASE tool for designing and constructing software systems, for business process modeling, and for more generalized modeling purposes [79][74].

EA was developed by Sparx Systems © and it covers all aspects of the software development cycle from requirements gathering, through analysis, model design, testing, change control and maintenance to implementation, with full traceability (identifies the way a given process has been, or is to be, developed in a system).

Also EA is a multi-user, visual tool with a great feature set, helping analysts, testers, project managers, quality control staff and deployment staff to build and document robust, maintainable systems and processes [64].

EA can be downloaded from the official website of Sparx Systems. See [79].

3.5.1 Importance of EA

EA has proven to be highly popular across a wide range of industries and is used by thousands of companies worldwide, from large, well known, multinational organizations to smaller independent companies and consultants [64].

Sparx Systems © software is used in the development of many kinds of applications and systems in a wide range of industries, including aerospace, banking, web development, engineering, finance, medicine, military, research, academia, transport, retail, utilities (such as gas and electricity) and electrical engineering. It is also used effectively for UML and enterprise architecture training in many prominent colleges, training companies and universities around the world [64].

3.5.2 Uses of EA

EA can be used in the [64][75]:

1. *Design and Building of Diverse Systems Using UML*

   EA enables using the leverage of the full expressive power of UML to model, design and build diverse systems in an open and well understood manner.

2. *Sharing Models*

   EA enables to share complete models or specific aspects of a model between members of a team.

3. *Modeling and Managing Complexity*

   EA helps individuals, groups and large organizations to model and manage complex information. Often this relates to software development

and information technology system design and deployment, but it can also relate to business analysis and business process modeling.

4. *Linking EA to Integrated Development Environment*

Using Sparx Systems, Model Driven Generation (MDG) Link plug-ins, which enable programmers to develop source code in any Integrated Development Environment (IDE) such as Visual Studio .NET or Eclipse.

5. *Customizing EA*

EA also includes a Software Developers' Kit (SDK) that enables experienced tool developers to customize and extend EA to suit the specific requirements of their organization, for example Add-Ins. Also, the very detailed Automation Interface gives the leverage to access to most element features and major functions, such as XMI import/export and attached information.

6. *Other uses*

EA can be used in modeling, managing and tracing requirements, generating documentation, modeling databases, etc.

For all those reasons mentioned earlier in addition to the powerful description of UML class diagrams as XMI, this study tends to use EA as a platform for the proposed tools to work with.

3.5.3 EA Add-In Model

The EA Add-In Model enables to build a new functionality into EA; thus creating mini programs that can extend the capabilities of EA, defining menus, and also creating custom views [65][64].

Add-Ins enable to add more functionality to EA. The EA Add-In model builds on the features provided by the Automation Interface which enables to extend the EA user interface [64].

Automation Interface means a way of other applications to access the information in an EA model using Windows OLE (Object Linking and Embedding) Automation (later renamed simply "Automation") which makes it possible for one application to manipulate objects implemented in another application or to expose objects; so they can be manipulated [76][64]).

Add-Ins have several advantages [64]:

1. Add-Ins can define EA menus and sub-menus.
2. Add-Ins receive notifications about various EA user interface events including menu clicks and file changes.
3. Add-Ins can (and should) be written as in-process Dynamic Link Library (DLL) components. This provides lower call overhead and better integration into the EA environment.
4. Add-In interface is flexible.

3.6 DLL

DLL is an executable file that acts as a shared library of functions. See figure (3-8). Dynamic linking provides a way for a process to call a function that is not part of its executable code. The executable code for the function is located in a DLL, which contains one or more functions that are compiled, linked, and stored separately from the processes that use them. DLLs also facilitate the sharing of data and resources. Multiple applications can simultaneously access the contents of a single copy of a DLL in memory [70].

Figure (3-8) DLL icon.

Dynamic linking differs from static linking in that it allows an executable module (either a .dll or .exe file) to include only the information needed at run time to locate the executable code for a DLL function. In static linking, the linker gets all

of the referenced functions from the static link library and places it within the code into the executable file [70].

3.6.1 Differences between Applications and DLLs

Even though DLLs and applications are both executable program modules, they differ in several ways. To the end user, the most obvious difference is that DLLs are not programs that can be directly executed. From the system's point of view, there are two fundamental differences between applications and DLLs [71]:

1. An application can have multiple instances of itself running in the system simultaneously, whereas a DLL can have only one instance.
2. An application can own things such as a stack, global memory, file handles, and a message queue, but a DLL cannot.

3.6.2 Advantages of Using DLLs

DLL has the following advantages [72]:

1. It saves memory and reduces swapping. Many processes can use a single DLL simultaneously, sharing a single copy of the DLL in memory. In contrast, Windows must load a copy of the library code into memory for each application that is built with a static link library.
2. It saves disk space. Many applications can share a single copy of the DLL on disk.
3. Upgrades to the DLL are easier. When the functions in a DLL change, the applications that use them do not need to be recompiled or re-linked as long as the function arguments and return values do not change. In contrast, statically linked object code requires that the application be re-linked when the functions change.
4. It supports Multilanguage programs. Programs written in different programming languages can call the same DLL function as long as the programs follow the function's calling convention.

3.7 Component Object Model (COM)

Microsoft COM technology in the Microsoft Windows-family of operating systems enables software components to communicate. COM is used by developers to create re-usable software components, link components together to build applications, and take advantage of Windows services. COM objects can be created with a variety of programming languages. OO languages, such as C++ or C#, provide programming mechanisms that simplify the implementation of COM objects [73].

3.8 Windows Registry

Windows Registry is a central repository or hierarchical database of configuration data for the operating system and most of its programs. Windows Registry Editor can be used to access Windows Registry [83].
Windows Registry is used to store information that is necessary to configure the system for one or more users, applications and hardware devices. The Registry contains information that Windows continually references during operation, such as profiles for each user, the applications installed on the computer and the types of documents that each can create, property sheet settings for folders and application icons, what hardware exists on the system, and the ports that are being used [77].
Windows Registry Editor can be started by using the "run" command to run the "regedit.exe" file. Figure (3-9) shows the Windows Registry Editor when it is started [83].

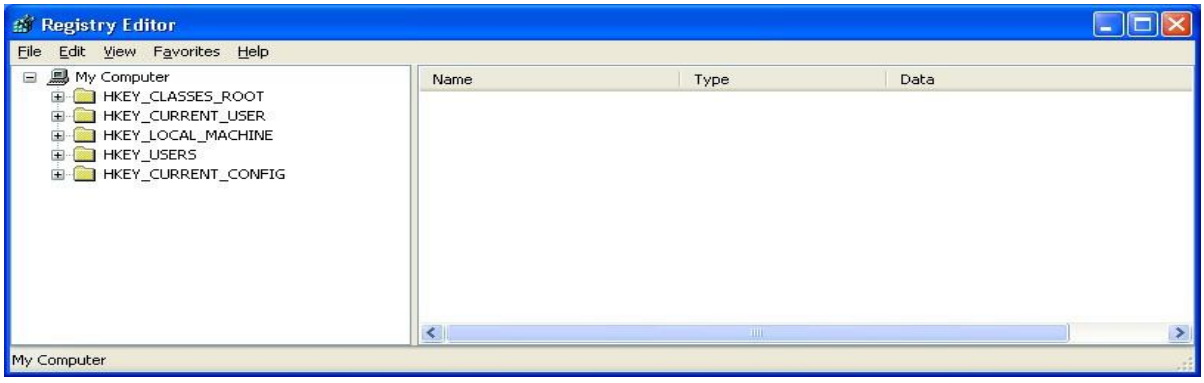


Figure (3-9) Windows Registry Editor

3.8.1 The Structure of Windows Registry

The Windows Registry Editor is divided into two panels (Figure 3-9). The left one is key panel and the right one is value panel. In the left panel, there are five root keys [77][83]:

1. *HKEY_CLASSES_ROOT*: The information that is stored here makes sure that the correct program opens when opening a file by using Windows Explorer. This key is sometimes abbreviated as "HKCR".

2. *HKEY_CURRENT_USER:* This key allows all the Windows programs and applications to create, access, modify, and store the information of current console user without determining which user is logging in. This key is sometimes abbreviated as "HKCU", under the root key HKCU. There are also five sub keys, namely Environment, Identities, Network, Software, and Volatile Environment.
    a) *Environment* is about the environmental configurations.
    b) *Identities* are related to Outlook Express.
    c) *Network* contains settings related to connect the mapped network drive.
    d) *Software* refers to the user application settings.
    e) *Volatile Environment* is used to define the environmental variables according to different users who log on a computer.

3. *HKEY_LOCAL_MACHINE*: It contains all the configuration settings of a computer. When a computer starts up, the local machine settings will boot before the individual user settings.

4. *HKEY_USERS*: It contains all the per-user settings such as current console user and other users who logged on this computer before.

5. HKEY_CURRENT_CONFIG: It contains information about the hardware profile that is used by the local computer at system startup.

HKCU will be used in this study.

3.9 EA Add-In Architecture

EA is a great UML CASE tool, but we can make it even better by adding and extending new functionality in the form of an add-in (discussed in paragraph 3.5.3 on page 48).

To fully understand the steps necessary to get the add-in running, we should first understand how EA's add-in architecture works see figure (3-10) [7].

When EA starts up, it will read the registry key [HKEY_CURRENT_USER \Software\Sparx Systems\EAAddins].

Each of the keys in this location represents an add-in for EA to load. The (default) value of the key contains the name of the assembly and the name of the add-in class separated by a dot. EA then asks Windows for the location of the assembly (An assembly is a file that is automatically generated by the .NET compiler upon successful compilation of every .NET application. It can be either a DLL or an executable file), which is stored on the COM codebase entries in the registry, and it will use the public operations defined in the add-in class.

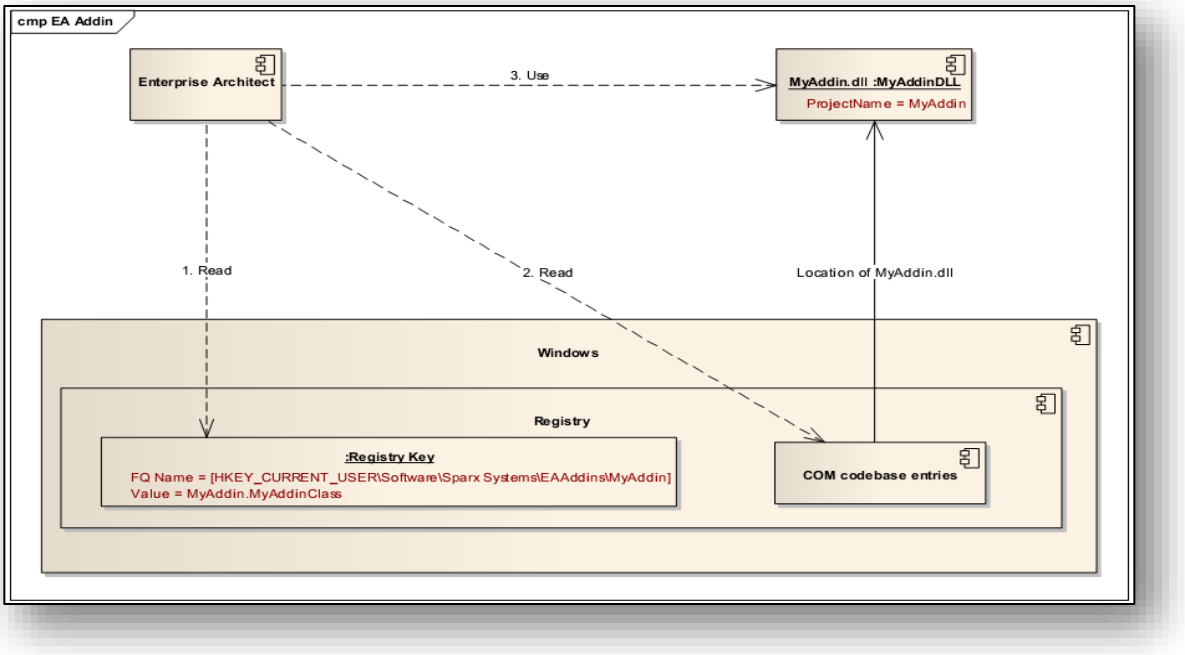


Figure (3-10) Enterprise Architect Add-In operation.

The add-in will work correctly by performing these steps [7]:

1. Creating the add-in dll containing the add-in class.
2. Adding a key to registry containing the name of the assembly and the name of the add-in class.
3. Registering the dll in the COM codebase entries in the registry.

3.10 CASE Tools Integration

Software integration is the process of linking together different computing systems and software applications functionally, to act as a coordinated whole.

There are four levels of CASE tool integration (see figure (3-11)). At the low end of the integration spectrum is the individual (point solution tool), which is used to assist in a particular software engineering activity (e.g., analysis modeling) but does not directly communicate with other tools and is not tied to a project database. The second level is when individual tools provide facilities for data exchange (most do), the integration level is improved slightly. Such tools produce output in a standard format that should be compatible with other tools that can read the format, like the case in this study. In some cases, the builders of complementary CASE tools work together to form a bridge between the tools (e.g., an analysis and design tool that is coupled with a code generator). Using this approach, cooperation between tools can produce end products that would be difficult to create using either tool separately. The Third level is the single-source integration which occurs when a single CASE tools vendor integrates a number of different tools and sells them as a package. Finally, at the high end of the integration spectrum is the integrated project support environment (IPSE) [46].

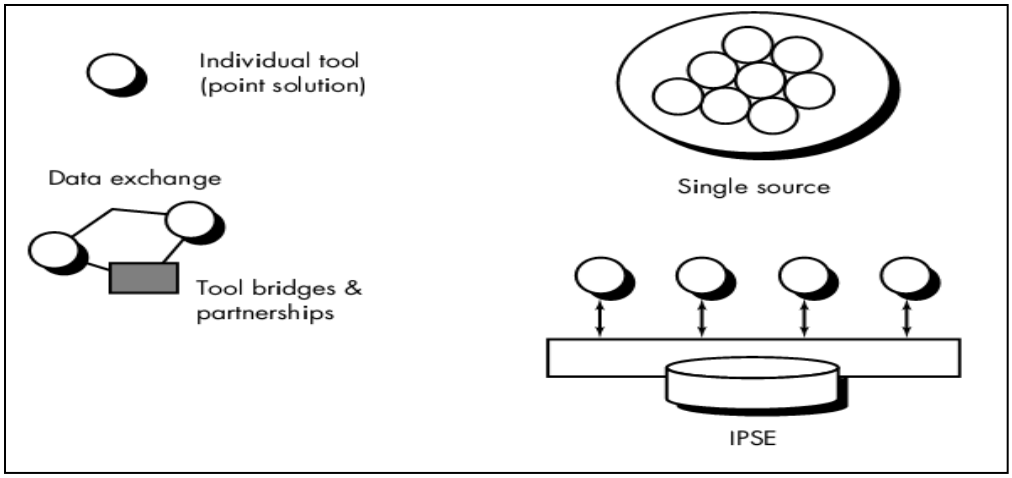


Figure (3-11) Levels of CASE Tool Integration [46].

3.11 Documentation in Software Engineering

For large software projects, it is usually the case that documentation starts being generated well before the development process begins, For some types of system, a comprehensive requirement document may be produced which defines the features required and expected behavior of the system. During the development process itself, all sorts of different documents may be produced – project plans, design specifications, test plans, etc [63].

So, generally documentation that will be produced falls into two categories [63]:

1. Process Documentation

   Effective management requires the process being managed to be visible. Because software is intangible and the software process involves apparently similar cognitive tasks rather than obviously different physical tasks, the only way this visibility can be achieved is through the use of process documentation. Process documentation falls into a number of categories:

   a) *Plans, estimates and schedules*: These are documents produced by managers and are used to predict and to control the software process.

b) *Reports*: These are documents which report how resources were used during the process of development.
c) *Standards*: These are documents which set out how the process is to be implemented. These may be developed from organizational, national or international standards.
d) *Working papers*: These are often the principal technical communication documents in a project. They record the ideas and thoughts of the engineers working on the project.
e) *Memos and electronic mail messages*: These record the details of everyday communications between managers and development engineers.

2. Product Documentation

Product documentation is concerned with describing the delivered software product. Unlike most process documentation, it has a relatively long life. It must evolve in step with the product which it describes. Product documentation includes user documentation which tells users how to use the software product and system documentation which is principally intended for maintenance engineers. Generally, this documentation comes as a user guide with the delivered product.

3.11.1 Reporting

It is obvious from the previous section how important is the documentation in the development of software systems either as a process or product documentation. Sometimes a project manager wants to know the progress of some process in the development. So, here comes the reporting services. A *report* can be defined as a self-explanatory statement of facts that is related to a specific subject and provides information for decision making and follow up actions.

There are very large numbers of reporting tools, such as:

1. Microsoft Report.
2. Telerik Report.

3. Active Report.
4. Crystal Report.

The crystal report will be used in this study due to its simplicity and power set of features integrated with visual studio. It is defined as a business intelligence application used to design and generate reports from a wide range of data sources (i.e. SQL server, MySQL, Oracle, Microsoft Access and XML) [43].

3.11.2 Help Files

Good documentation help users, but bad documentation can be downright insulting. It is therefore essential that the users can easily find what they are looking for. To get that, “help files” should be organized in a logical manner, easily searchable, extensively linked and indexed. Here comes **Microsoft Compiled HTML Help** (CHM) which is a Microsoft proprietary online help format that consists of a collection of HTML (Hyper Text Markup Language) pages.

There are a lot of tools for creating such help files such as Fly Help, HTML-kit, Help &Manual 5, etc.

Help files will be adopted in this study for the proposed tools.

Chapter Four

Analysis and Design of KDM, KRS and KDB Tools

Analysis and Design of KDM, KRS, and KDB Tools

4.1 Introduction

This chapter explains in detail the proposed tools from the analysis and design point of view.

These tools are named; Khalil Design Metrics (KDM) Tool, Khalil Reporting Service (KRS) Tool and Khalil Database (KDB) Tool, which help the software engineer in the design phase of the software life cycle.

For modeling the proposed tools, we use the following CASE tools:

1. Edraw Max.
2. Microsoft Visio.
3. EA.

4.2 The Proposed Tools from the User Point of View

Before start analyzing the proposed tools in detail, it is needed to describes them in a general way by showing how the final user of the proposed tools like a software engineer, project manager or programmer will use them.

This can be done by using the Use-Case diagram see figure (4-1).

It is obvious from the figure that KDM tool is used to calculate the metrics for the OOD and considered being the main tool, while KRS tool can help with the documentation of the results, and finally KDB tool can help by storing the metrics in the database.

4.3 Analysis of the Proposed Tools

First of all, the idea was to create a tool that helps the software engineer to make a decision especially in the design phase and to correct software design (class diagram) by measuring it using the OOD metrics. This was done by developing a Windows application to calculate the metrics. After that, the idea of CASE tool integration and developing an add-in for EA was born.

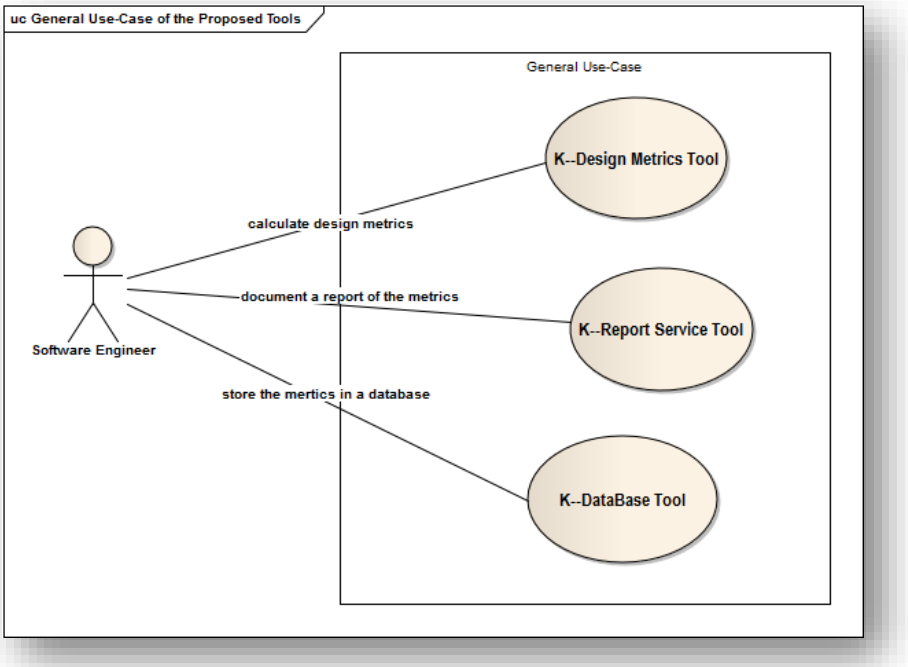


Figure (4-1) General Use-Case for the proposed tools

4.4 KDM Tool

EA does not support any tool that measures the class diagram. So, in this study KDM tool was developed to work from inside the EA as add-in so it can help software engineer to understand better the design of the software by scrutinizing the class diagram of that software by means of design metrics.

Hence, the KDM tool (add-in) can be deployed to work on other machines not just on the machine where it is developed, so that other software engineers can use it. The proposed KDM tool accepts XMI 1.1 for the UML 1.3 generated by the EA v7.5 as **input** and calculates metrics for that design.

The **output** of KDM tool is the value of metrics and 3-dimension pie chart which visualizes the value of each metric. It also gives statistics about that design and produces XML document. See figure (4-2) which shows the input and output of the KDM tool.

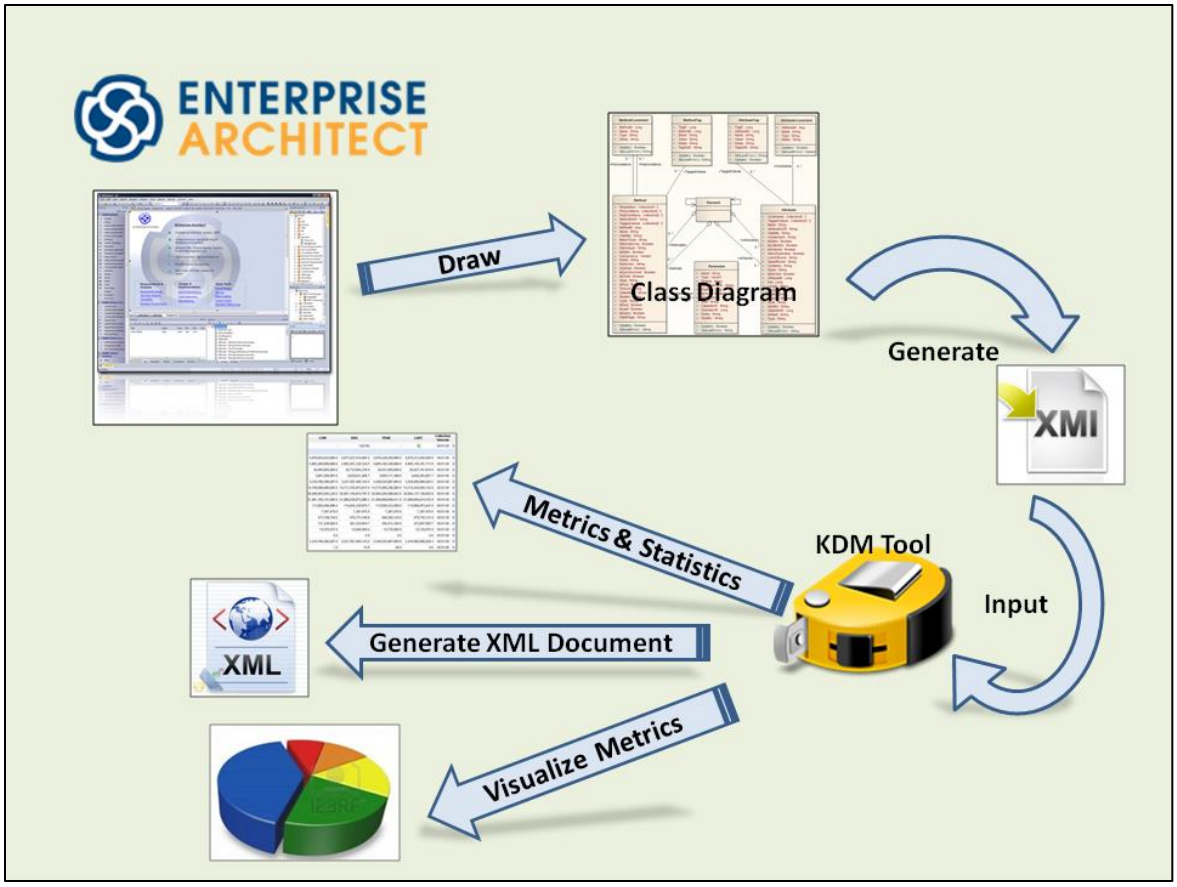


Figure (4-2) Input and output for KDM tool

4.4.1 KDM Tool in SDLC

KDM tool operates on UML class diagram either in the analysis phase (high-level design) or in the design phase (low-level or detailed design).

As in (paragraph 3.3.1) KDM tool is classified as Upper CASE Tool (front-end) since it works in the upper level of the SDLC (see figure (4-3)).

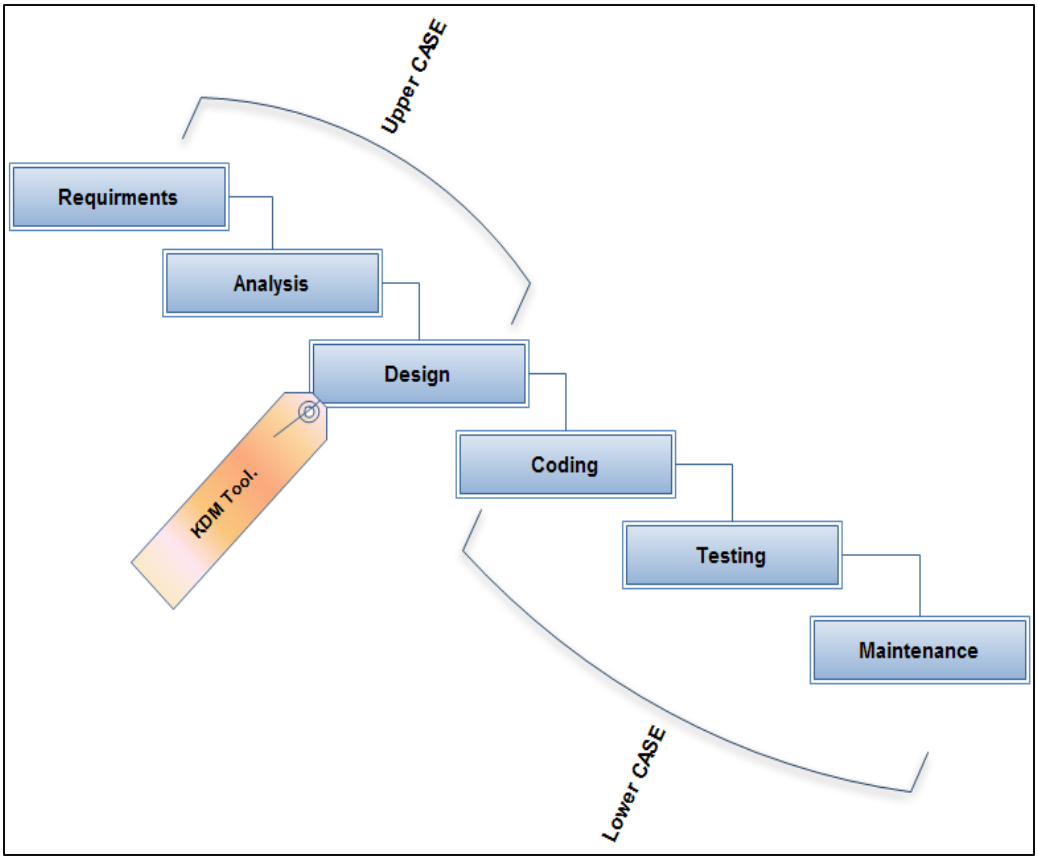


Figure (4-3) KDM tool classification according to SDLC.

4.4.2 How KDM Tool Works

KDM tool imports XMI document which then will be fed into the XMI parser. The parser will extract the required information from XMI document and pass it to the metric module which contains the MOOD model and MEMOOD model which in turn calculates the metrics for that design. KDM tool draws 3D pie chart, gives recommendations about design naming conventions, gives design statistics and exports XML, (see figure (4-4)).

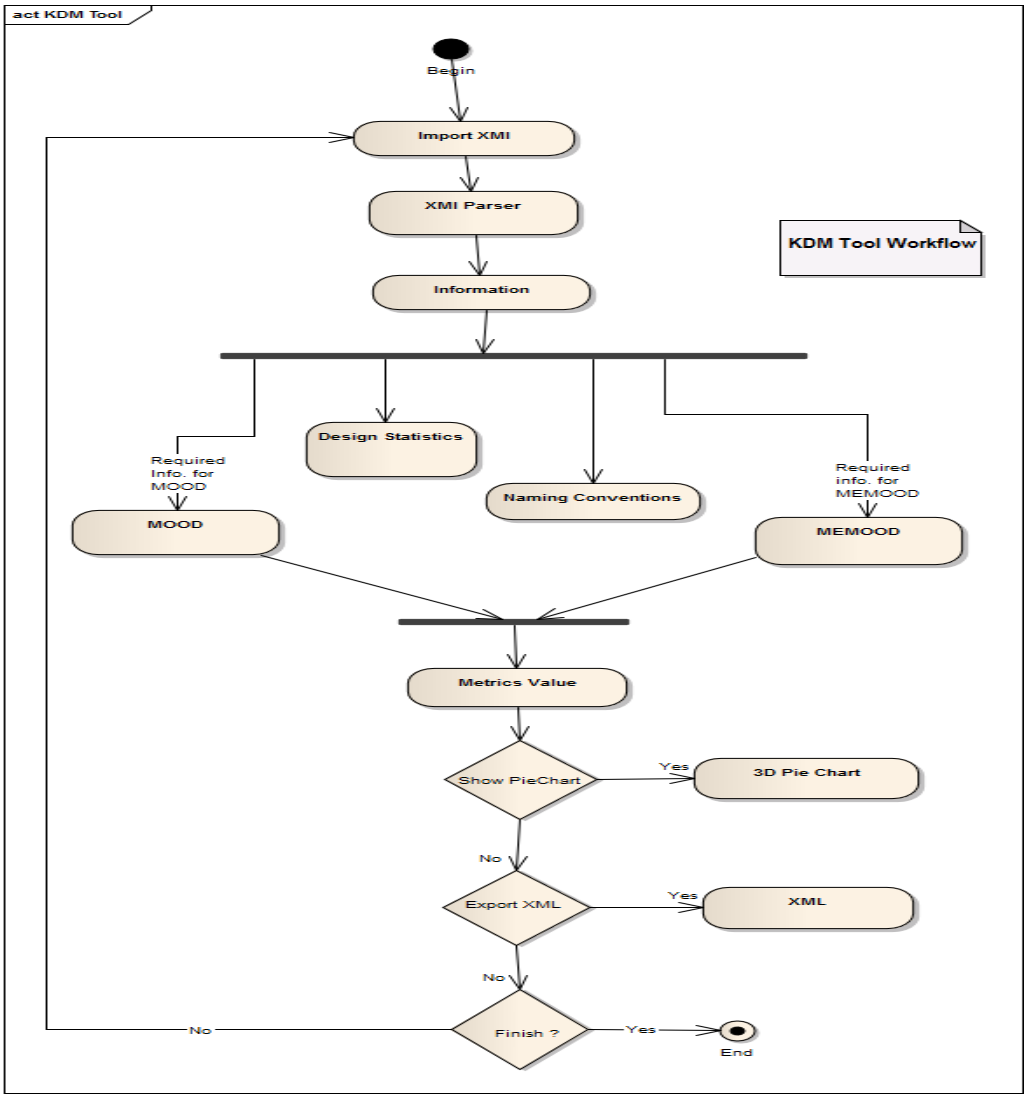


Figure (4-4) KDM tool work flow.

Class diagram for KDM tool is in Appendix D (see figures D-11 through D-15).

4.4.2.1 XMI Document

XMI is a way of saving UML diagrams as XML. XMI document contains huge data which describe the UML diagram (in this study the class diagram) in detail such as the name of each class, its attributes, operations, relationships, style, etc. XMI document is stored either as XML or XMI extension which means that the information is represented or structured as **tags**. Those tags are similar to

HTML tags (a tag begins with < then its name and ends with > such as <class> tag). See figure (4-5) which represents XMI document sample.

Figure (4-5) XMI document sample

XMI document has a large set of tags. Some are important but others are not, such as the style of each class, date of creation, etc.

The tags used to calculate the metrics in this study are listed in table (4-1) with their description.

Table (4-1) XMI tags used in the study

Tag	Description
<UML:Class>	This tag is used to represent the class element. **UML:** is a namespace (Namespace provides a means to distinguish one XML vocabulary from another, which enables to create richer documents by combining multiple vocabularies into one document type [28]) which stands for "**omg.org/UML1.3**"
<UML:Attribute>	This tag is used to represent the attribute of the class
<UML:Operation>	This tag is used to represent the methods of the class

<UML:TaggedValue>	Tagged Values are a way of adding additional information to an element
<UML:Generalization>	This tag is used to represent inheritance relationship and it has two tagged values: "**ea_sourceName**" which represents source class (sub class) that inherits form target class (super class). "**ea_targetName**" which represents target class (super class) in which subclass inherits from it.
<UML:Association>	This tag is used to represent association, aggregation and composition. We can tell the difference between them by their tagged values. It has two tagged values for the source and the target classes.

4.4.2.2 XMI Parser

XMI parser is used to extract data from XMI document, especially those tags listed in table (4-1).

Two important programming technologies were used in building XMI parser. They are:

LINQ (Language Integrated Query) -to-XML and Lambda expressions (for further information see Appendix A).

XMI parser will store all values of tags in lists like a list which contains the names for all classes; a list which may contain operations for each class; a list which may contain source classes and target classes for generalization relationship, ...etc.

In order to find the name for the classes, attributes or operations in the XMI document, the following algorithm can be used:

Algorithm:

Step 1: Read XMI document and load it into XDocument object

Step 2: Determine the tag = “Class”

Step 3: repeat for each tag

Step 3-1: extract the value of the name attribute of the tag

Step 3-2: save the name in the class list

Step 3-3: if not finish reading all tags, go to 3

Step 4: Display the class list

"Class list" will contain the name of each class in the XMI document; this list is the basis for all other methods in the XMI parser, because in order to find the name of each method in some class, it is needed to know the class name first.

Hence, to find attributes or methods names, the same algorithm can be used except for the tag which can be either as "Attribute" or "Operation".

In case of inheritance relationship, when it is needed to find the source classes (sub classes) and target classes in generalization relationship, the following algorithm can be used:

Algorithm:

Step 1: Read XMI document and load it into XDocument object

Step 2: Determine the relation = "Generalization"

Step 3: repeat for each tag

Step 3-1: extract the value of the ea_source tag attribute of the relation tag

Step 3-2: save the name in a source list

Step 3-3: extract the value of the ea_target tag attribute of the relation tag

Step 3-4: save the name in a target list

Step 3-5: if not finish reading all tags, go to 3

Step 4: Display the list

"Source list" and "target list" contain the subtype classes and supertype classes in XMI document. By knowing the source and target classes in the generalization relationship, this will help calculating metrics like MIF or AIF which are related to inheritance concept.

Thus, to find the source and the target list of another relation, only tag relation will change.

Now consider the following figure (4-6) which represents a simple class diagram for aircraft types.

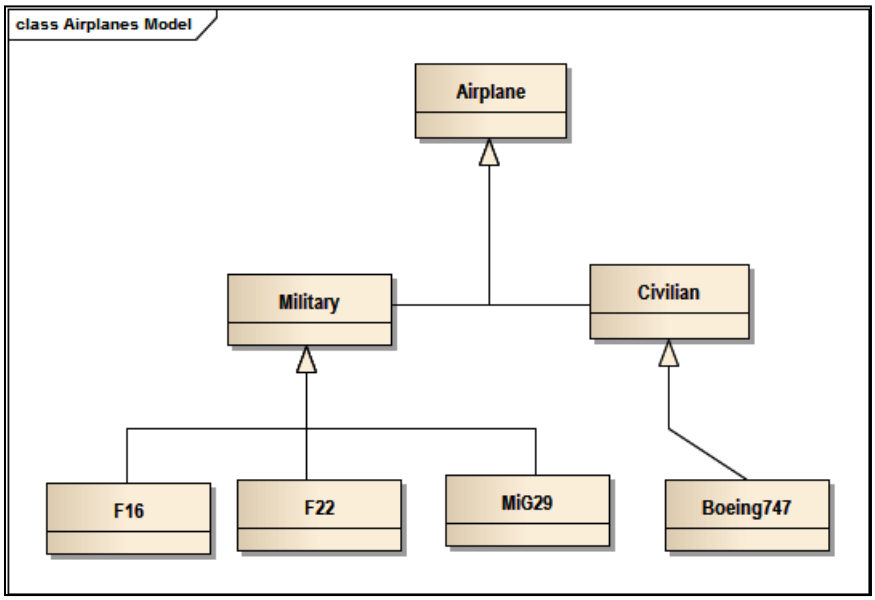


Figure (4-6) Simple class diagram for aircraft classification.

According to the above algorithms, “class list” will contain the names for all classes, see table (4-2)

Table (4-2) Sample of the class list

Classes Name
Airplane
Military
Civilian
Boeing747
MiG29
F16
F22

“source list” and “target list” for generalization relationship can be seen in table (4-3).

Table (4-3) Sample of source and target lists

Source Classes	Target Classes
Military	**Airplane**
Civilian	**Airplane**
Boeing747	**Civilian**
F16	**Military**
F22	**Military**
MiG29	**Military**

After collecting all required information, it is time to calculate the metrics for that design.

4.4.3 Suggested Algorithms for MOOD Model

In chapter two (paragraph 2.8 page 23) there is a description of MOOD model in detail that will help to explain how to calculate each equation of MOOD model.

4.4.3.1 Algorithm for MHF Metric

MHF metric is used to measure encapsulation for the class diagram, actually for the invisibilities of methods for that class. The flowchart is in figure (4-7). Sequence diagram can be seen in Appendix D figure (D-1).

Algorithm:

Step 1: import and verify XMI document

Step 2: Parse XMI document

Step 3: Define a list for each access modifier of the methods

Step 4: Repeat for each class

Step 4-1: Store methods for each class; where the public methods are stored in the public list, private methods in private list and protected methods (if existed) in the protected list

Step 4-2: Go to step4

Step 5: Calculate MHF equation (see equation 2.1)

Step 6: Display MHF for the design

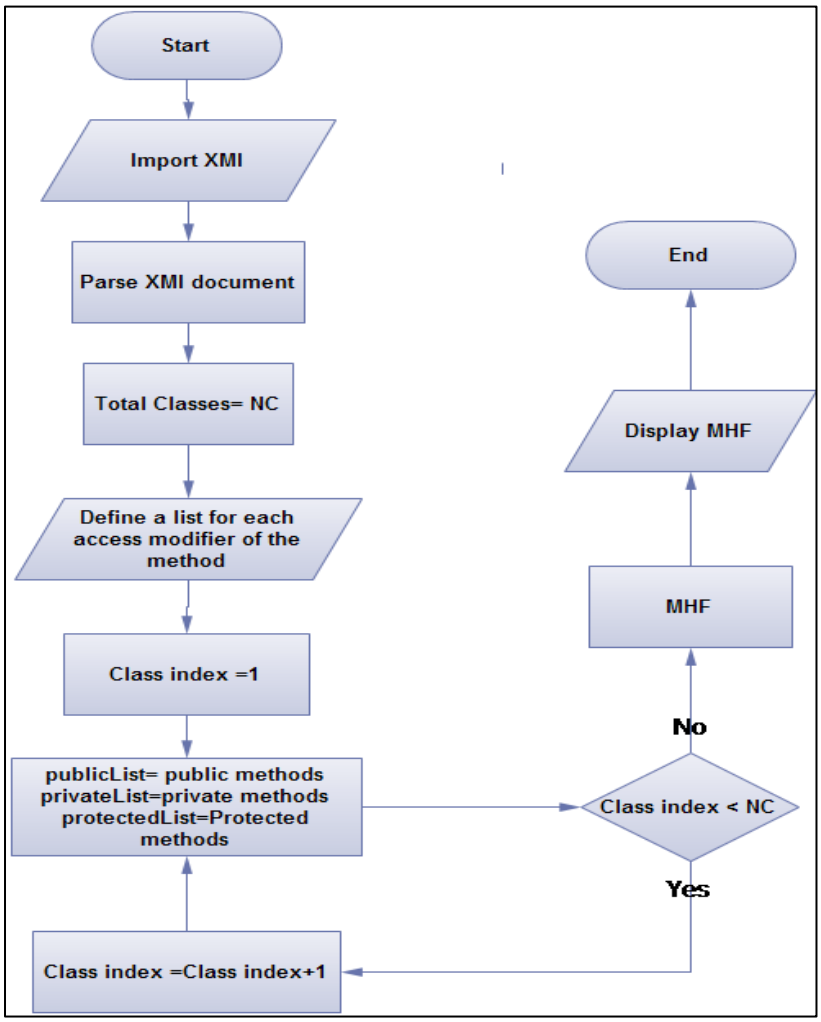


Figure (4-7) Flowchart for MHF algorithm

4.4.3.2 Algorithm for AHF Metric

AHF metric is used to measure encapsulation for the class diagram, actually for the invisibilities of the attributes for that class. Sequence diagram can be seen in Appendix D figure (D-2).

Algorithm:

Step 1: Import and verify XMI document

Step 2: Parse XMI document

Step 3: Define a list for each access modifier of the attributes

Step 4: Repeat for each class

Step 4-1: Store attributes for each class; where public attributes are stored in the public list, private attributes in private list and protected attributes (if existed) in the protected list

Step 4-2: Go to step4

Step 5: Calculate AHF equation (see equation 2.2)

Step 6: Display AHF for the design

See figure (4-8) which represents the flowchart for AHF Algorithm.

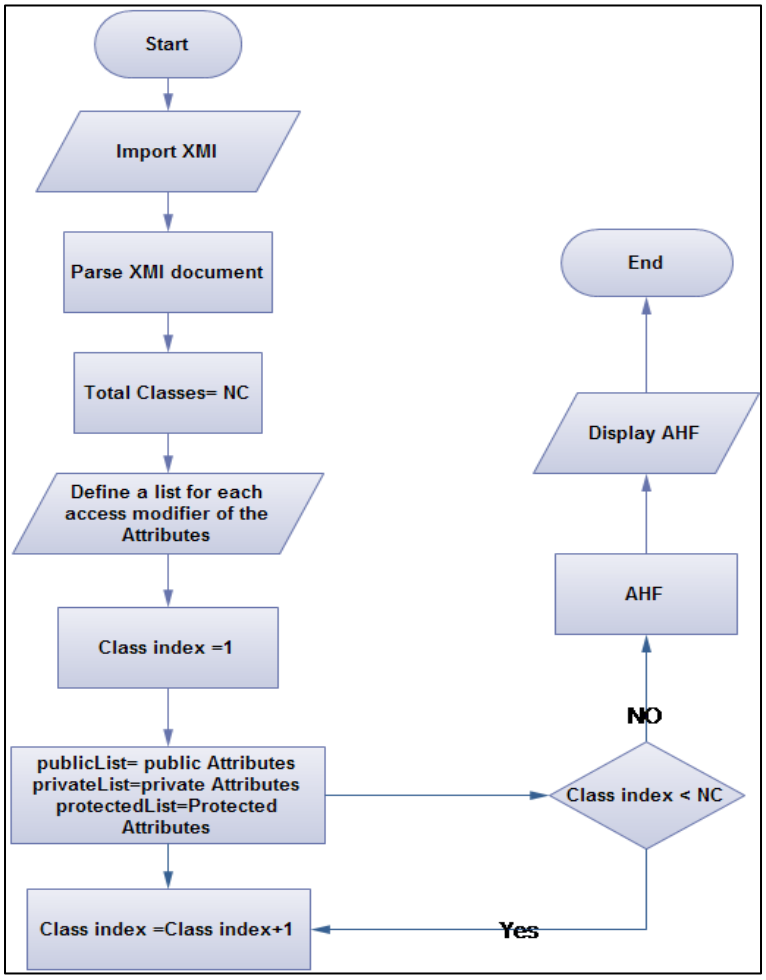


Figure (4-8) Flowchart for AHF Algorithm

4.4.3.3 A Suggested Algorithm for Finding the Root of Generalization or Aggregation Relationship

Sometimes a number of either a generalization hierarchy or aggregation hierarchy exists. This means that there are a number of roots in the design. In order to find the root of either of them, the following algorithm is suggested.

Algorithm:

Step 1: Import and verify XMI document

Step 2: Parse XMI document

Step 3: Determine the type of the relationship

Step 4: Define lists for root classes, subclasses, and super classes.

Step 5: Repeat for each class in the list of super classes

Step 5-1: If any class is not in the list of source (subclasses), it means that the class does not inherit from other classes, so it is a root, add it to root list

Step 5-2: Go to step 5

Step 6: If some class is repeated more than once, then delete it.

Step 7: Display root list

See Figure (4-9) which represents the flowchart for finding the root.

Sequence diagram for root algorithm can be seen in Appendix D figure (D-3).

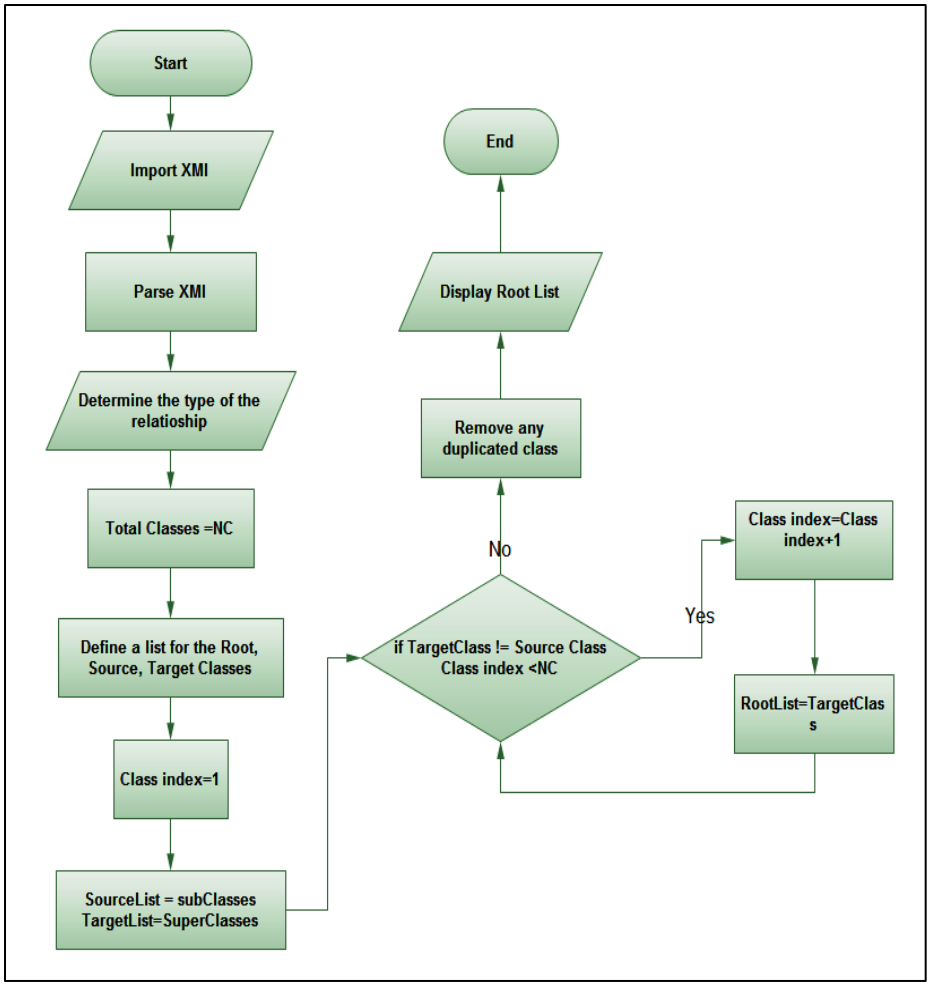


Figure (4-9) Flowchart for Root algorithm

4.4.3.4 Algorithm for MIF Metric

MIF metric is used to measure the inheritance of the class diagram, which is the ratio of the inherited methods in it. Sequence diagram can be seen in Appendix D figure (D-4).

Algorithm:

Step 1: Import and verify XMI document

Step 2: Parse XMI document

Step 3: Define a list of source classes (subclasses) and another list of the target classes (super classes).

Step 4: Find the root of the generalization relationship

Step 5: Repeat for each class in source and target lists

Step 5-1: Store inherited methods in a list called inherited list

Step 5-2: Go to 5

Step 6: Calculate the equation of MIF (see equation 2.3)

Step 7: Display MIF for the design

See figure (4-10) which shows the flowchart for MIF algorithm.

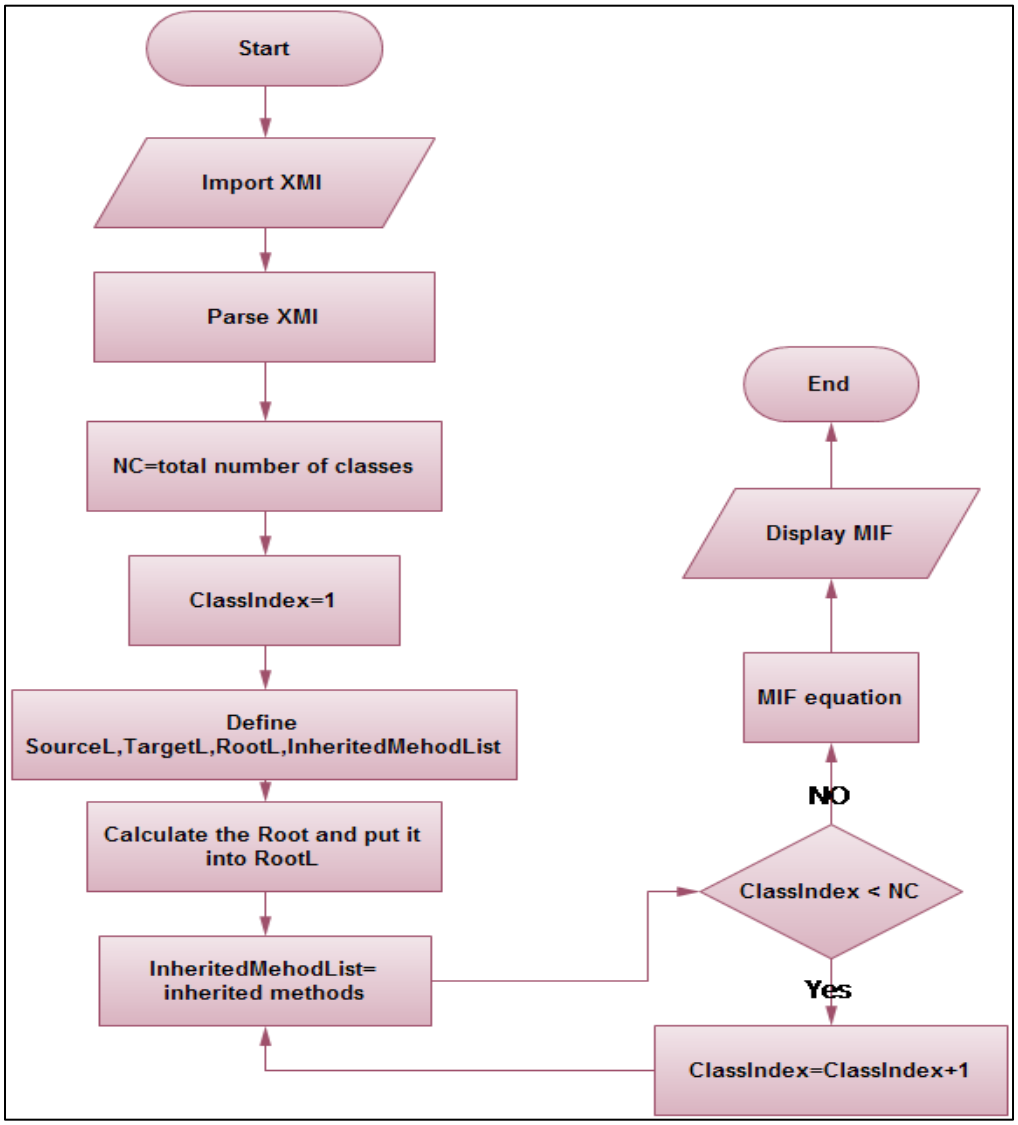


Figure (4-10) Flowchart for MIF Algorithm

4.4.3.5 Algorithm for AIF Metric

AIF metric is used to measure the inheritance of the class diagram, which is the ratio of the inherited attributes in it.

Algorithm:

Step 1: Import and verify XMI document

Step 2: Parse XMI document

Step 3: Define a list for source classes (subclasses) and another list for the target classes (super classes).

Step 4: Find the root of the generalization relationship

Step 5: Repeat for each class in source and target lists

Step 5-1: Store inherited attributes in a list called inherited list

Step 5-2: Go to 5

Step 8: Calculate the equation of AIF (see equation 2.4)

Step 9: Display AIF for the design

See figure (4-11) which shows the flowchart for AIF algorithm

Sequence diagram for AIF can be seen in Appendix D figure (D-5).

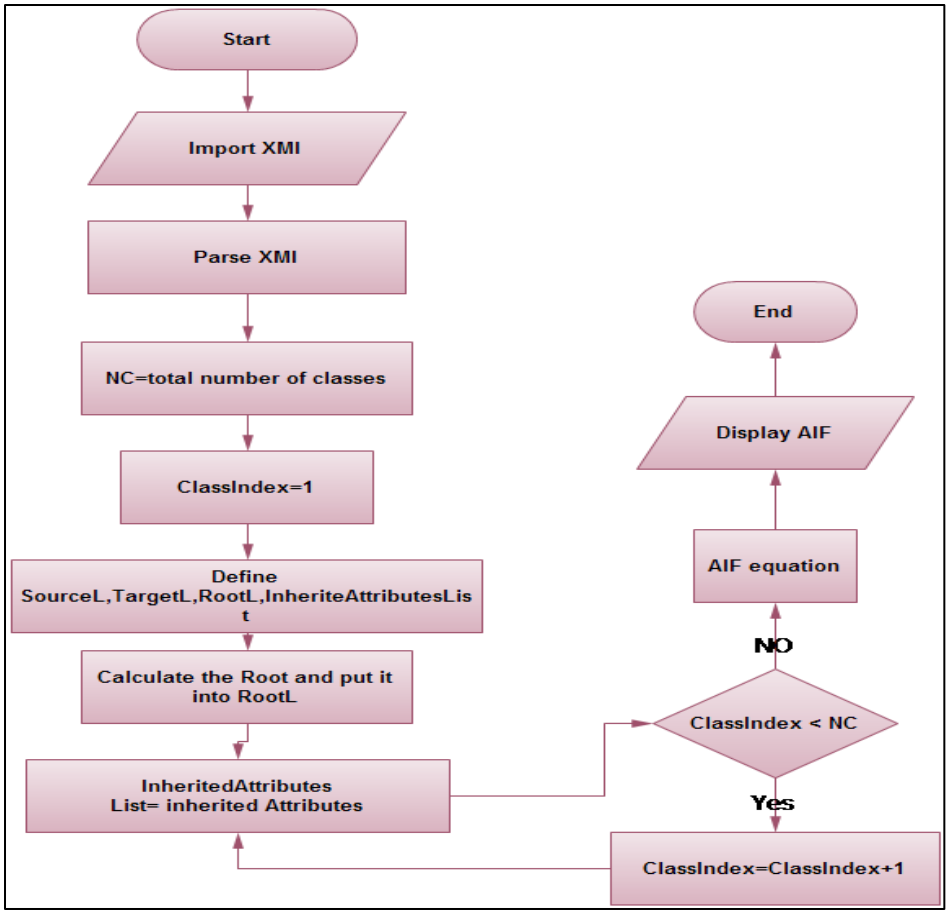


Figure (4-11) Flowchart for AIF Algorithm

4.4.3.6 Algorithm for POF

POF measures the polymorphism of the class diagrams. This metric calculates the ratio of the polymorphic methods (degree of overriding in class diagram).

Algorithm:

Step 1: Import and verify XMI

Step 2: Parse XMI document

Step 3: Calculate the source and target classes

Step 4: NC = total number of classes

Step 5: Repeat for each class while < NC

Step 5-1: Find the descendant classes for each class in the target list

Step 5-2: Find the new (declared) method for each class and put them in a list

Step 5-3: Find the overridden methods and put them in a list

Step 5-4: Go to 5

Step 6: Calculate POF (see equation 2.5)

Step 7: Display POF for the design

See figure (4-12) which shows the flowchart of POF algorithm.

Sequence diagram for POF algorithm can be seen in Appendix D figure (D-6).

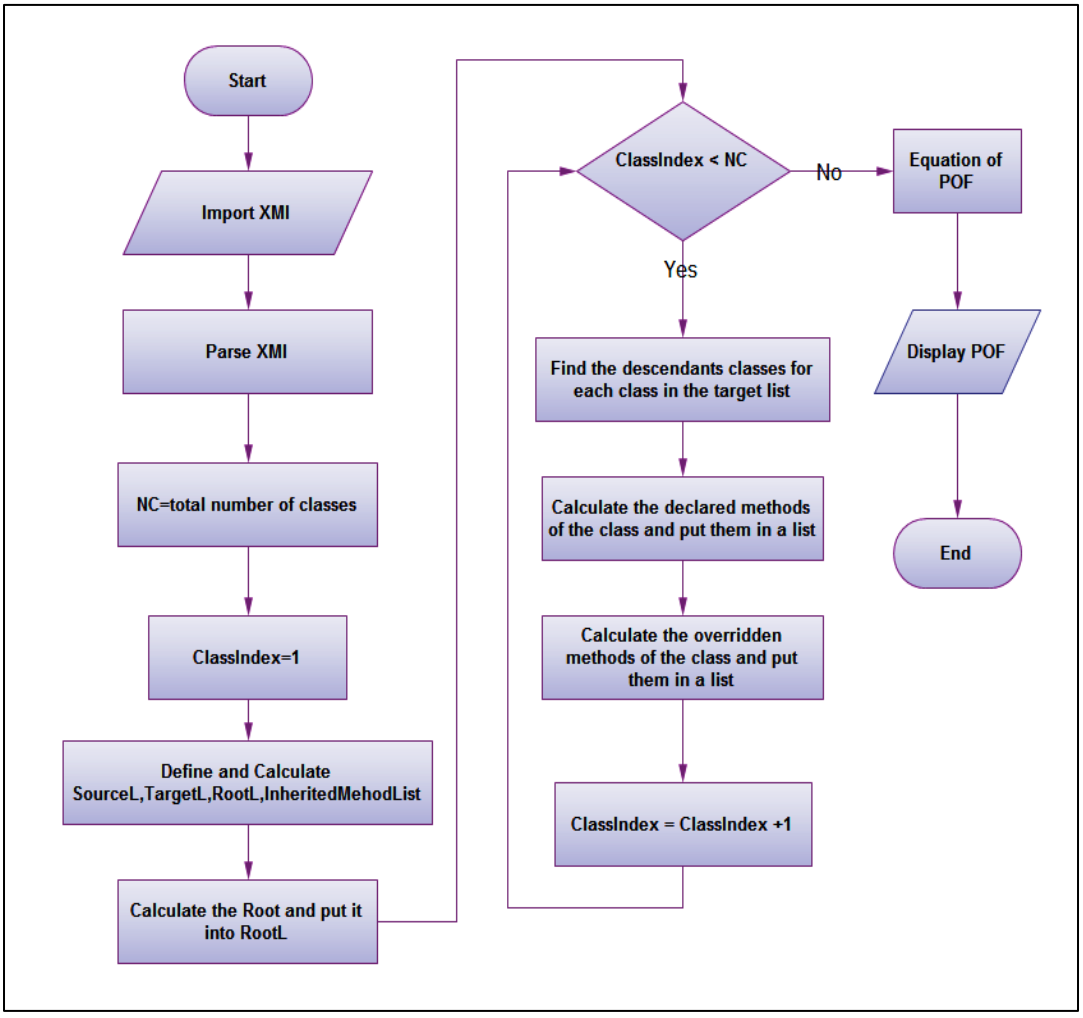


Figure (4-12) Flowchart for POF Algorithm

4.4.3.7 Algorithm for CF

CF is used to measure the coupling of the class diagram when one class calls a method of another class, then they are coupled.

Algorithm:

Step 1: Import and verify XMI

Step 2: Parse XMI document

Step 3: Find the source and target classes of the association relationship

Step 4: Concatenate the target list with the source list, remove duplication and put it into a new list called c list

Step 5: Repeat for each class in c list

Step 5-1: If a class has any relationship but not generalization then put it into a list

Step 5-2: Go to 5

Step 6: Apply CF equation (see equation 2.6)

Step 7: Display CF

See figure (4-13) which shows the flowchart of CF algorithm.

Sequence diagram for CF algorithm can be seen in Appendix D figure (D-7).

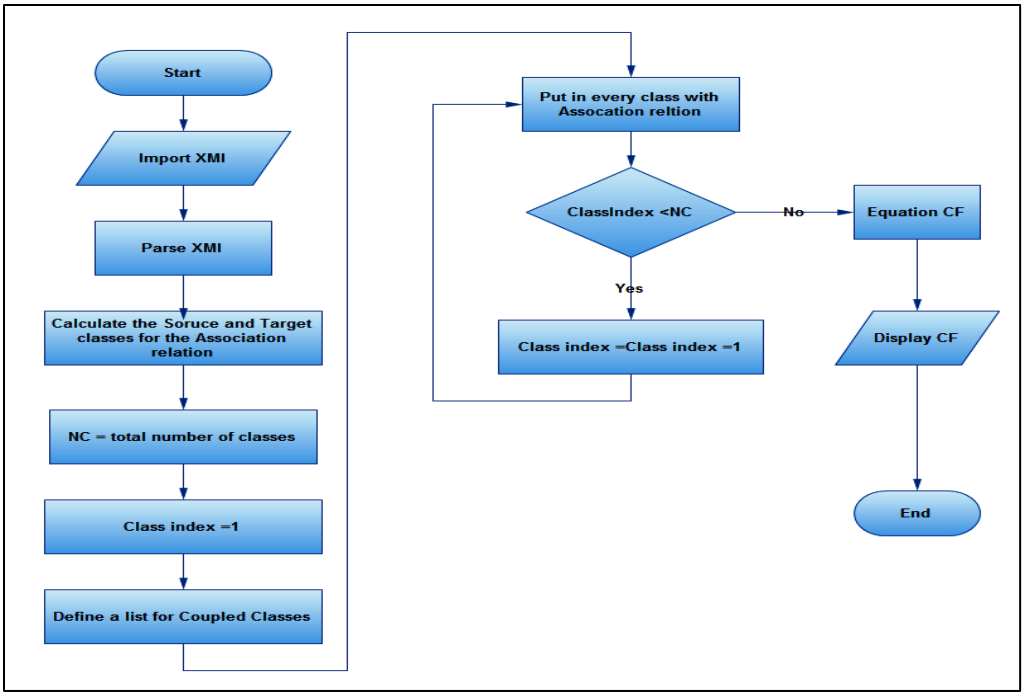


Figure (4-13) Flowchart for CF Algorithm

Now, after all algorithms about MOOD model are explained, consider the following example which illustrates all algorithms above, see figure (4-14)

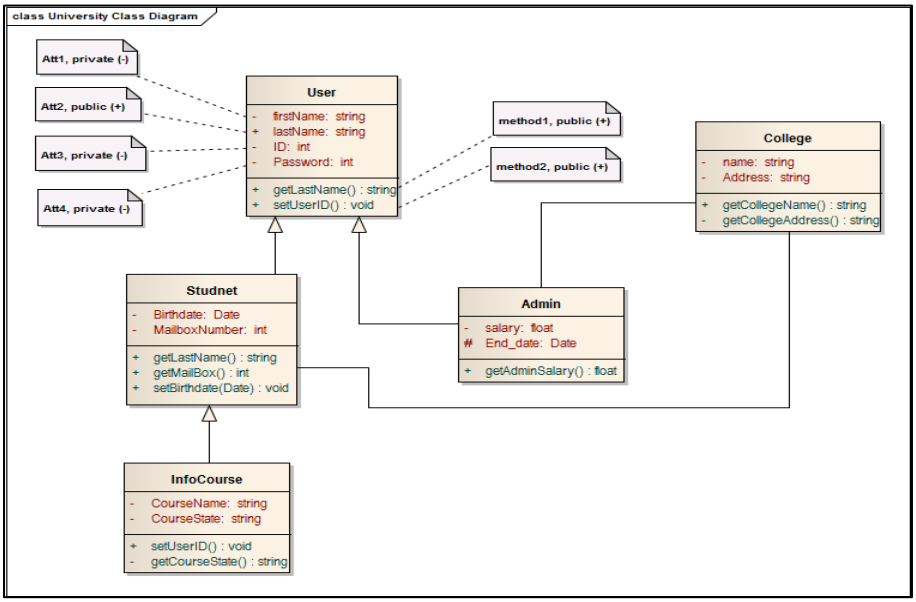


Figure (4-14) Simple Class Diagram for a University

Now consider the table (4-4) which represents the class diagram as numbers.

Table (4-4) Class Diagram Analysis

Class	Att.	Method	+ Att.	-Att.	#Att.	+M.	-M.	#M.
User	4	2	1	3	0	2	0	0
Student	2	3	0	2	0	3	0	0
Admin	2	1	0	1	1	1	0	0
InfoCourse	2	2	0	2	0	1	1	0
College	2	2	0	2	0	1	1	0

Where:

- Att. is an abbreviation for attribute.
- M. is an abbreviation for method.
- + prefix means public modifier.
- - prefix means private modifier.
- # prefix means protected modifier.

Figure (4-14) shows a simple class diagram with 5 classes, 12 attributes and 10 methods, to calculate the metrics according to the table above using metrics equations.

1. **Encapsulation (equations 2.1 and 2.2)**

$$MHF = \frac{0+1+0+0+1}{2+2+1+3+2} = \frac{2}{10} = 20\ \%$$

$$AHF = \frac{3+2+1+2+2}{4+2+2+2+2} = \frac{10}{12} = 83.33\ \%$$

2. **Inheritance (equations 2.3 and 2.4)**

$$MIF = \frac{0+2+2+5}{2+5+3+7} = \frac{9}{17} = 52.94\%$$

$$AIF = \frac{0+4+4+6}{4+6+6+8} = \frac{14}{24} = 58.33\%$$

3. **Polymorphism (equation 2.5)**

$$POF = \frac{0+1+0+1}{4+2+0+0} = \frac{2}{6} = 33.33\%$$

4. **Coupling (equation 2.6)**

$$CF = \frac{3}{25-5} = \frac{3}{20} = 15\%$$

It is concluded from table (2- 1) that AHF, MIF, AIF are within the limit while MHF, POF, CF are not within the standard limit. So, a correction or a review of the design is needed.

4.4.4 Suggested Algorithms for MEMOOD model

In chapter two (paragraph 2.9 page 30), MEMOOD model was described in detail, and now the way to calculate each equation of MEMOOD model is explained.

4.4.4.1 Algorithm for Understandability

Understandability means how much the software engineer understands the design that he is working on or how easy to comprehend it.

In order to calculate the understandability of the design it is needed first to find two metrics, named NC and NGenH (see table 2-2). These two metrics along with some constant numbers are used to calculate the understandability of the design.

Flowchart for the understandability algorithm is depicted in figure (4-15).

Algorithm:

Step 1: Import XMI

Step 2: Parse XMI document

Step 3: Calculate NC and NgenH for the design

Step 4: Apply understandability equation (see equation 2.7)

Step 5: Save the value of understandability

Step 6: Display understandability

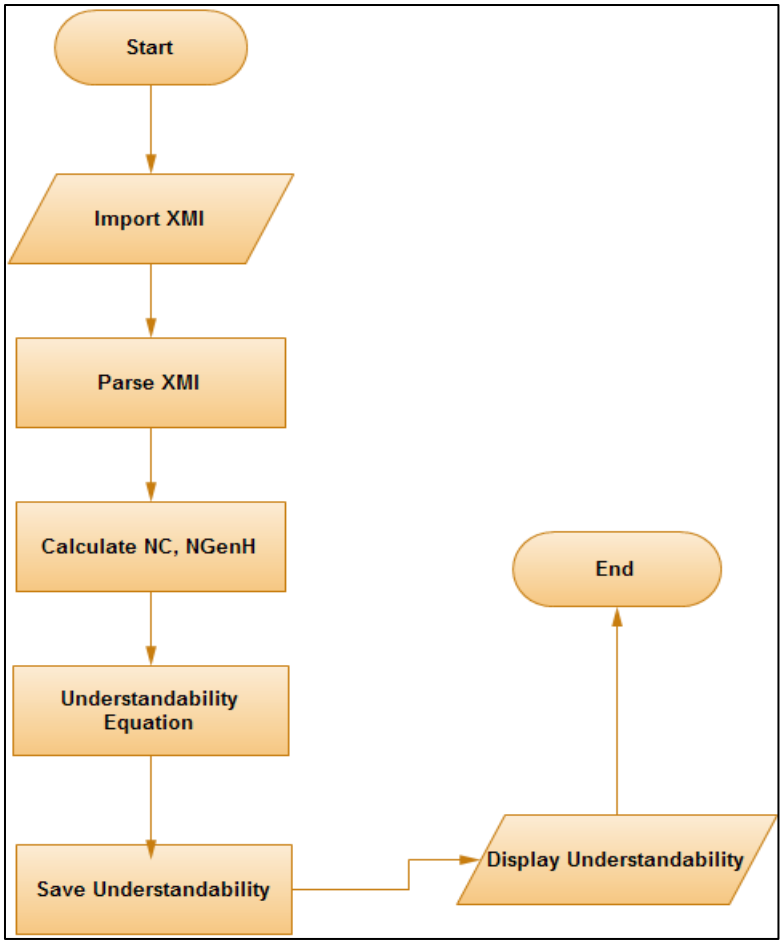


Figure (4-15) Flowchart for Understandability Algorithm

Sequence diagram for Understandability Algorithm can be seen in Appendix D figure (D-8).

4.4.4.2 Algorithm for Modifiability

Modifiability means the ability of software engineer to modify the design without affecting it.

In order to calculate the modifiability of the design it is needed first to find five metrics, named NC, Ngen, NgenH, NaggH and MaxDIT (see table 2-2). These five metrics along with some constant numbers are used to calculate the modifiability of the design.

Algorithm:

Step 1: Import XMI

Step 2: Parse XMI document

Step 3: Calculate NC, Ngen, NgenH, NaggH and MaxDIT for the design

Step 4: Apply modifiability equation (see equation 2.8)

Step 5: Save the value of modifiability

Step 6: Display modifiability

See figure (4-16) which represents the flowchart for modifiability quality attribute.

Sequence diagram for Modifiability Algorithm can be seen in Appendix D figure (D-9).

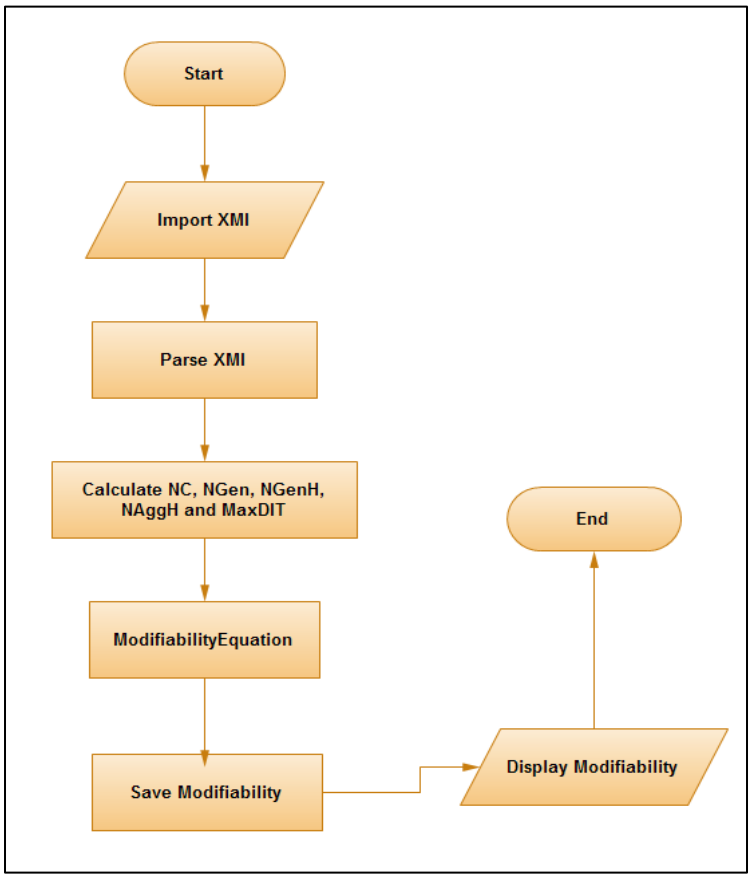


Figure (4-16) Flowchart for modifiability algorithm

4.4.4.3 Algorithm for Maintainability

Maintainability means how easy it is for software engineer to maintain the design by means of adapting, correcting or improving the design.

In order to calculate the maintainability, understandability and modifiability are used along with a number of constants to form the equation (see equation 2.9).

Algorithm:

Step 1: Import XMI

Step 2: Parse XMI document

Step 3: Calculate understandability and modifiability

Step 4: Apply maintainability model (see equation 2.9)

Step 5: Display maintainability

See figure (4-17) which represents the flowchart for maintainability quality attribute.

Sequence diagram for Maintainability Algorithm can be seen in Appendix D figure (D-10).

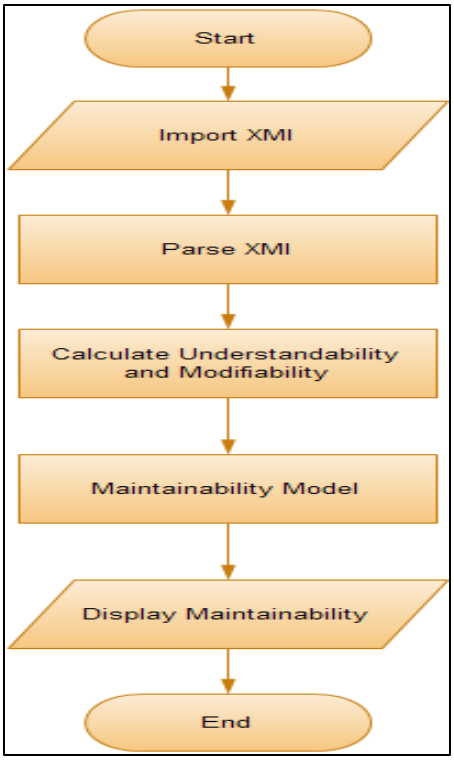


Figure (4-17) Flowchart for Maintainability Model

Now go back to figure (4-14), the following table (4-5) can be deduced.

Table (4-5) Metrics used to calculate MEMOOD Model, also see table (2-2)

NC	Ngen	NgenH	NaggH	MaxDit
5	3	1	0	2

Where

- NC is the total number of classes = 5.
- Ngen is the number of generalization relationships = 3.
- NgenH is the number of generalization hierarchy =1, since there is only one generalization tree.
- MaxDit is the maximum number of depth of inheritance tree =2, since the User class is in level 0 of the generalization hierarchy, Student and Admin classes are in level 1, and InfoCourse class is in level 2.

- NaggH is the number of the aggregation hierarchy = 0, since there is no aggregation hierarchy.

1. **Understandability (equation 2.7)**

   Understandability = 1.166 + 0.256*5 – 0.394*1 = 2.05

2. **Modifiability (equation 2.8)**

   Modifiability = 0.629 + 0.471*5 – 0.173*3 – 0.616*0 –0.696*1+ 0.396*2 = 2.56

3. **Maintainability (equation 2.9)**

   Maintainability = -0.126 + 0.645* 2.05 + 0.502*2.56 = 2.48

4.4.5 XML

XML is a standard technology that is concerned with the description and structuring of data by means of **tags** that are similar to HTML ones.

XML can be used almost in every application especially in the web. See figure (4-18) which represents a sample of XML.

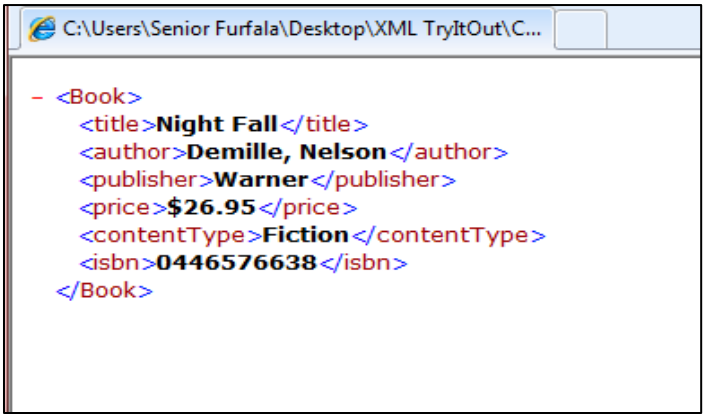


Figure (4-18) XML sample

It can be seen from the figure above that XML is used to describe a book; its title, author, price, etc.

XML sometimes is used as intermediate data that flow between applications and these applications pass these XML between each other, so XML can be used as a bridge between various applications.

Going back to figure (4-2 page 62), it can be seen that XML was used as **output** from KDM tool and as it was said before that KDM tool is the main tool and KRS, KDB are used to support it. So, how can these tools communicate between each other? The answer is by using XML as a bridge between them. XML parser was built for that XML in which it will be understood and used properly.

A specific structure of XML is proposed in this study (see table (4-6)).

Table (4-6) XML structure of the KDM XML output

Tag	Description
<Metrics>	This tag is used as root for a number of metrics
<Metric Id="" />	This tag is used as an identifier for the metrics
<DesignerName>	This tag is used to describe the designer name
<ModelName>	This tag is used to describe the model name
<MHF>	This tag is used to describe the MHF metric
<AHF>	This tag is used to describe the AHF metric
<MIF>	This tag is used to describe the MIF metric
<AIF>	This tag is used to describe the AIF metric
<CF>	This tag is used to describe the CF metric
<POF>	This tag is used to describe the POF metric
<Understandability>	This tag is used to describe the understandability
<Modifiability>	This tag is used to describe the modifiability
<Maintainability>	This tag is used to describe the maintainability

A sample of XML document can be seen in figure (4-19) which is also the XML output from KDM tool.

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
- <Metrics>
    <Metric Id="10" />
    <DesignerName>Khalil Ahmed</DesignerName>
    <ModelName>My Model</ModelName>
    <MHF>7.14</MHF>
    <AHF>84.21</AHF>
    <MIF>63.04</MIF>
    <AIF>55.56</AIF>
    <CF>11.11</CF>
    <POF>9.52</POF>
    <Understadability>3.08</Understadability>
    <Modifiability>3.83</Modifiability>
    <Maintainability>3.78</Maintainability>
  </Metrics>
```

Figure (4-19) XML output from KDM tool

4.4.6 KDM Tool Sequence Diagram

KDM tool sequence of operations starts after importing XMI document and ends with exporting XML, see figure (4-20).

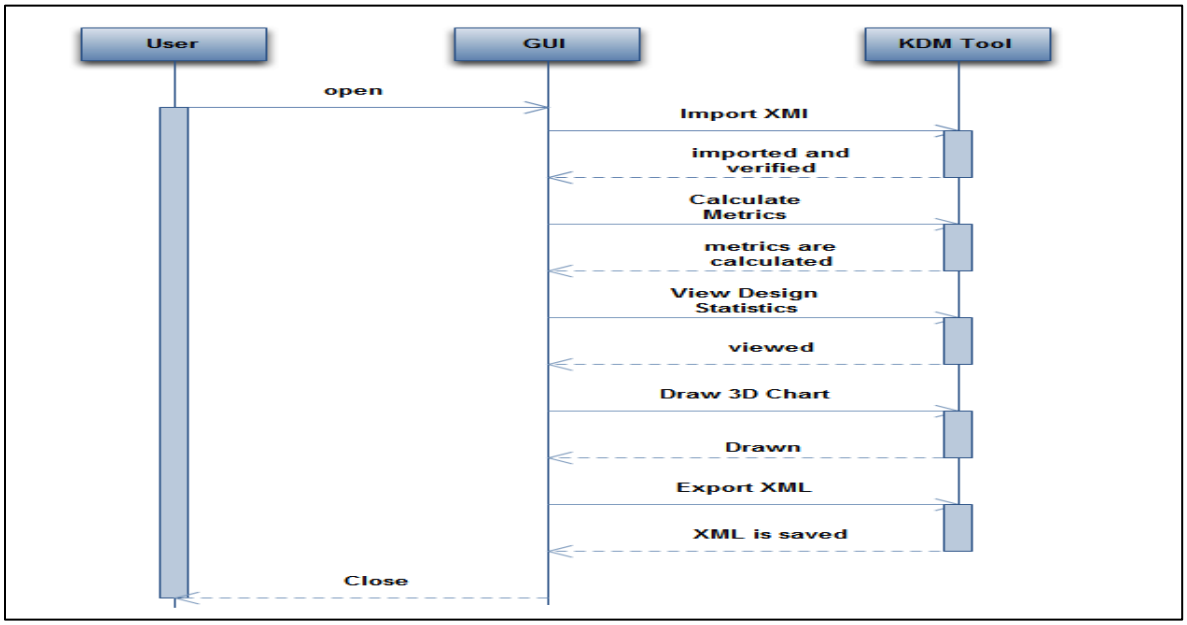


Figure (4-20) KDM tool sequence diagram

4.5 KRS Tool

KRS tool is a reporting tool that is integrated with EA. The purpose of this tool is to document metrics as a report for a project manager or maybe for the team leader.

It is said earlier that KRS tool supports KDM tool and they communicate by exchanging XML. The **input** for KRS tool is the XML output of KDM tool. So, KRS has a parser for the XML generated by KDM tool. The **output** of KRS tool is a crystal report which contains metrics and two graphs, see figure (4-21).

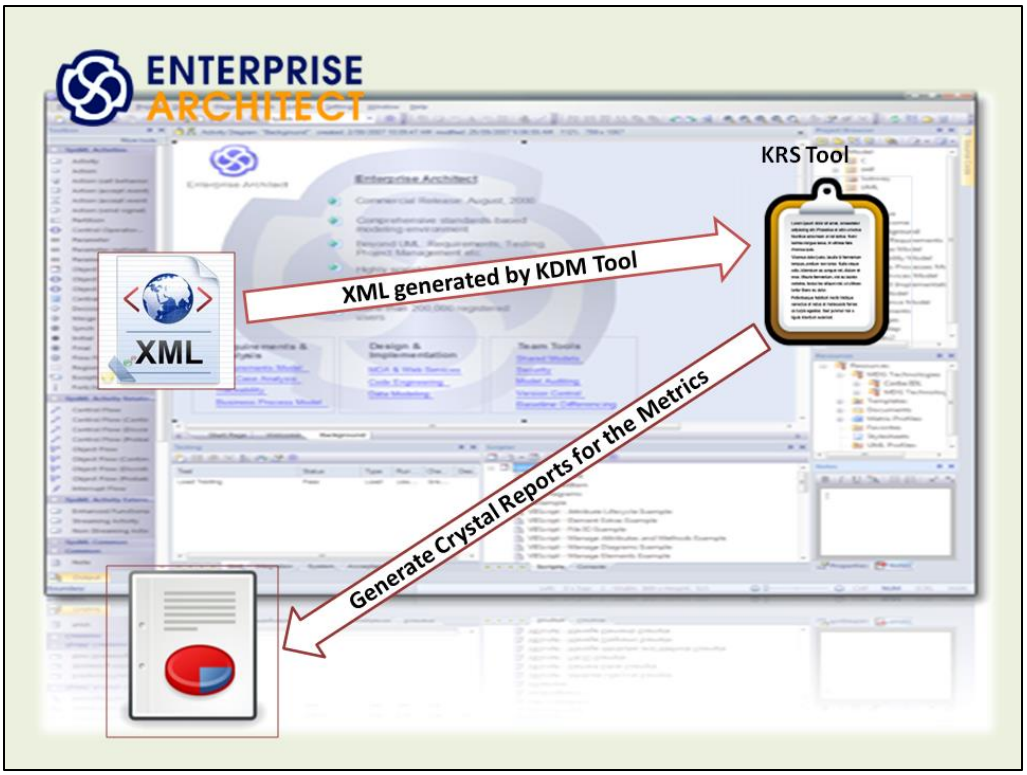


Figure (4-21) Input and output for KRS Tool

4.5.1 How KRS Tool Works

Before discussing how it works, it is needed to know where KRS tool works, and in which phase it supports in SDLC, see figure (4-3). KDM tool is an Upper CASE tool and since KRS works with the documentation of metrics in the same phase, it is deduced that KRS tool is also an Upper CASE tool.

KRS tool accepts XML document that is generated by KDM as input, see figure (4-19). Then XML document proceeds to XML parser which extracts the information and prepares it to be fed into the report generator and produces a crystal report of the design metrics. See figure (4-22).

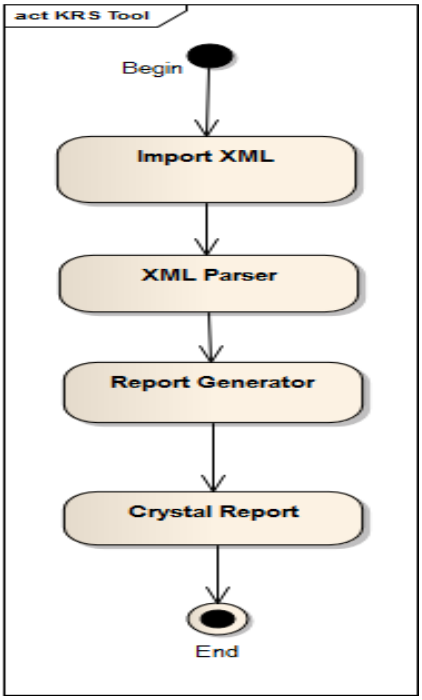


Figure (4-22) KRS Tool Workflow

4.5.2 XML Parser

XML parser extracts the value of each tag listed in table (4-6) from the XML document which is depicted in figure (4-23)

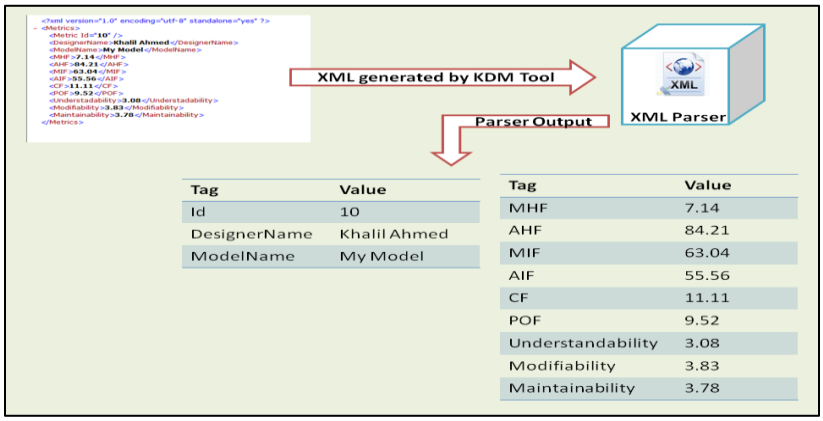


Figure (4-23) XML Parser

Algorithm:

Step 1: Import and verify XML

Step 2: Extract model name and designer name from XML document and put them in a list.

Step 3: Extract all metrics and put them in a list

Step 4: End

See figure (4-24) which shows the flowchart for the XML parser.

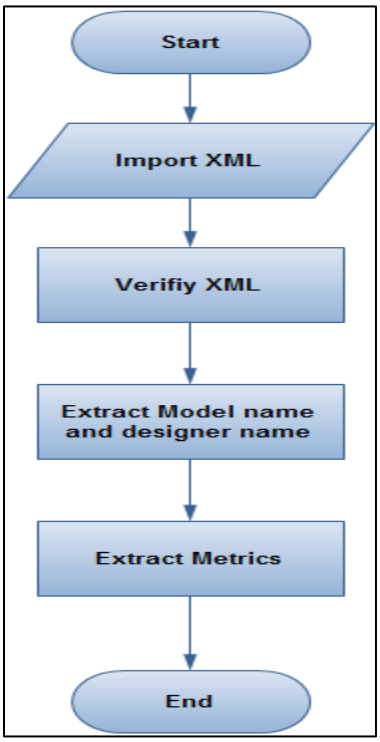


Figure (4-24) XML parser flowchart

4.5.3 KRS Tool Sequence Diagram

KRS tool sequence of operations starts after importing XML document that is generated by KDM and ends with showing a crystal report of that metrics which is depicted in figure (4-25).

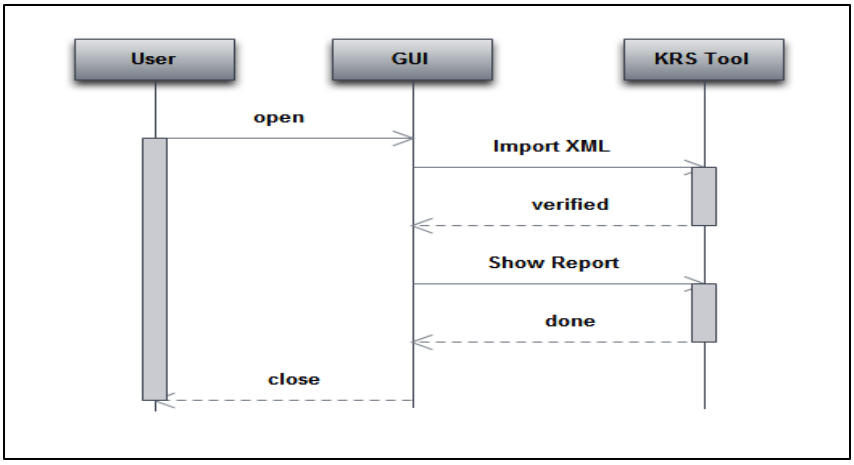


Figure (4-25) KRS Tool Sequence Diagram

4.6 KDB Tool

KDB tool is a database tool that is integrated with EA; KDB tool is used to store metrics in a database; may be for checking the metrics against another system (design) which has similar requirements or used as a historical data.

KDB tool has the same XML parser of KRS tool. KDB tool accepts the same XML which is generated by KDM tool as **input**, and stores the numeric (double data type) value of metrics in the database. See figure (4-26).

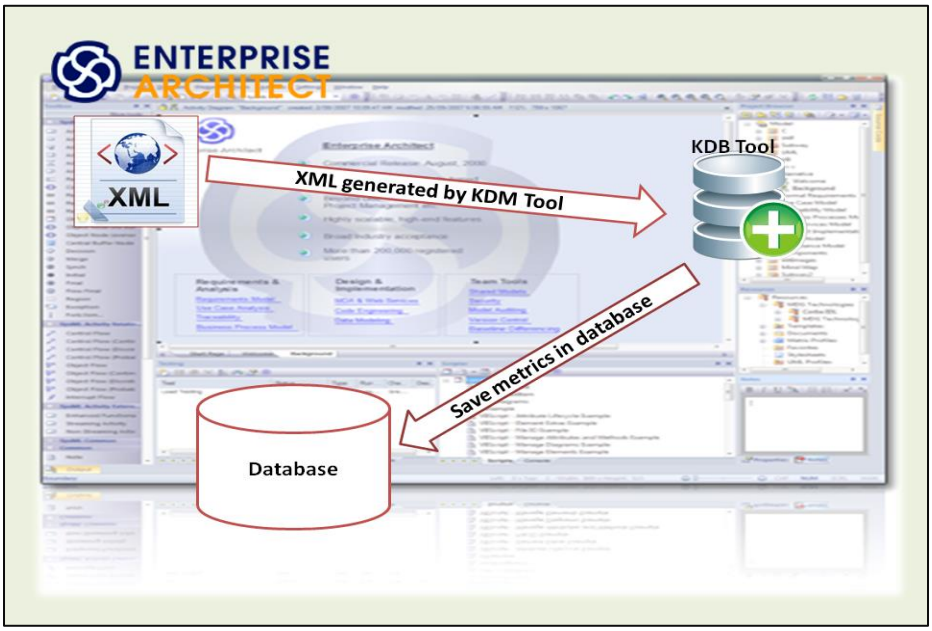


Figure (4-26) KDB Tool

4.6.1 How KDB Tool Works

Any tool that supports any phase in SDLC is considered a CASE tool, otherwise it is not. Since KDB tool stores metrics which are software engineering information and supports KDM tool, so, as a result, it is a CASE tool and can be considered an Upper CASE tool.

KDB tool takes XML document which is generated by KDM tool, parses it, and formats metrics in a way that can be stored in the database, see figure (4-27)

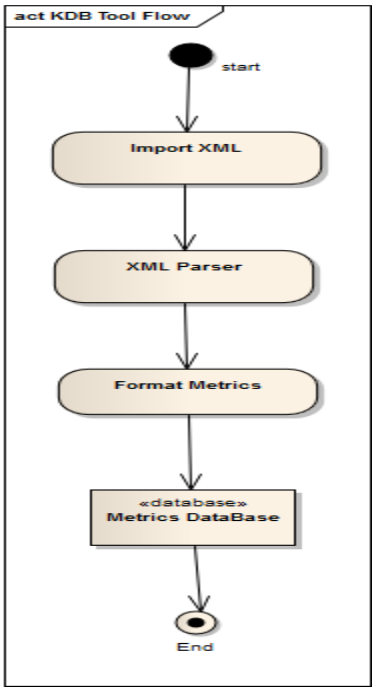


Figure (4-27) KDB Tool Wok flow

4.6.2 KDB Tool Entity Relationship Diagram

KDB tool has two entities; one for the information extracted from the XML document (i.e. model name, designer name, id), and another for metrics. See figure (4-28).

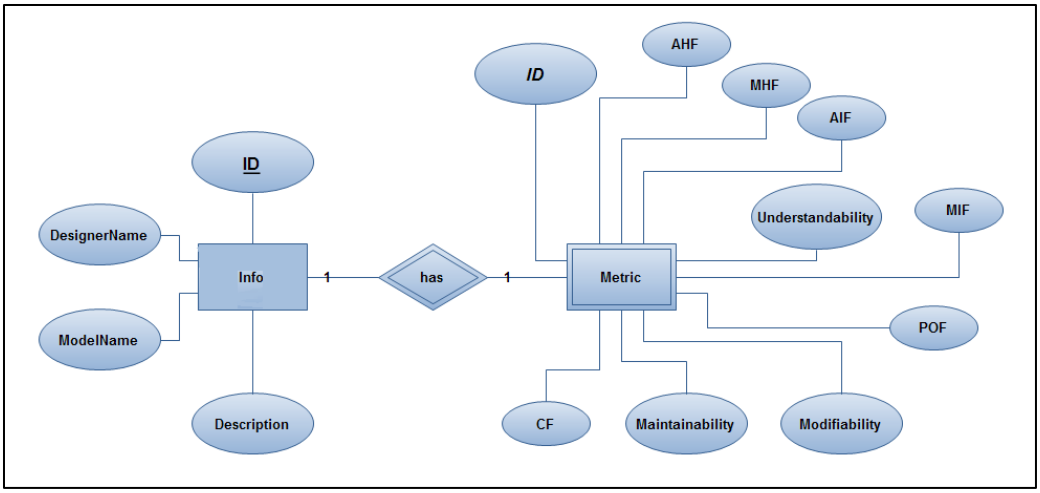


Figure (4-28) ER-Diagram for the KDB Tool

4.6.3 KDB Tool Sequence Diagram

KDB tool sequence of operations can be shown in figure (4-29).

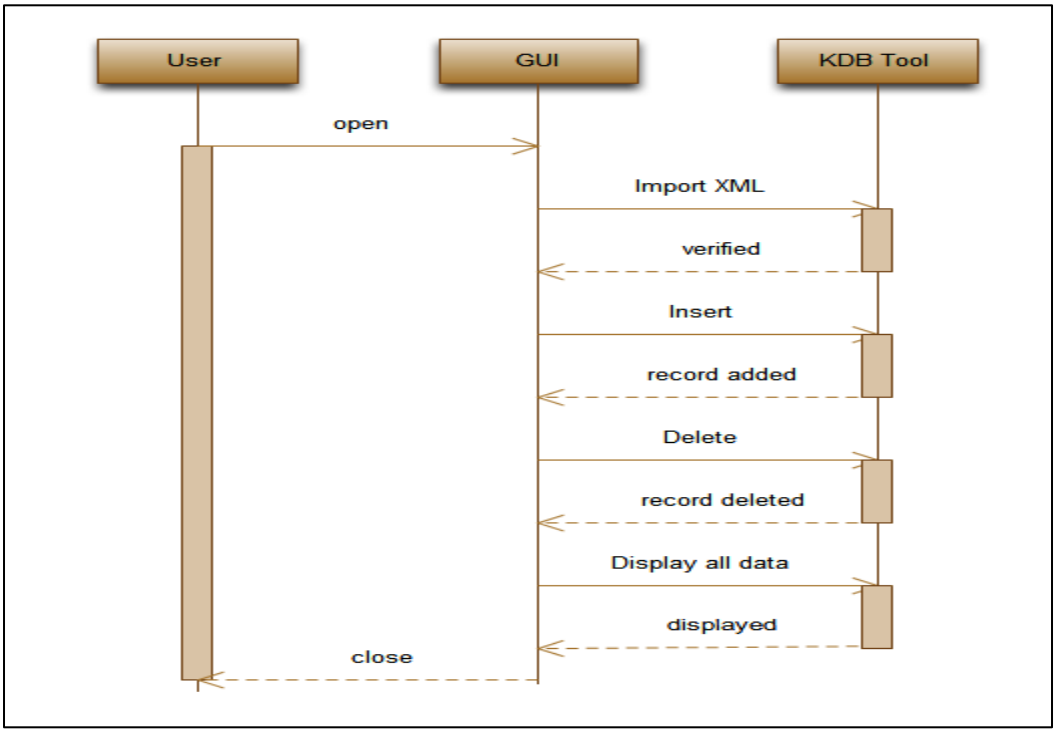


Figure (4-29) KDB Tool Sequence Diagram

Chapter Five

Implementation and Testing of KDM, KRS and KDB Tools

Implementation and Testing of KDM, KRS and KDB Tools

5.1 Introduction

This chapter explains the proposed tools from the implementation point of view and how they are used (tested). Also, the Graphical User Interface (GUI) followed by a case study and code metrics are explained.

5.2 Implementation of KDM, KRS, KDB Tools

All tools were implemented using C# programming language under Microsoft Framework version 4 with the aid of Microsoft Visual Studio 2010 (Ultimate) as IDE. Large number of libraries were used, some of them are listed in table (5-1).

Table (5-1) Libraries used

Library	Description
Interop.EA	Used for connecting with EA
System.XML System.XML.LINQ	Used for XMI and XML parsing
DevComponents.DotNetBar2	Used for creating GUI
System.Speech	Used for speech options
System.Data.SqlClient	Used for the database operations
CrystalDecisions	Used for creating crystal reports
Windows.DataVisualization	Used in building 3D pie charts

5.3 Requirements of KDM, KRS, KDB Tools

In order to work correctly, these programs must be installed on the computer:

- EA v7.5
- SQL Server 2008
- Microsoft Framework v4
- Windows Installer v3.1 and v4.5

5.4 GUI of KDM Tool

In order to open KDM tool, EA should be started first, then from the menu strip option (Add-Ins)→ Khalil-Package→ select KDM tool. See figure (5-1).

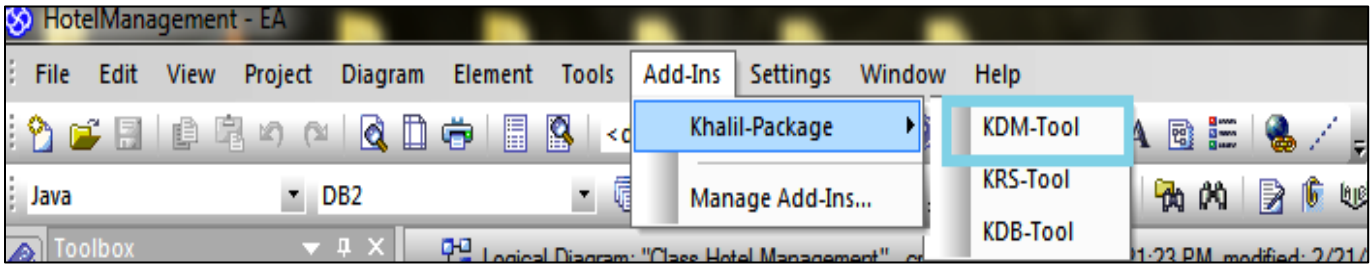


Figure (5-1) Add-Ins Menu

After clicking on KDM-Tool option from Khalil-Package menu strip, KDM tool will open. See figure (5-2).

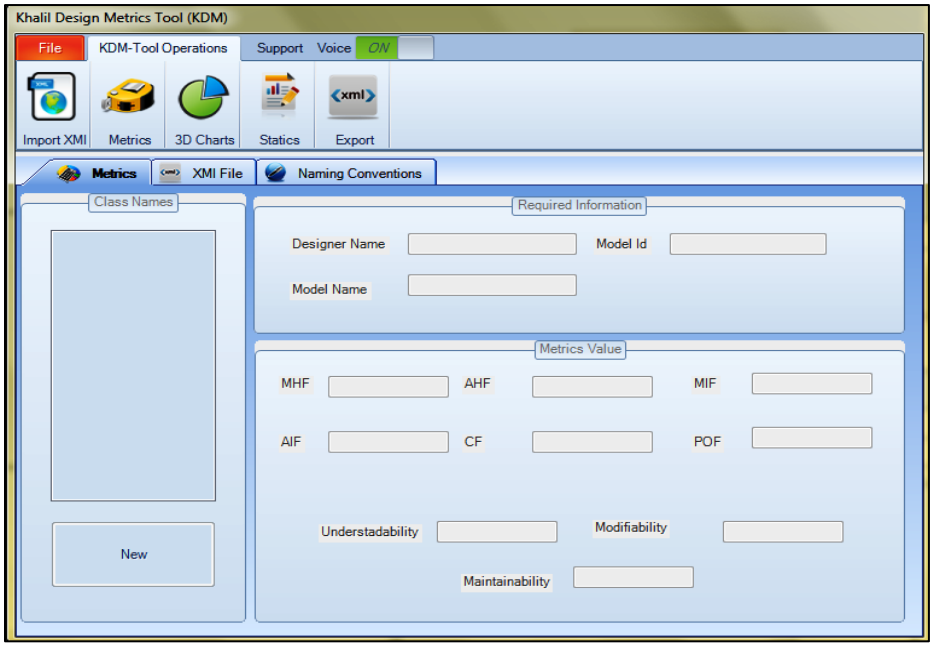
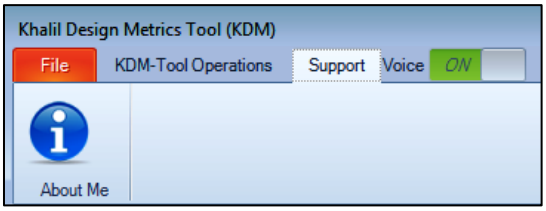


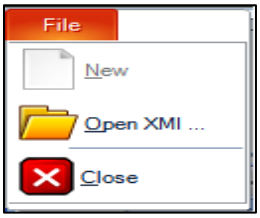
Figure (5-2) Main Window for KDM Tool

KDM tool has two main components; Ribbon bar and Tab Panel. Ribbon bar has two tabs (KDM-Tool Operations and Support) and file menu strip. See figure (5-3 a, b, c). For the description of each button, see table (5-2).

Figure (5-3) KDM-Tool Operation Tab in Ribbon Bar (a)

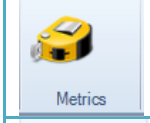
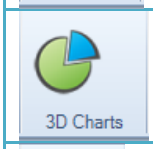
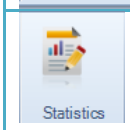


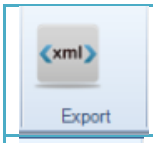
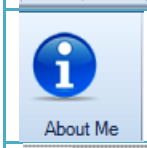
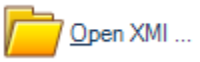
(b) Support Tab in Ribbon Bar



(c) File menu strip

Table (5-2) Ribbon Bar buttons description

Button	Description
	This button is used to import XMI document which is exported from EA.
	This button is used to calculate MOOD and MEMOOD metrics.
	This button is used to show 3D-Pie chart for MOOD and MEMOOD metrics. See figure (5-4).
	This button is used to show design statistics like number or methods, attributes, interfaces in design. Also some metrics that were mentioned in table (2-2). By clicking on this button a new form will be opened. See figure (5-5).

	This button is used to export MOOD and MEMOOD metrics as XML document in order to communicate with KRS and KDB tools.
	This button is used to show information about the KDM tool like product name, version, copyrights and description. See figure (5-6).
	This button is used so that a new XMI document can be imported.
	This button is used to import XMI document. So it acts the same as the "Import XMI file" button.
	This button is used to exit KDM tool.
	This switch button is used to turn the sound off.
	This switch button is used to turn the sound on.

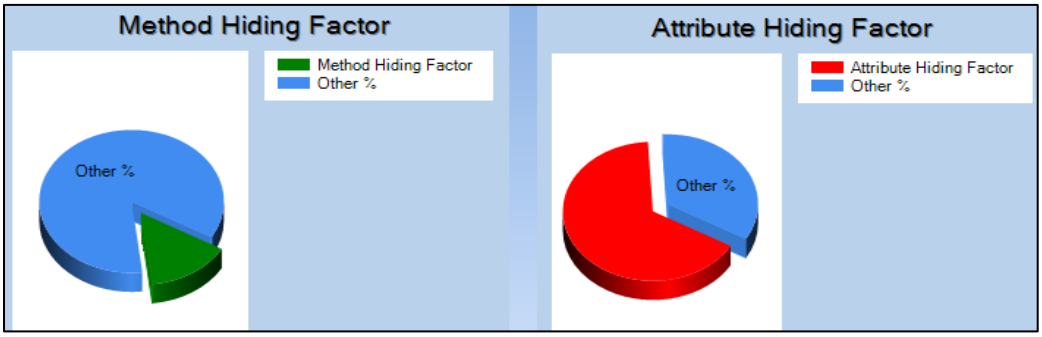


Figure (5-4) 3D-Pie Chart Sample for MHF and AHF Metrics.

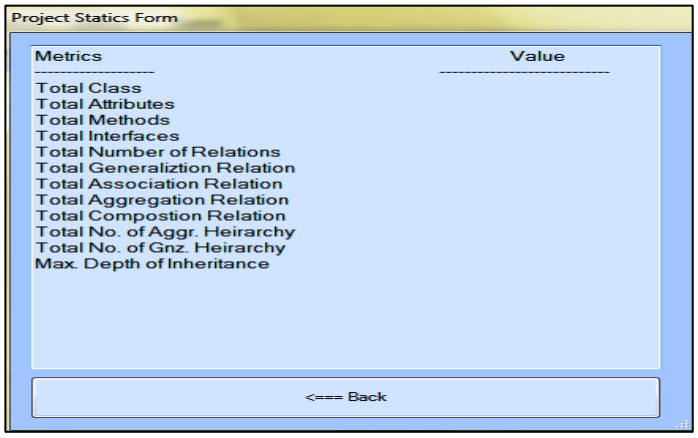


Figure (5-5) Design Statistics Form.

Figure (5-6) About Me Form.

The second component of KDM tool is the Tab Panel which contains three tabs. First is the Metrics tab which in turn has also three group boxes (class names, required information, and metrics value). Class names group box will load all class names in class diagram into the list box after clicking on the metrics button. See figure (5-7).

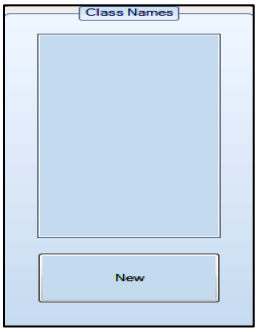


Figure (5-7) Class Names Group Box

New button in figure (5-7) is used to initialize KDM tool in order to open a new XMI document. So, this button will operate exactly as the "New" in the file menu strip. Required information group box is used to enter the name of the model, designer name, and model id. This information is used in the exporting process of the XML document because it is important in KRS and KDB tools. See figure (5-8).

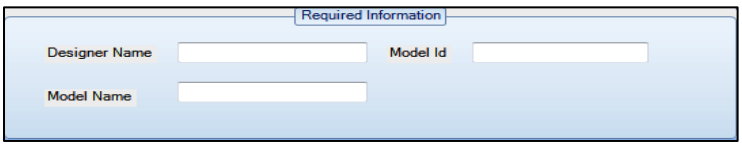


Figure (5-8) Required Information Group Box

Metrics value group box will contain all metrics values expressed in float numbers, and it will be loaded with the metrics value once clicking on the metrics button. See figure (5-9).

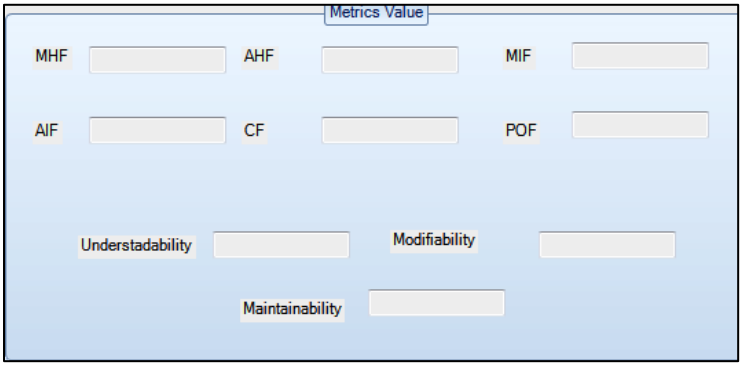


Figure (5-9) Metrics Value Group Box

The second tab is XMI tab which contains a rich text box. When pressing import XMI button, the XMI document will load into the rich text box. See figure (5-10).

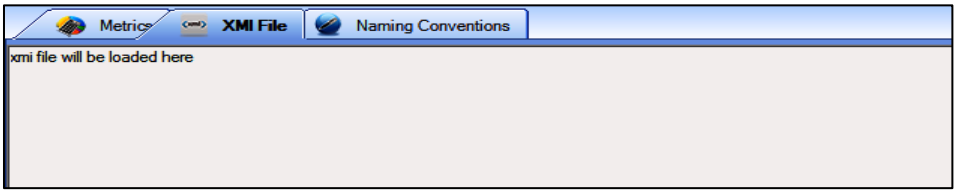


Figure (5-10) XMI File Tab.

The third tab is Naming Conventions tab which contains three group boxes; class information contains all classes that should be renamed to meet java design naming conventions. Methods information contains all methods that should be renamed, and finally attributes group box which contains the private and static attributes that should be renamed to meet java design naming conventions. See figure (5-11). For further information about java naming conventions, see Appendix B.

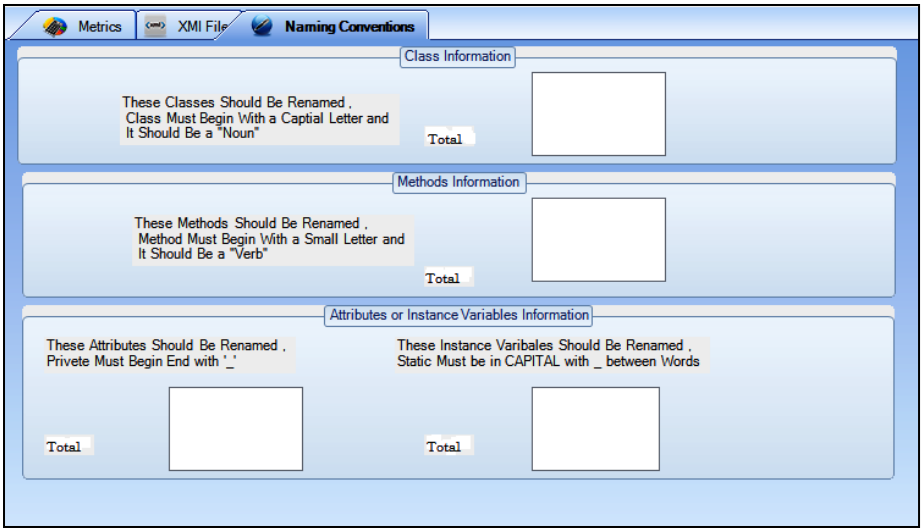


Figure (5-11) Naming Conventions Tab.

5.5 GUI of KRS Tool

After EA is started, KRS tool can be opened by clicking on Add-Ins→Khalil Package→KRS tool. See figure (5-12).

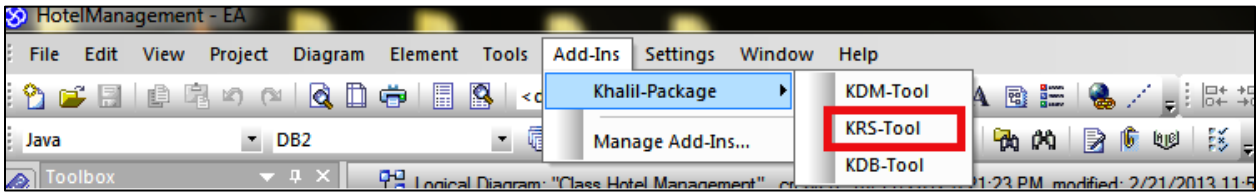


Figure (5-12) KRS Tool Menu Strip.

KRS tool will be opened, see figure (5-13) which represents the main window of KRS tool.

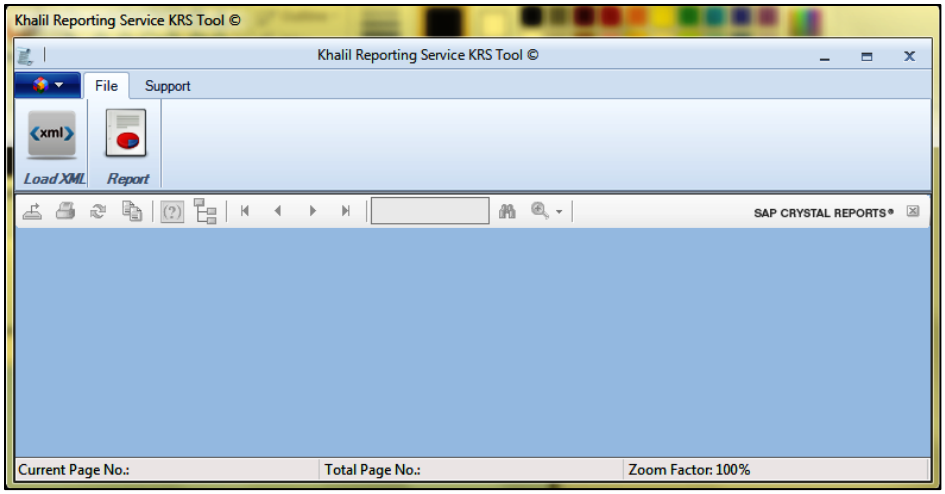


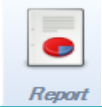
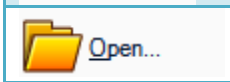
Figure (5-13) KRS Tool Main Window.

KRS tool has two components; a ribbon bar which contains file tab, support tab, and a menu strip option  which is depicted in figure (5-14).

Figure (5-14) KRS Tool File Menu Strip.

Table (5-3) contains each button of KRS tool with its description.

Table (5-3) KRS tool buttons description

Button	Description
	This button is used to import or load XML document generated by KDM tool and pass it to XML parser.
	This button is used to show the report in the crystal report tab. See figure (5-15)
	This button is used to initialize KRS tool to open a new XML document.
	This button operates the same as the load XML button; it is added for more functionality.

Exit	This button is used to close KRS tool.
About Me	This button is used to show information about the KRS tool like product name, version, copyrights and description. See figure (5-16).

The second component of KRS tool is the crystal report viewer which is depicted in figure (5-15).

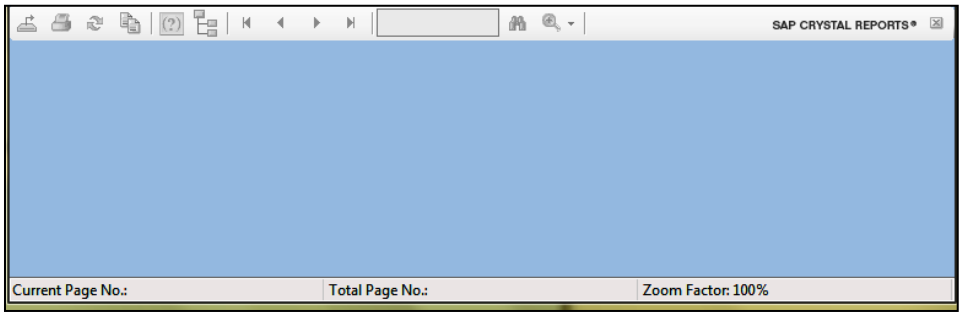


Figure (5-15) Crystal Report Viewer.

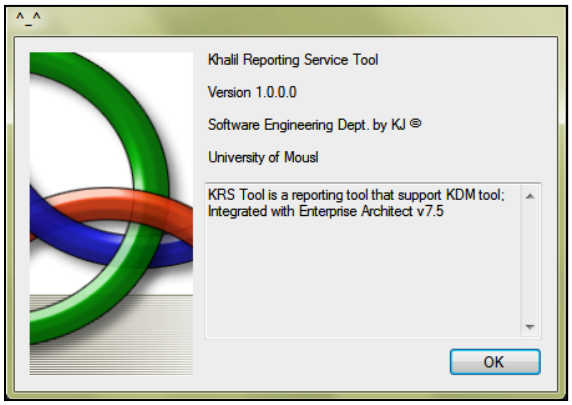


Figure (5-16) About Me Form for KRS Tool

5.6 GUI for KDB Tool

After starting EA, KDB tool can be opened by clicking on Add-Ins→Khalil Package→KDB tool. See figure (5-17).

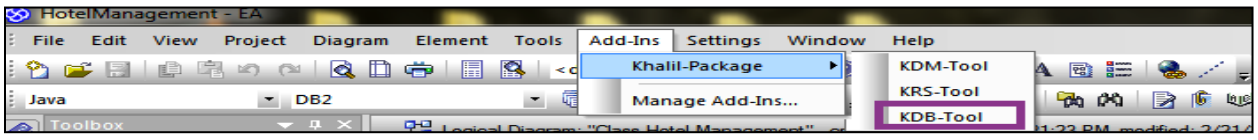


Figure (5-17) KDB Tool Menu Strip.

After pressing on KDB tool, the main window of KDB tool will appear which is depicted in figure (5-18). KDB tool also has two components; a ribbon bar which contains two tabs (database processing and support) and a  file menu strip which is depicted in figure (5-19).

Table (5-4) contains each button of KDB tool with their description.

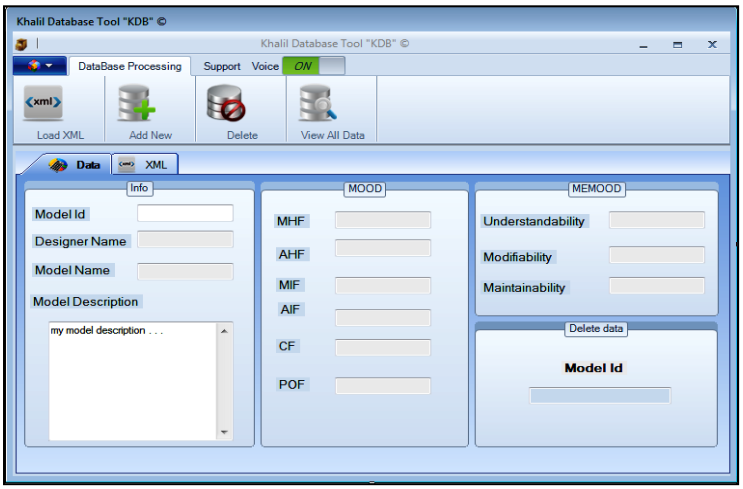
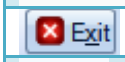
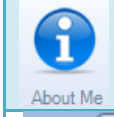
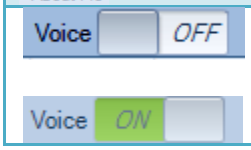


Figure (5-18) KDB Tool Main Window.

Figure (5-19) KDB Tool File Menu Strip.

Table (5-4) KDB Tool Buttons Description

Button	Description
	This button is used to import or load XML document generated by KDM tool and pass it to XML parser.
	This button is used to add new record (metrics) to the database.
	This button is used to delete a record from the database by specifying model id. See figure (5-20)

	This button is used to show all data (metrics) in the database. See figure (5-21).
	This button is used to initialize KDB tool to open a new XML document.
	This button is used to close KDB tool.
	This button is used to show information about the KDB tool like product name, version, copyrights and description, which is similar to KRS tool "About Me" form in figure (5-16).
	This switch button is used to turn the sound off. This switch button is used to turn the sound on.

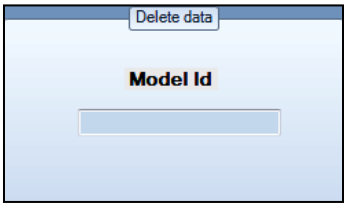


Figure (5-20) Delete Data Group Box.

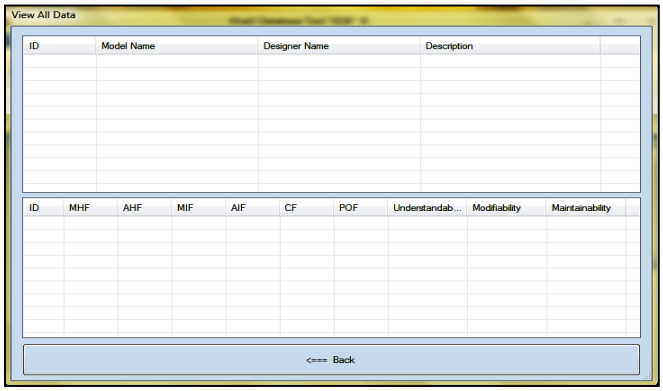


Figure (5-21) View Data Form.

The second component of KDB tool is the tab panel which contains two tabs; the first is the Data tab  which will be loaded by all metrics extracted from XML document into the text boxes of MOOD, MEMOOD and info group boxes, see figure (5-22) and (5-23).

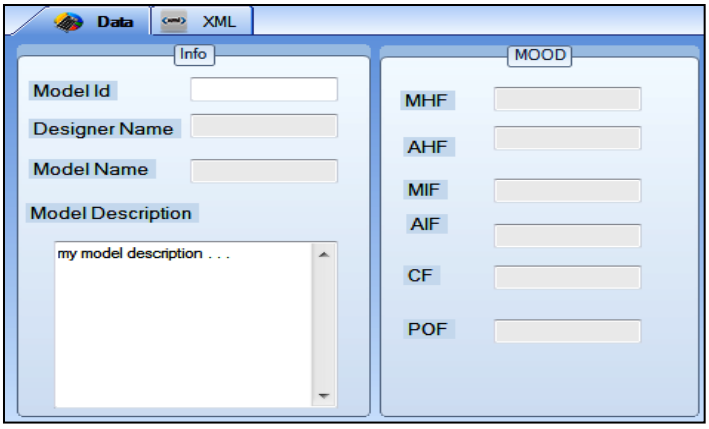


Figure (5-22) Data Tab Showing info and MOOD Group Boxes.

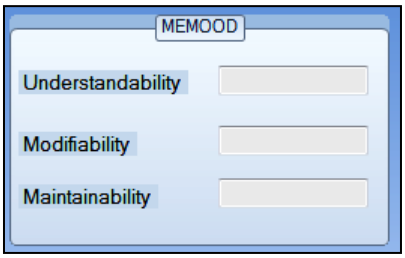


Figure (5-23) MEMOOD Group Box.

The second tab is XML tab  which contains a rich text box. When pressing load XML button, XML document will load into the rich text box. See figure (5-24).

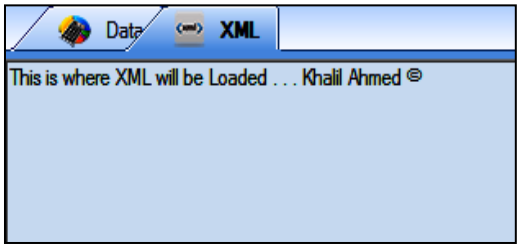


Figure (5-24) XML Tab.

5.7 Testing the Proposed Tools

A case study has been taken from [20] and modified so that all metrics can be calculated. The case study is about a Student Registration System at university. By using this system, students have access to the information of available courses, and they can also register in the system. The system is managed by a special user who is allowed to modify the required courses in the catalogue. This system was

modeled using EA v7.5. See figure (5-25) which represents the class diagram for the system. The complete XMI file is on the accompanying CD.

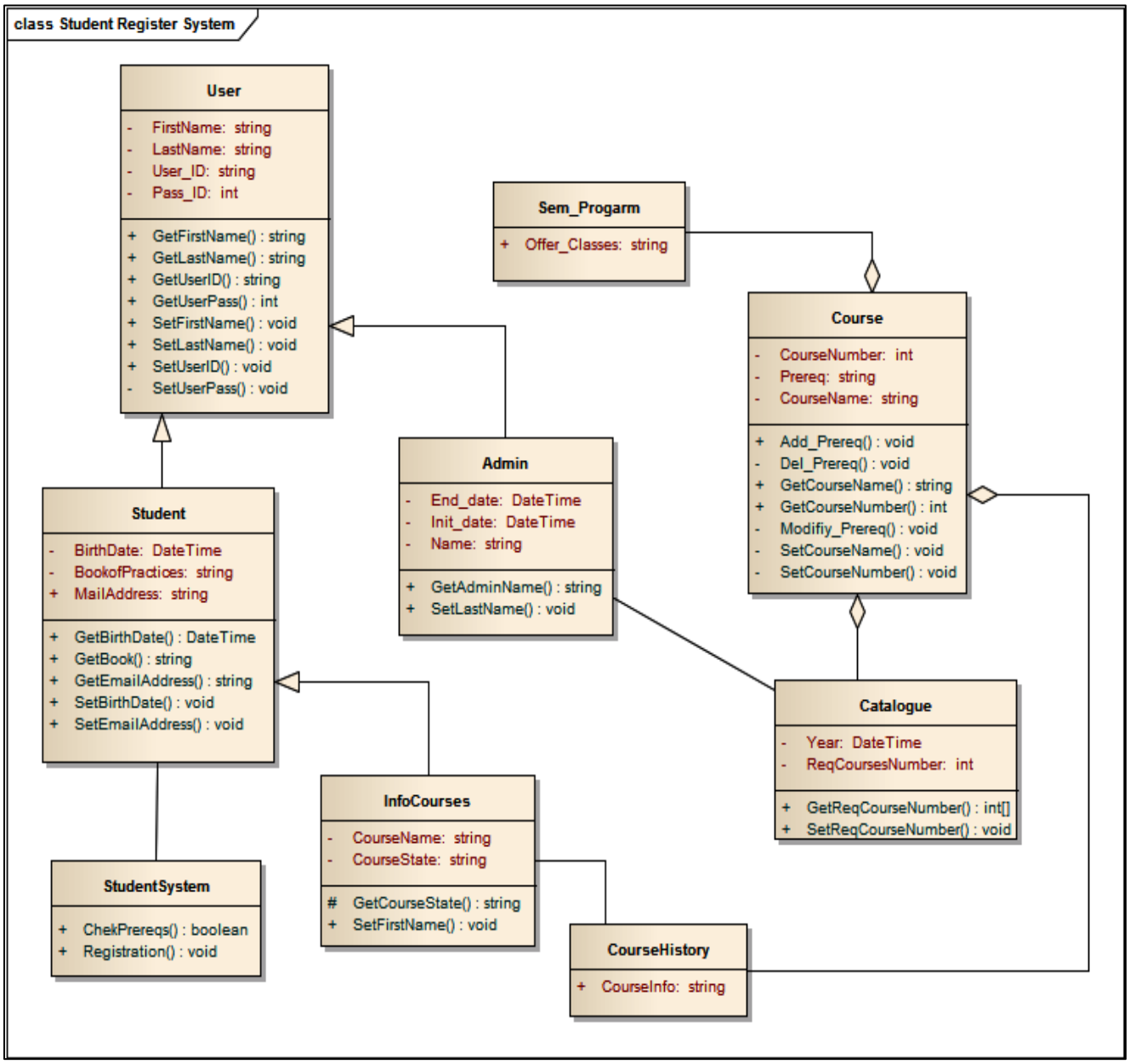


Figure (5-25) Student Registration System Class Diagram.

This class diagram is exported from EA as XMI document which will be the input for KDM tool, and by pressing on Metrics button the metrics are calculated. See figure (5-26).

Figure (5-26) Metrics for Student Registrations System.

It can be seen that class names group box is filled with all classes from the class diagram. MOOD and MEMOOD values are calculated. Design statistics can be seen in figure (5-27).

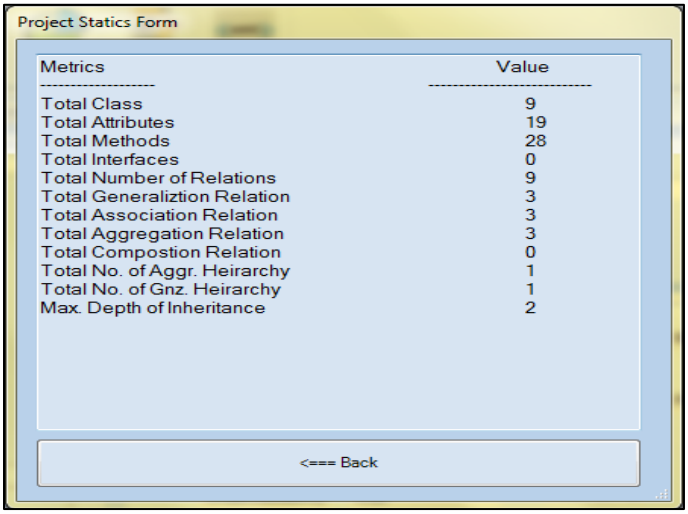


Figure (5-27) Design Statistics for the Class Diagram.

Form the statistics above, it can be seen that there are 9 classes, 19 attributes, 28 methods with 9 relations, one aggregation hierarchy, one generalization hierarchy and the maximum depth of inheritance is 2. Required information group box is used to export XML document that contains the metrics along with the model name, designer name and model id. This document is used as input for KRS and KDB tool. See figure (5-28) which shows the XML document for the system.

Visualization of metrics can be seen in figure (5-29), (5-30), (5-31), (5-32), (5-33), (5-34), (5-35), where red color means that the design is needed to be reviewed for that metric, green color means that the metric value is within the range and no review is needed.

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<!-- CLass Diagram Metrics  -->
- <Metrics>
   <Metric Id="1" />
   <DesignerName>Khalil Ahmed</DesignerName>
   <ModelName>Student Regestration Sys.</ModelName>
   <MHF>17.86</MHF>
   <AHF>84.21</AHF>
   <MIF>63.04</MIF>
   <AIF>55.56</AIF>
   <CF>11.11</CF>
   <POF>9.52</POF>
   <Understadability>3.08</Understadability>
   <Modifiability>3.83</Modifiability>
   <Maintainability>3.78</Maintainability>
</Metrics>
```

Figure (5-28) XML Document as Output from KDM Tool.

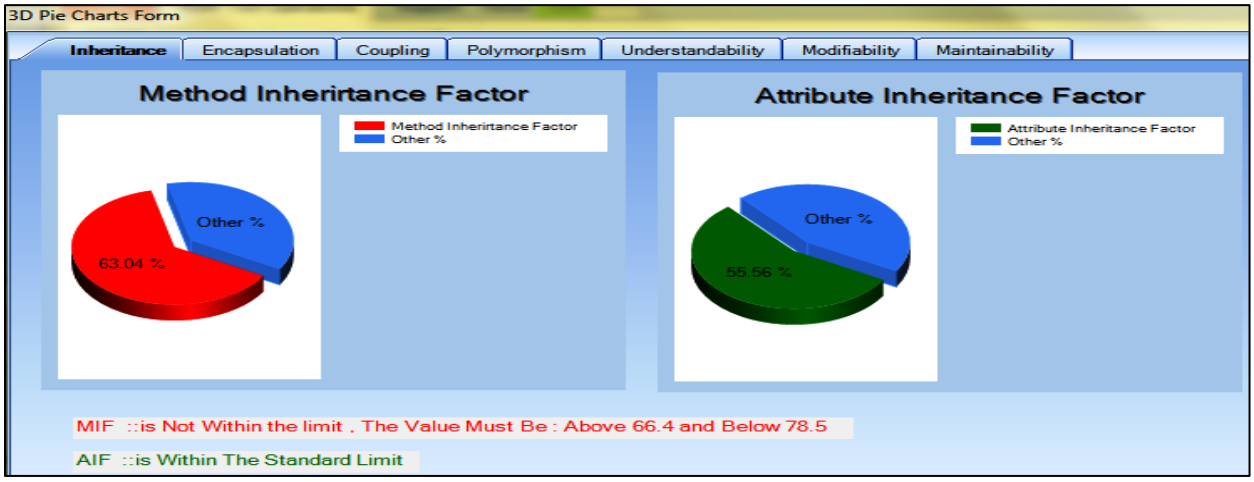


Figure (5-29) MIF and AIF Metrics 3D-Pie Chart.

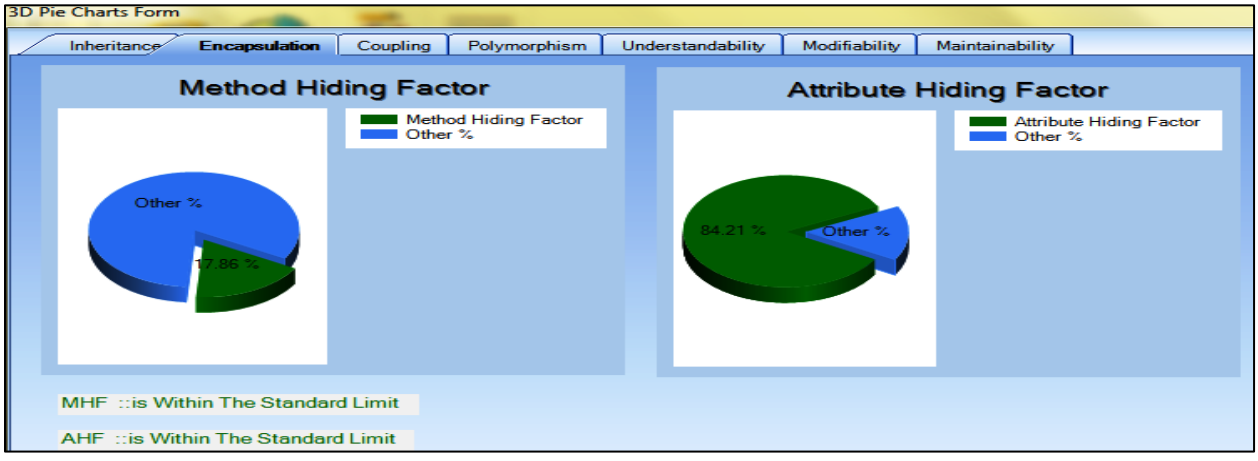


Figure (5-30) MHF and AHF Metrics 3D-Pie Chart.

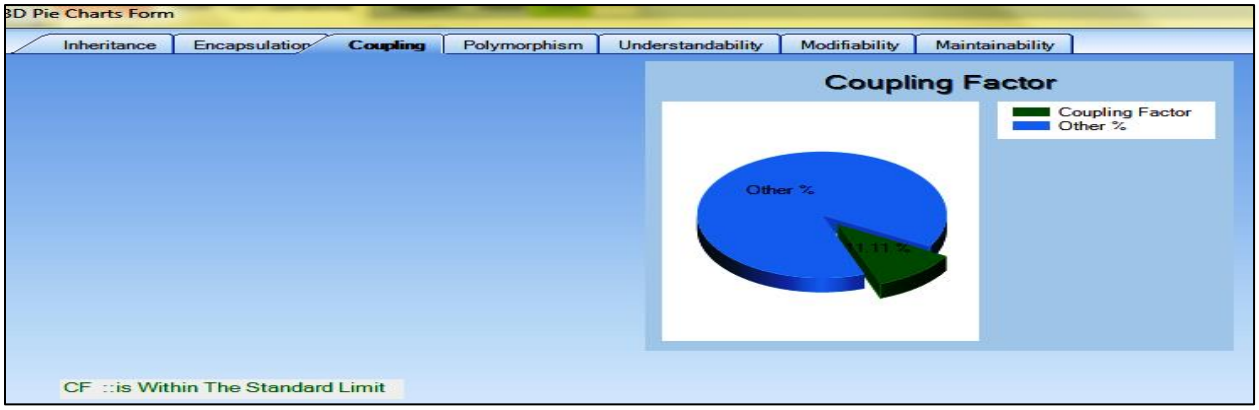


Figure (5-31) CF Metric 3D-Pie Chart..

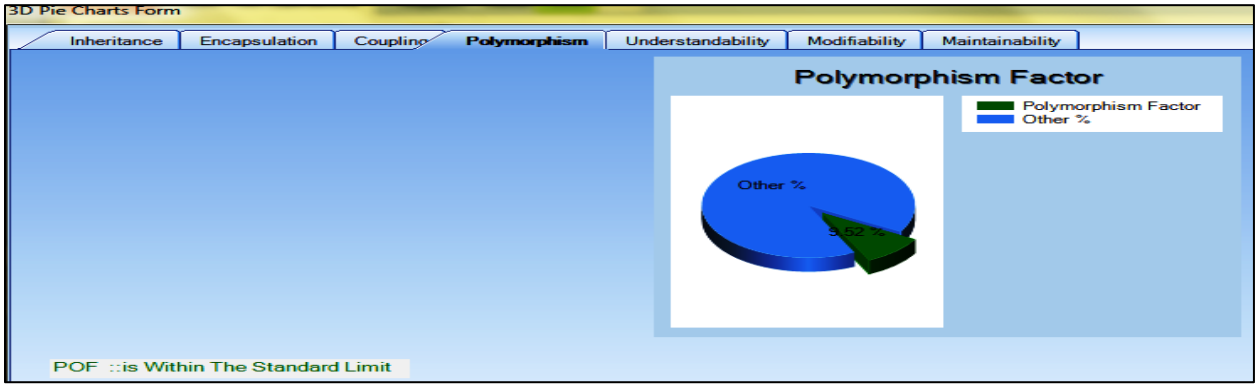


Figure (5-32) POF Metric 3D-Pie Chart.

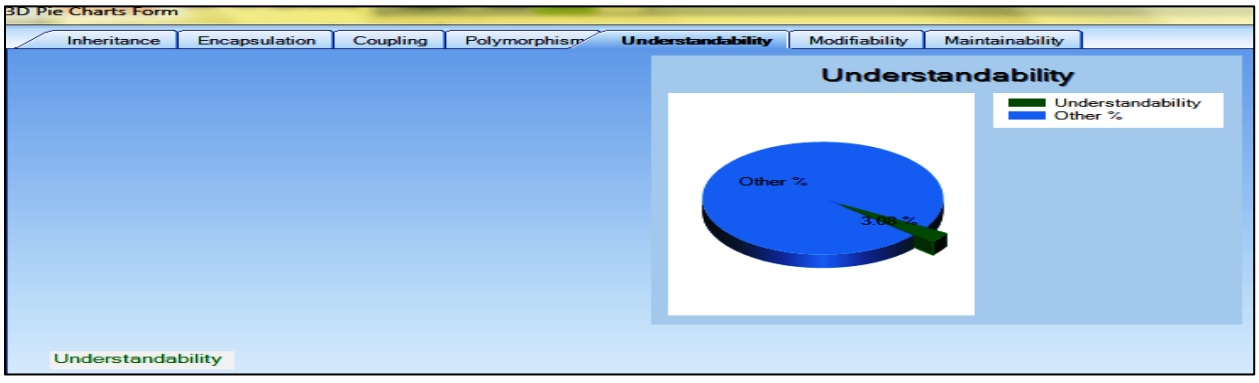


Figure (5-33) Understandability Quality Attribute 3D-Pie Chart.

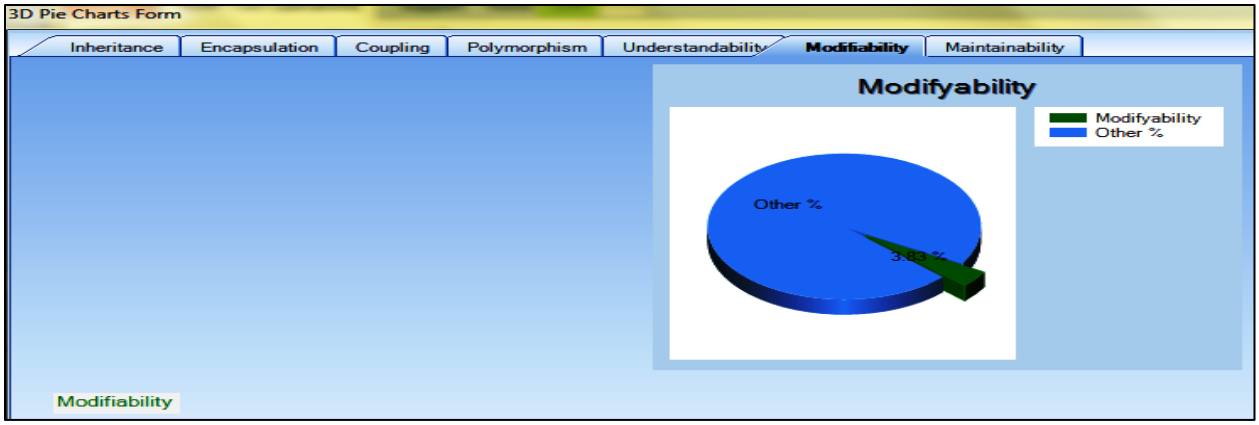


Figure (5-34) Modifiability Quality Attribute 3D-Pie Chart.

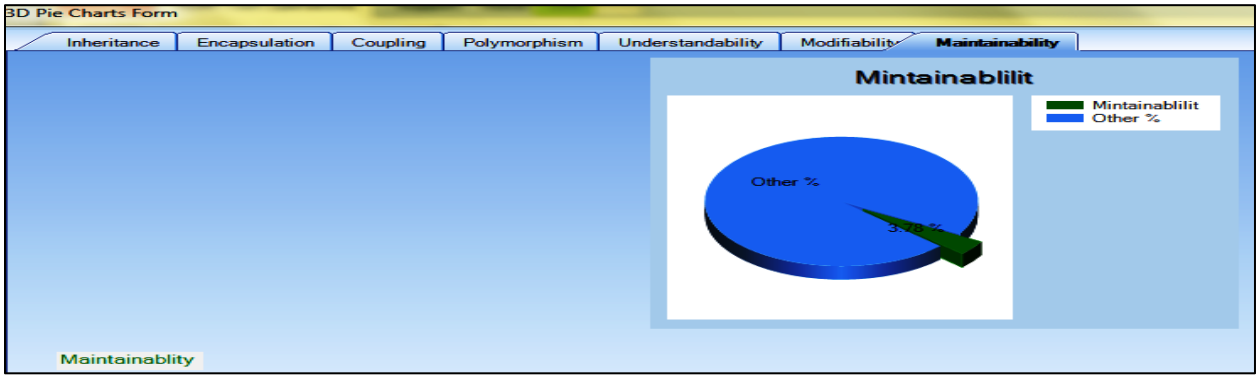


Figure (5-35) Maintainability Quality Attribute 3D-Pie Chart.

KDM tool supports JAVA design naming conventions of the design. See figure (5-36).

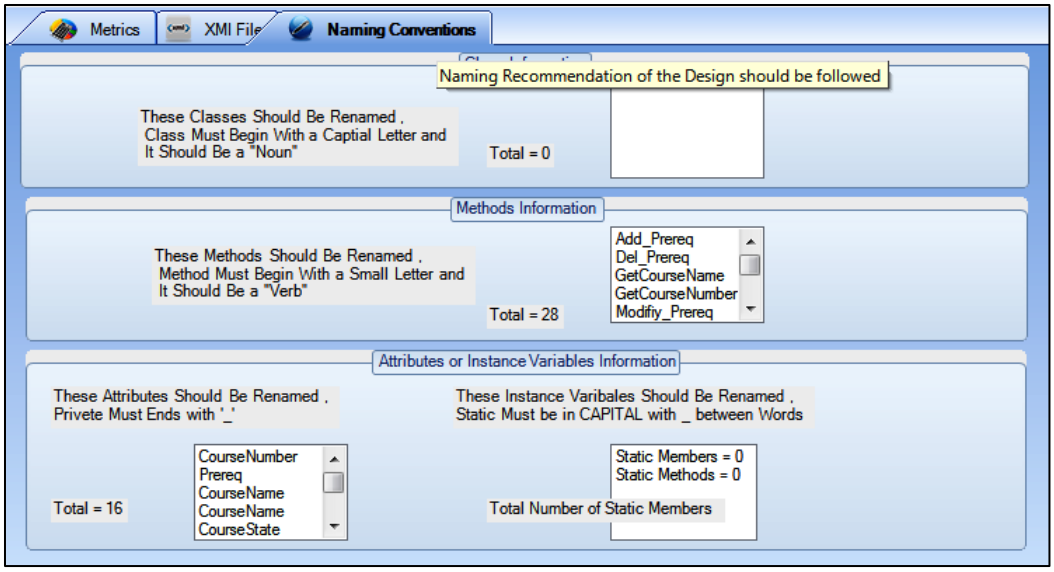


Figure (5-36) Naming Conventions.

By opening KRS tool and importing the XML document generated by KDM tool, the output is a crystal report. See figure (5-37).

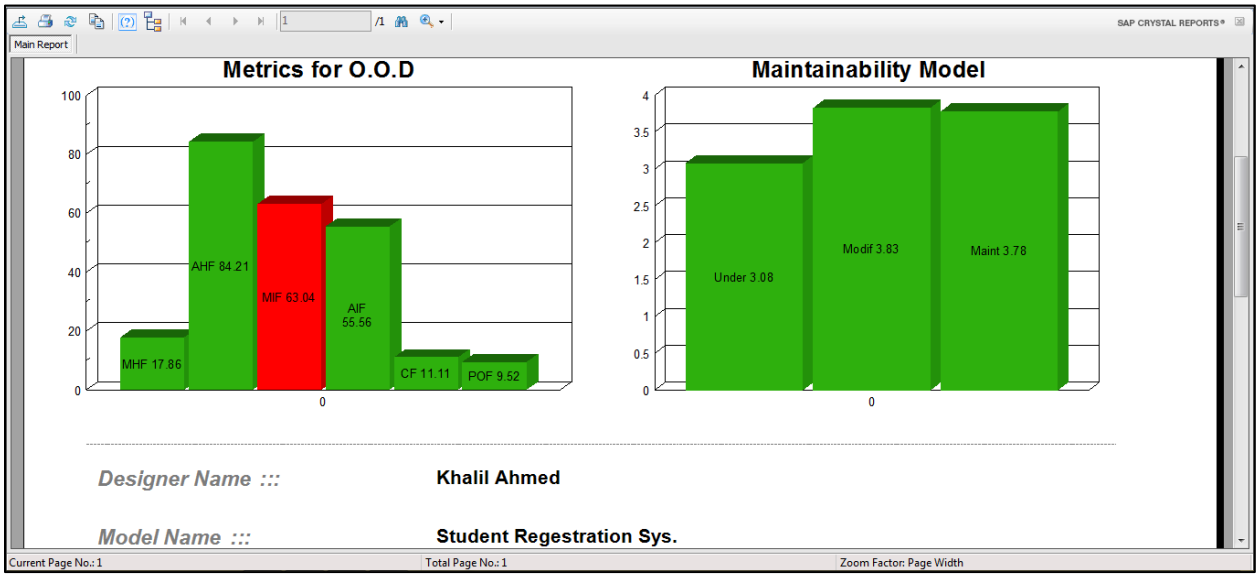


Figure (5-37) A Crystal Report for the Metrics of the System.

By opening KDB tool and importing XML document generated by KDM tool, XML parser will extract metrics and KDB tool will load them into the text boxes and into the XML tab . See figures (5-38), (5-39).

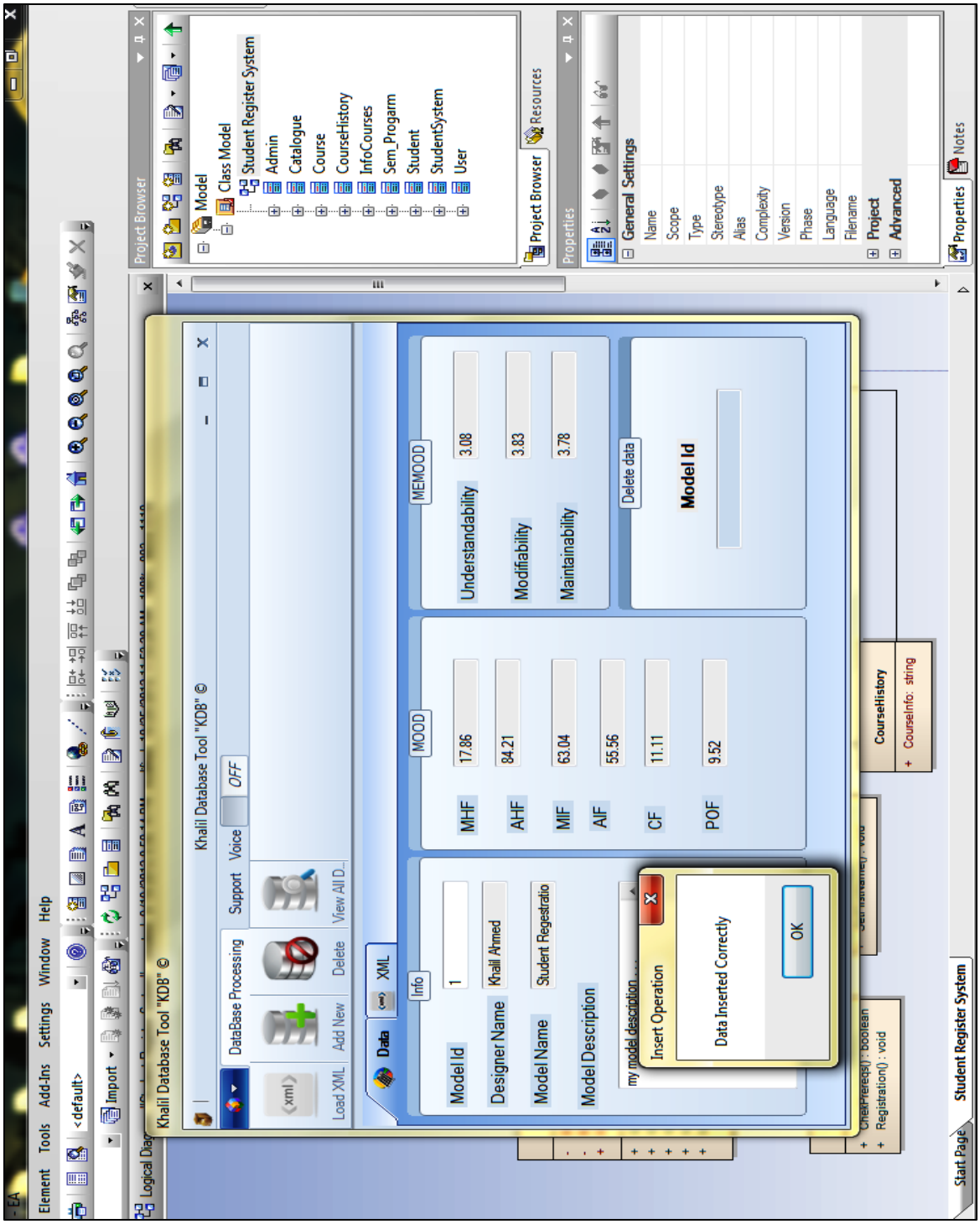


Figure (5-38) KDB Tool Output.

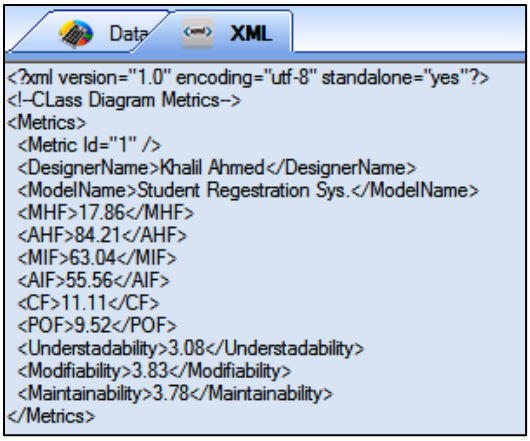


Figure (5-39) XML Document as KDB Tool Input.

By pressing on View All Data button, a new form will open and it will contain the metrics that are stored in the database. See figure (5-40).

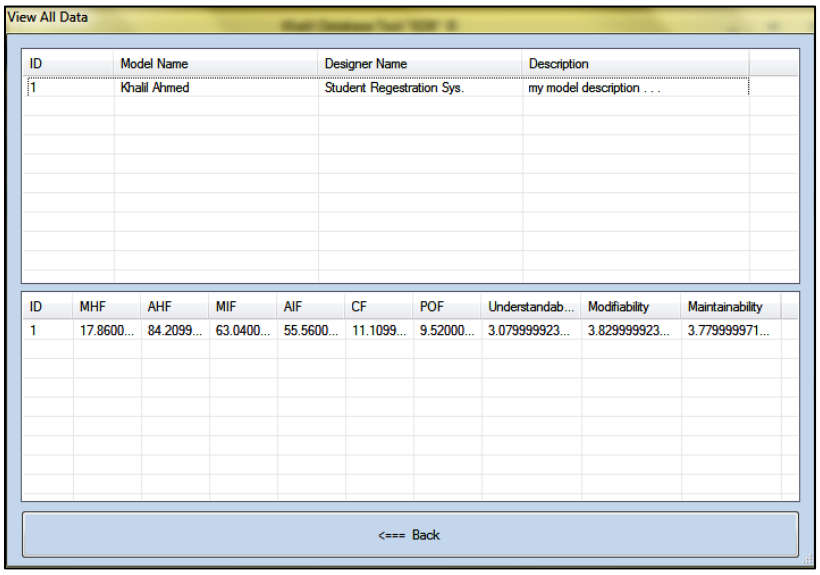


Figure (5-40) View All Data Form.

If it is needed to delete a record, the model id is written in the delete data group box, and by pressing on delete the record with that id will be deleted.

5.8 Discussion of Testing Results

KDM tool has succeeded in calculating MOOD and MEMOOD metrics and it gives 100% correct results, because the metrics are calculated by hand and have the same values of KDM tool. From the results of KDM tool, table (5-5) can be deduced.

Table (5-5) Metrics Discussion

Metric	Recommendation	Value	Within the limit	Outside the limit
MHF	No recommendation is needed	17.86	✓	-
AHF	No recommendation is needed	84.21	✓	-
MIF	**It is recommended that the number of inherited methods in the design should be reduced**	**63.04**	-	✓ √
AIF	No recommendation is needed	55.56	✓	-
CF	No recommendation is needed	11.11	✓	-
POF	No recommendation is needed	9.52	✓	-
Understandability	No recommendation is needed	3.08	✓	-
Modifiability	No recommendation is needed	3.83	✓	-
Maintainability	No recommendation is needed	3.78	✓	-

From the table above, it can be concluded that the design is fine and only MIF metric needs to be corrected.

5.9 Code Metrics

Total LOC of the proposed tools was equal to 4761. For KDM LOC was 2876, for KDB 1351 and for KRS 624.

5.10 Evaluation of the Proposed Tools

In this study, a questionnaire has been conducted by a twenty person who are considered as users of the proposed tools (programmers and software engineers). The samples were taken from people within the field (Computer Science and Software Engineering). The questionnaire divided into four sections namely:

1. Evaluating the tools generally,
2. Evaluating KDM Tool,
3. Evaluating KRS Tool, and
4. Evaluating KDB Tool.

Using SPSS program to get the results, see the table below.

Table (5-6) Questionnaire Results

Tool Name	Questionnaire Result
Evaluating the tools generally	**94.4**
Evaluating KDM Tool	**93.8**
Evaluating KRS Tool	**96.6**
Evaluating KDB Tool	**90.7**

Questionnaire form can be seen in Appendix F.

Chapter Six

Conclusion and Suggestions for Future Work

Conclusion and Suggestions for Future Work

6.1 Conclusion

Through the building and testing of KDM, KRS, and KDB tools, conclusions are:

1. KDM tool accepts XMI or XML documents generated by EA since EA export UML diagram as .XML or .XMI extension.

2. Documentation of metrics can help project managers or team leaders to monitor the progress by using KRS tool.

3. Storage of metrics can help designers to compare the metrics of some system with others. So, it can be used as a historical data by using KDB tool.

4. MOOD model help to identify problems of the design by means of metrics that uses the OO concepts which allow software engineers to early access software design and yet improve it.

5. MEMOOD model calculates understandability, modifiability, and maintainability of the design which are vital to know early in design phase.

6. Without XMI, no UML diagram can be described.

7. XML can be used as a bridge between tools or as intermediate data.

8. Generics in C# (Lists) are really important due to their dynamic allocation.

9. When EA does not support database or reports as add-in, integration must be used.

6.2 Suggestions for Future Work

Future works can be summarized as the follows:

- Developing an add-in for ArgoUML and StarUML to calculate metrics since they also do not support metrics for the design.
- Including other OOD metrics like CK and QMOOD (Quality Metrics for Object Oriented Design)
- Class diagram drawn by EA is default with java language, so KDB tool can be developed to calculate metrics of class diagram drawn by C#, and C++ languages.
- Evolving KDM tool to take not just XMI or XML as input but also the source code of Java, C# and C++.

References

References

1 Abreu, F.B., and Melo, W. , (1996), "Evaluating the Impact of Object-Oriented Design on Software Quality" , Proceedings of the 3rd International Software Metrics Symposium (METRICS'96), March, IEEE

2 Abreu, F.B., (1995), "Design Metrics for Object-Oriented Software Systems", ECOOP'95 Quantitative Methods Workshop in Aarhus, August

3 AGARWAL, B. B., TAYAL, S. P. , and GUPTA, M. , (2010), "SOFTWARE ENGINEERING & TESTING An Introduction" , Jones and Bartlett Publishers, ISBN: 978-1-934015-55-1

4 Ahmed, S.H. , Soliman, T.H.A., and Sewisy, A.A., (2013) , "A Hybrid Metrics Suite for Evaluating Object-Oriented Design" , International Journal of Software Engineering , Vol.6 , Issue-1, pp: 65-82, ISSN: 16876954

5 Albahari, J. and Albahari, B. , (2012) , "C# 5.0 in a nutshell " , 5th Edition , O'Reilly publishing , ISBN: 978-1-449-32010-2

6 Al-Zobaidy, L. M. and Ibrahim, K. A. , (2012) , "Existing Object Oriented Design Metrics a Study and a Comparison" , published in the 5th Scientfic Conference in Information Technology (CCIT), Computer Science and Mathematics ,Dec 19-20, University of Mosul , Iraq

7 Bellekens, G. , (2011) , "Tutorial: Create your first C# Enterprise Architect add-in in 10 minutes" , Sparx System Community

8 Booch, G., Maksimchuk, R.A., Engle, M.W., Young, B.J., Conallen, J., and Houston, K.A., (2007)," Object-Oriented Analysis and Design with Applications" , 3rd Edition, Addison Wesly, ISBN 0-201-8955

9 Cahill, J., Hogan, J.M., and Thomas, R., (2002) , "The Java Metrics Reporter – An Extensible Tool for OO Software Analysis", Software Engineering Conference, Ninth Asia-Pacific,pp: 507-516, ISBN: 07695185, IEEE

10 Calero, C., Genero, M., and Piattini,M. , (2002), "Empirical Validation of Class Diagram Metrics" ,Journal of Empirical Software Engineering, Proceedings of the International Symposium on Empirical Software Engineering (ISESE'02) , pp:195-203, ISBN: 07695179, IEEE

11 Calver, C., and Kulkarni, D. , (2009), "Essential LINQ", Addison-Wesly , ISBN :0-321-56416-2

12 Connolly, T.M., and Begg, C.E. , (2005), "Database systems : a practical approach to design,implementation, and management", 4th edition, chapter 25 ,Addison-Wesley ISBN: 321210255

13 Dennis, A. , Wixom, B.H., and Tegarden, D. , (2005), "Systems Analysis and Design with UML Version 2.0 An Object-Oriented Approach", 2nd edition , Wiley Publishing , ISBN 0471-34806-6

14 Dubey, S.K., Mittal, A., and Rawat, M.S., (2012), "Survey on Impact of Software Metrics on Software Quality", International Journal of Advanced Computer Science and Applications (IJACSA), Vol. 3, No-1, pp: 137-141, ISSN:21565570

15 El-Wakil, M., El-Bastawisi, A., Boshra, M. ,and Fahmy, A. , (2004), "Object-Oriented Design Quality Models A Survey and Comparison", In 2nd International Conference on Informatics and Systems (INFOS04) , March

16 Folwer, M., (2004), "UML Distilled a brief guide to the standard object modeling language", 3rd edition , Addison-Wesley , ISBN 0321193687

17 Galin, D. , (2004) , "Software Quality Assurance From Theory to Implementation " , Pearson Addison Wesly, ISBN 0201 70945 7

18 Genero, M., Olivas, J., Piattini, M., and Romero, F., (2001) ,"A Controlled Experiment for Corroborating the Usefulness of Class Diagram Metrics at the Early Phases of Object-Oriented Developments" Proc. of the ADIS a Workshop on Decision Support in Software Engineering, vol. 84. Spain

19 Genero, M., Piattini, M., and Calero, C., (2005), " A Survey of Metrics for UML Class Diagrams" , Journal of Object Technology (JOT) , Vol. 4, No-9 , pp. 59-92

20 Genero, M., Piattini, M., and Calero, C., (2002), "An Empirical Study to Validate Metrics for Class Diagrams", Department of Computer Science,University of Castilla-La Mancha, Spain

21 Ghosh, S., Dubey, S.K., and Rana, A., (2012), "Fuzzy Maintainability Model for Object Oriented Software System" ,International Journal of Computer Science Issues (IJCSI) , Vol. 9, Issue-4, No 2, pp:338-342, ISSN: 1694-0814

22 Girgis, M.R., Mahmoud, T.M., and Nour, R.R, (2009) , "UML Class Diagram Metrics Tool", International Conference on Computer Engineering & Systems , pp: 423-428, IEEE

23 Grose, T.J., Doney, G.C., and Brodsky, S.A., (2002) , "Mastering XMI Java Programming with XMI, XML, and UML ", John Wiley & Sons, ISBN: 0471384291

24 Hamilton, K., and Miles, R., (2006), "Learning UML 2.0" ,O'Reilly Publishing, ISBN 0-596-00982-8

25 Harrison, R. , Counsell, S.J., and Nithi, R.V. , (1998) , "An Evaluation of the Mood Set of Object-Oriented Software Metrics", IEEE Transaction Software Engineering, Vol.24, No- 6, pp:491–496, ISSN: 00985589

26 Harrison, R., and Counsell, S. , (1992), "Theoretical Validation and Empirical Evaluation of Object-oriented Design Metrics", IEEE Transactions on Software Engineering, Vol.18, Issue-5, pp: 410-422.

27 Hilera, J.R., and Fernández, L., (2012), "A Web Service for Calculating the Metrics of UML Class Diagrams" , Dobb's Special Report June , www.drdobbs.com, The world of software development

28 Hunter, D., Rafter, J., Fawcett, J., Vlist, E., Ayers, D., Duckett, J., Watt, A., and McKinnon, L., (2007), "Beginning XML", 4th Edition, Wiley Publishing , ISBN:978-0-470-11487-2

29 IEEE Standard Glossary of Software Engineering Terminology, (1990), IEEE-Std 610.12 , ISBN 1-55937-067

30 Jassim, F. , and Altaani, F. , (2013) , "Statistical Approach for Predicting Factors of Mood Method for Object Oriented", Uinversity of Irbid, Irbid, Jordan , arXiv preprint arXiv:1302.5454.

31 Kan, S.H., (2002) , "Metrics and Models in Software Quality Engineering" , 2nd Edition , Addison Wesley , ISBN:0-201-72915-6

32 Kaur, G., and Kumar, R., (2011), "Comparing Complexity in Accordance with Object Oriented Metrics" ,International Journal of Computer Applications ,Vol. 15, No-8, pp:42-45 ,ISSN: 09758887

33 Liu, J., Li, T., Zhang, S., and Li, M., (2010), "Research on Information Transformation Based on XMI" ,3rd International Conference on Computer Science and Information Technology, Vol. 6 pp: 356-359, IEEE

34 Mago, J. , and Kaur, P., (2012) , "Analysis of Quality of the Design of the Object Oriented Software using Fuzzy Logic", International Journal of Computer Applications (IJCA) , Vol. iRAFIT Issue: 3 , pp : 21-25

35 Mehra, S., and Maini, R., (2011), "A METRIC FRAMEWORK FOR ANALYSIS OF OOD" , Journal of Global Research in Computer Science , Vol. 2 , No-7, pp: 67-70 , ISSN :2229-371X

36 Mohapatra, P. K.J., (2010) ,"Software Engeineering (A Life Cycle Approach) ", New Age International Publisher , ISBN : 978-81-224-2846-9

37 Nentwich, C., Emmerich, W., Finkelstein, A., and Zisman, A. , (2000) , "BOX: Browsing Objects in XML",SoftwarePractice and Experience, Vol. 30 Issue: 15, pp: 1661-1676, ISSN: 00380644

38 Nugroho, A., Chaudron, M. R. V., and Heijstek, W., (2012), "How effective is UML modeling ?An empirical perspective on costs and benefits", Journal of Software & Systems Modeling, Vol.11, Issue-4, pp:571–580, ISSN: 16191366, Springer

39 O'Docherty, M., (2005), "Object-Oriented Analysis and Design Understanding System Developmentwith UML 2.0" , John Wiley & Sons , ISBN :10 0-470-09240-8

40 OMG, (2009), "OMG Unified Modeling LanguageTM (OMG UML) Superstructure Version 2.2" , http://www.omg.org/spec/UML/2.2/

41 Oosterman, J., Irwin, W., and Churcher, N., (2011), "EvoJava: A Tool for Measuring Evolving Software", In Proc. Australasian Computer Science Conference (ACSC 2011)

Perth, Australia , Vol.113 , pp:117-126

42 Paterson, T., Russell, C., and Dewar, R. , (2002), "Object-oriented software design metrics from XMI", School of Mathematical and Computer Science, Heriot-Watt University, Scotland

43 Peck ,G. , (2008), "The Complete Reference Crystal Reports 2008", McGraw Hill , ISBN: 0-07-159099-4

44 Pialorsi, P., and Russo, M., (2012) ,"Programming Microsoft LINQ in Microsoft .NET Framework 4", O'Reilly , ISBN:978-0-735-64057-3

45 Poornima, U.S. , (2011) , "A Quantitative Measure for Object Oriented Design Approach for Large-Scale Systems" , International Journal on Computer Science and Engineering (IJCSE) ,Vol. 3 No-9 , pp: 3237-3242, ISSN : 0975-3397

46 Pressman, R., (2001) ,"Software Engineering: A Practitioner's Approach" ,5th Edition, McGraw-Hill , ISBN: 0073655783

47 Pressman, R., (2011) ,"Software Engineering: A Practitioner's Approach" 7th Edition, McGraw-Hill , ISBN :978–0–07–337597–7

48 Rani, T., Sanyal, M., and Garg, S., (2012) , "Measuring Software Design Class Metrics:- A Tool Approach" , International Journal of Engineering Research & Technology (IJERT), Vol.1,Issue-7, ISSN: 2278-0181

49 Reißing, R., (2001), "Assessing the Quality of Object-Oriented Designs", OOPSLA, Institute of Computer Science, University of Stuttgart , Germany

50 Rizvi, S.W.A., Khan, R.A., (2010), "Maintainability Estimation Model for Object-Oriented Software in Design Phase (MEMOOD)" Journal of Computing, Vol. 2, Issue-4, ISSN: 2151-9617

51 Sainin, S. , (2004) , "Analysis of CK and MOOD Metrics for inconsistencies and prediction of Quality Attributes" , *a Master Thesis*,Coumputer Sciences and Engineering Department, Deemed University

52 Salem, A. M. , and Quresh, A. A. , (2011), "Analysis of Inconsistencies in Object Oriented Metrics" , Journal of Software Engineering and Applications (JSEA),Vol. 4,pp:123-128

53 Saradhi,M.V.V. , and Sastry,B.R., (2010), "Impact of Software Metrics on Object-Oriented Software Development Life Cycle" International Journal of Engineering Science and Technology Vol.2 , No-2, pp: 67-76 , ISSN: 0975-5462 67

54 Sarker, M., (2005), "An overview of Object Oriented Design Metrics", *a Master Thesis*, Department of Computer Science, Umeå University, Sweden

55 Sastry, J.S.V.R.S. , Ramesh, K.V., and Padmaja, M. , (2011), "Measuring Object-Oriented Systems Based On The Experimental Analysis Of The Complexity Metrics" , International

Journal of Engineering Science and Technology (IJEST) , Vol. 3 , No. 5 , pp: 3726-3731, ISSN : 0975-5462

56 Schach, S.R., (2011) ,"Object-Oriented and Classical Software Engineering" , 8th edition, McGraw Hill , ISBN: 978-0-07-337618-9

57 Schmuller, J., (2004), " Teach.Yourself UML In 24.Hours",3rd Edition, SAMS Publishing , ISBN: 0-672-32640-X

58 Sharma, A., and Dubey, S.K., (2012), "Comparison of Software Quality Metrics for Oriented Oriented System", International Journal of Computer Science & Management Studies (IJCSMS), Vol. 12, June 2012 , ISSN:2231 –5268

59 Sharma, A.K., Kalia, A. , and Singh, H. , (2012) , " Metrics Identification for Measuring Object Oriented Software Quality" ,International Journal of Soft Computing and Engineering (IJSCE), Vol. 2, Issue-5, November 2012 , ISSN: 2231-2307

60 Sharma, A.K., Kalia, A., and Singh, H. , (2012) ,"Empirical Analysis of Object Oriented Quality Suites",International Journal of Engineering and Advanced Technology (IJEAT) , Vol 1, Issue-4, April 2012, ISSN: 2249 – 8958

61 Shrivastava, D.P. , (2012), "Unit Test Case Design Metrics in Test Driven Development", Journal of Software Engineering, Scientific & Academic Publishing, Vol. 2, Issue 3 , pp:43-48

62 Sommerville, I. , (2011) , "Software Engineering" , 9th Edition , Addison Wesley, ISBN-13: 978-0-13-703515-1

63 Sommerville, I. ,(1992), "Software Engineering", 4th edition , Addison Wesley, ISBN : 9780201565294

64 Sparx Systems, (2009) , "Enterprise Architect User Guide v7.5 " , Enterprise Architect Software , http://www.sparxsystems.com/products/ea/7.5/

65 Sparx Systems, (2013) , "Enterprise Architect 10 Reviewer's Guide" , Sparx Systems Community , http://www.sparxsystems.com/resources/whitepapers/

66 Summers, B.L., (2011), " Software Engineering Reviews and Audits", CRC Press , ISBN:978-1-4398-5145-6

67 Troelsen, A. , (2012)," Pro C# 5.0 and the .NET 4.5 Framework", Sixth Edition, Apress Publishing , ISBN: 978-1-4302-4233-8

68 Vedros, K.G., (2011), "SAPHIRE 8 Quality Assurance Software Metrics Report ", Technical Report, Report-No. :INL/EXT-11-23138, U.S. Department of Energy OSTI

69 Web page ,Class diagram examples, 2012, http://www.programsformca.com/2012/03/uml-diagrams-for-hospital-mgmt-system

70 Web page, http://msdn.microsoft.com/en-us/library/1ez7dh12.aspx

71 Web page, http://msdn.microsoft.com/en-us/library/d1587c1h.aspx

72 Web page, http://msdn.microsoft.com/en-us/library/dtba4t8b.aspx

73 Web page, http://www.microsoft.com/com/default.mspx

74 Web page, http://www.sparxsystems.com/enterprise_architect_user_guide/9.3/index.html

75 Web page, http://www.sparxsystems.com/enterprise-architect/index.html

76 Web page,http://msdn.microsoft.com/en-us/library/dt80be78.aspx

77 Web page, http://support.microsoft.com/kb/256986

78 Website, http://www.omg.org/spec/UML/Current

79 Website, http://www.sparxsystems.com/

80 Website, http://www.uml-diagrams.org/uml-24-diagrams.html

81 Website, Oracle, JAVA Naming Conventions, http://www.oracle.com/technetwork/java/javase/documentation/codeconventions-135099.html#367

82 Wixom, B.H., Dennis, A., and Roth, R. M. , (2012) , "System Analysis And Design", 5th Edition , John Wiley & Sons, ISBN :978-1-118-05762-9

83 Xie, H., Jiang, K., Yuan, X., and Zeng, H. , (2012) ,"Forensic Analysis Of Windows Registry Against Intrusion", International Journal of Network Security & Its Applications (IJNSA), Vol.4, Issue.2, pp: 121-134, ISSN: 09752307

84 Xiong, J., (2011) , "New Software Engineering Paradigm Based on Complexity Science -An Introduction to NSE" , Springer Publishing ,ISBN 978-1-4419-7325-2

Appendix

LINQ to XMI and Lambda Expressions

Language Integrated Query (LINQ)

LINQ is a programming model that introduces queries as a first class concept into any Microsoft .NET Framework language. These language extensions boost developer productivity, thereby providing a shorter, more meaningful, and expressive syntax with which to manipulate data. LINQ provides a methodology that simplifies and unifies the implementation of any kind of data access [49].

LINQ made its first appearance in September 2005 as a technical preview. Since then, it has evolved from an extension of Microsoft Visual Studio 2005 to an integrated part of .NET Framework 3.5 and Visual Studio 2008, both released in November 2007. The first released version of LINQ directly supported several data sources. Now with .NET Framework 4 and Visual Studio 2010, LINQ also includes LINQ to Entities, which is part of the Microsoft ADO.NET Entity Framework, and Parallel LINQ (PLINQ). Current LINQ implementations from Microsoft for accessing several different data sources, such as the following [44]:

- LINQ to Objects
- LINQ to ADO.NET
- LINQ to Entities
- LINQ to SQL
- LINQ to DataSet
- LINQ to XML

It can be deduced that LINQ is very efficient, simplifies a lot of things, and saves a lots of line of code. LINQ syntax is similar to SQL statements in some way, LINQ query syntax is depicted in figure (A-1).

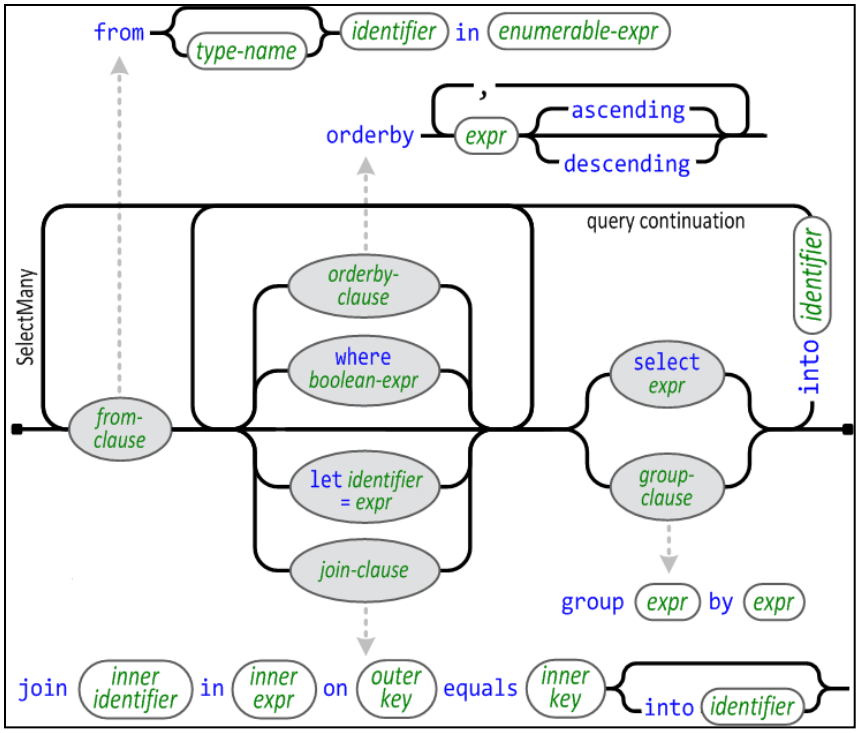


Figure (A-1) LINQ Query Comprehension Syntax [5]

LINQ to XML

The .NET Framework provides a number of APIs (Application Programming Interface) for working with XML data. From Framework 3.5, the primary choice for general-purpose XML document processing is LINQ to XML. LINQ to XML comprises a lightweight LINQ-friendly XML document object model, plus a set of supplementary query operators. LINQ to XML is supported fully in the Metro profile [5].

LINQ to XML provides support for querying, creating, and transforming XML documents. XML namespaces are included in the API, as well as support for XML schemas. LINQ to XML stands on its own as a compelling alternative to technologies such as XPath, XQuery, XSLT, and the XML DOM (Data Object Model). [11]

XMI parser that is built in this study uses LINQ to XML; the following c# code is a sample of the XMI parser class.

```
public List<List<string>> Get_ALL_Method_Names(List<string> ClassName)
{
    List<List<string>> Result = new List<List<string>>();
    List<string> Result_sub;
    for (int i = 0; i < ClassName.Count; i++)
    {
        var q = from p in MyXmiFile.Descendants(UML + "Class")
                where p.Attribute("name").Value == ClassName[i]
                select p;


        var q_Methods = from p in q.Descendants(UML + "Operation")

                        let pblic = p.Attribute("visibility").Value == "public"
                        let prvt = p.Attribute("visibility").Value == "private"
                        let prot = p.Attribute("visibility").Value == "protected"

                        where pblic || prvt || prot
                        select p.Attribute("name").Value;


        Result_sub = new List<string>((q_Methods.ToList()));
        Result.Add(Result_sub);


    }

    return Result;

}
```

The above method is used to get all operations for each class.

```
public int TotalNumberOfMethods() // total Number of Opeartions in whole Class Daigram
{
    var q = from p in this.MyXmiFile.Descendants(UML + "Operation")
            select p.Attribute("name").Value;

    return q.Count();
}


public int TotalNumberOfInterface() // total Number of interfaces in the model
{
    var q = from p in this.MyXmiFile.Descendants(UML + "Interface")
            select p.Attribute("name").Value;

    return q.Count();

}
```

The above method is used to gets the total number of methods and interfaces in class diagram.

Lambda Expressions

C# supports the ability to handle events “inline” by assigning a block of code statements directly to an event using anonymous methods, rather than building a stand-alone method to be

called by the underlying delegate. Lambda expressions are nothing more than a very concise way to author anonymous methods and ultimately simplify how we work with the .NET delegate type [67]

See the following code sample to understand how Lambda works.

```
namespace LmbdaConsole
{ // Code Sample By Khalil Ahmed . MSc. Student . Software Engineering
    class Program
    {
        static bool IsEvenNumber(int number)
        {
                return (number%2==0);
        }
        static void Main(string[] args)
        {
            List<int> NumbersList = new List<int>();
            NumbersList.AddRange(new int[] {20,5,7,4});

            Console.WriteLine("<<First Attempt using conventional programming>>");
            for (int i = 0; i < NumbersList.Count; i++)
                if (IsEvenNumber(NumbersList[i]))
                    Console.WriteLine(NumbersList[i]);
            // output 20 4

            Console.WriteLine("<<Second Attempt using Lambda Expresions>>");
            List<int> Result = NumbersList.FindAll(i => i % 2 == 0);
            // output 20 4
            foreach (int i in Result) { Console.WriteLine(i); }

            Console.ReadKey();
        }
    }
}
```

It can be seen from the code sample above the way that Lambda simplifies the work and allows defining a method in the middle of the code without needing to create a standalone method like IsEvenNumber method.

In this study, Lambda expressions were used. See the following code sample which is used to get the source and target classes for the aggregation relationship for the XMI sample in figure (A-2).

```
private static string ReadTaggedValue(XElement values, string tag)
{
    return values.Elements(UML + "TaggedValue")
        .Where(e => (string)e.Attribute("tag") == tag)
        .Select(e => (string)e.Attribute("value")).Single();
}

public List<string> Get_Source_ByRelation(string relation = "Association", string targetType = "Aggregation")
{
    List<string> myList = new List<string>();

    var associations =
        from p in this.MyXmiFile.Descendants(UML + relation)
        let values = p.Element(UML + "ModelElement.taggedValue")
        where XmiParser.ReadTaggedValue(values, "ea_type").Equals(targetType) == true
        select new
        {
            ea_sourceName = ReadTaggedValue(values, "ea_sourceName"),
        };

    foreach (var i in associations)
        myList.Add(i.ea_sourceName.ToString());

    return myList;
}
```

ReadTaggedValue method uses Lambda expression to get the value of both tag and value.

```
- <UML:Association xmi.id="EAID_014EFF8F_BCDD_4481_8829_F5D2D5173EEF" visibility="public"
  - <UML:ModelElement.taggedValue>
      <UML:TaggedValue tag="style" value="3" />
      <UML:TaggedValue tag="ea_type" value="Aggregation" />
      <UML:TaggedValue tag="direction" value="Source -> Destination" />
      <UML:TaggedValue tag="linemode" value="3" />
      <UML:TaggedValue tag="linecolor" value="-1" />
      <UML:TaggedValue tag="linewidth" value="0" />
      <UML:TaggedValue tag="seqno" value="0" />
      <UML:TaggedValue tag="headStyle" value="0" />
      <UML:TaggedValue tag="lineStyle" value="0" />
      <UML:TaggedValue tag="ea_localid" value="8" />
      <UML:TaggedValue tag="ea_sourceName" value="C" />
      <UML:TaggedValue tag="ea_targetName" value="Root1" />
      <UML:TaggedValue tag="ea_sourceType" value="Class" />
      <UML:TaggedValue tag="ea_targetType" value="Class" />
      <UML:TaggedValue tag="ea_sourceID" value="17" />
      <UML:TaggedValue tag="ea_targetID" value="14" />
```

Figure (A-2) XMI sample for aggregation relationship

JAVA Design Naming Conventions

Java design naming conventions follows exactly the same naming conventions for JAVA code.

Naming conventions make programs more understandable by making them easier to read. They can also give information about the function of the identifier-for example, whether it is a constant, package, or class-which can be helpful in understanding the code [81].

Naming conventions for each identifier type is in table (B-1).

Table (B-1) Java Naming Conventions [81]

Identifier Type	Rules for Naming	Examples
Classes	Class names should be nouns, in mixed case with the first letter of each internal word capitalized. Class names should be simple and descriptive. Using whole word, acronyms and abbreviations are avoided unless the abbreviation is much more widely used than the long form, such as URL (Uniform Resource Locator) or HTML.	Class Person;
Interfaces	Interface names should be capitalized like class names, in C# the interface must starts with a capital I.	interface Storing;
Methods	Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized.	getSalary();
Variables	Except for variables, all instance, class, and class constants are in mixed case with a lowercase first letter. Internal words start with capital letters. Variable names should not start with underscore _ or dollar sign $ characters, even though both are allowed. Variable names should be short yet meaningful. The choice of a variable name should be mnemonic- that is, designed to indicate to the casual observer the intent of its use. One-character variable names should be avoided except for temporary "throwaway" variables. Common	int i; char c;

	names for temporary variables are i, j, k, m, and n for integers; c, d, and e for characters	
Constants	The names of variables declared class constants and of ANSI (American National Standard Institute) constants should be all uppercase with words separated by underscores ("_"). (ANSI constants should be avoided, for ease of debugging.)	static int MIN_WIDTH = 4;

So, by following coding conventions, class diagram will be much more readable and ease coding for the programmer when he receives the class diagram (detailed).

Case Studies

In this appendix, two case studies are presented [69]:

1. Hospital Management System.
2. Hotel Management System.

Both systems are modeled as class diagram using EA v7.5 and used to test KDM tool along with Student Registration System.

Hospital Management System (HMS)

Class diagram for the system can be depicted in figure (C-1).

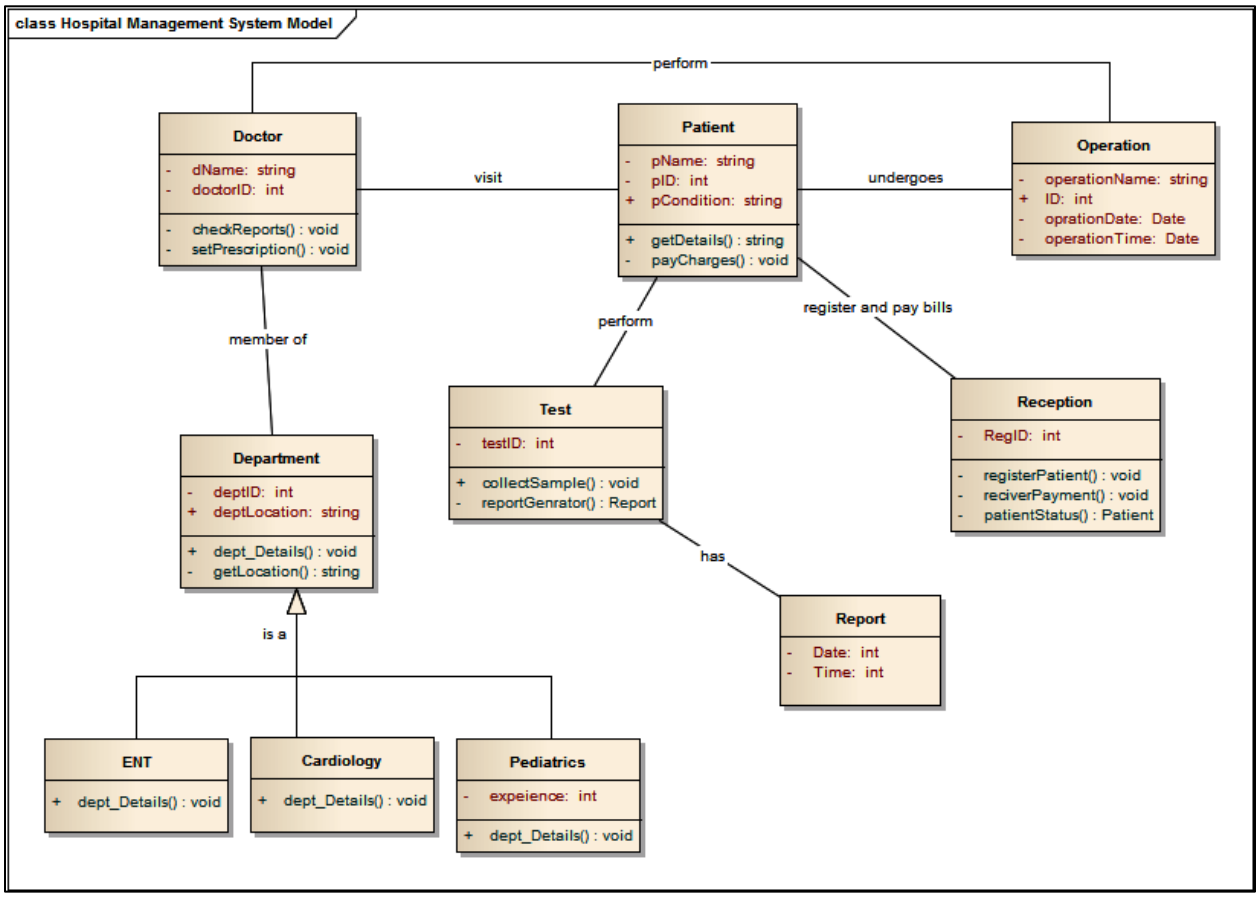


Figure (C-1) HMS Class Diagram.

Using XMI document that is generated by EA for HMSs class diagram as input to KDM tool, KDM tool will calculate metrics. See figure (C-2).

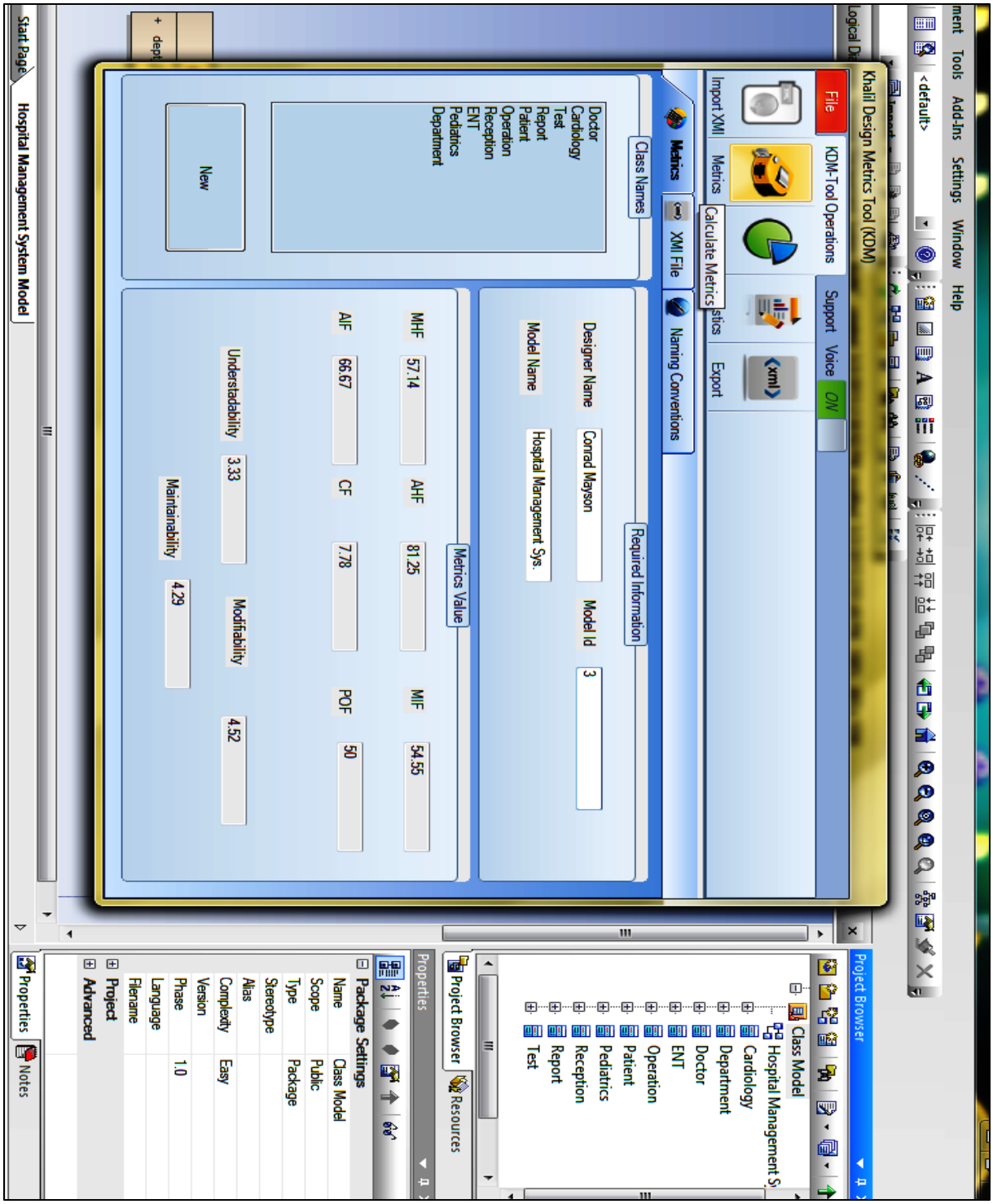


Figure (C-2) HMS Metrics

XMI file is loaded in the XMI tab. See figure (C-3).

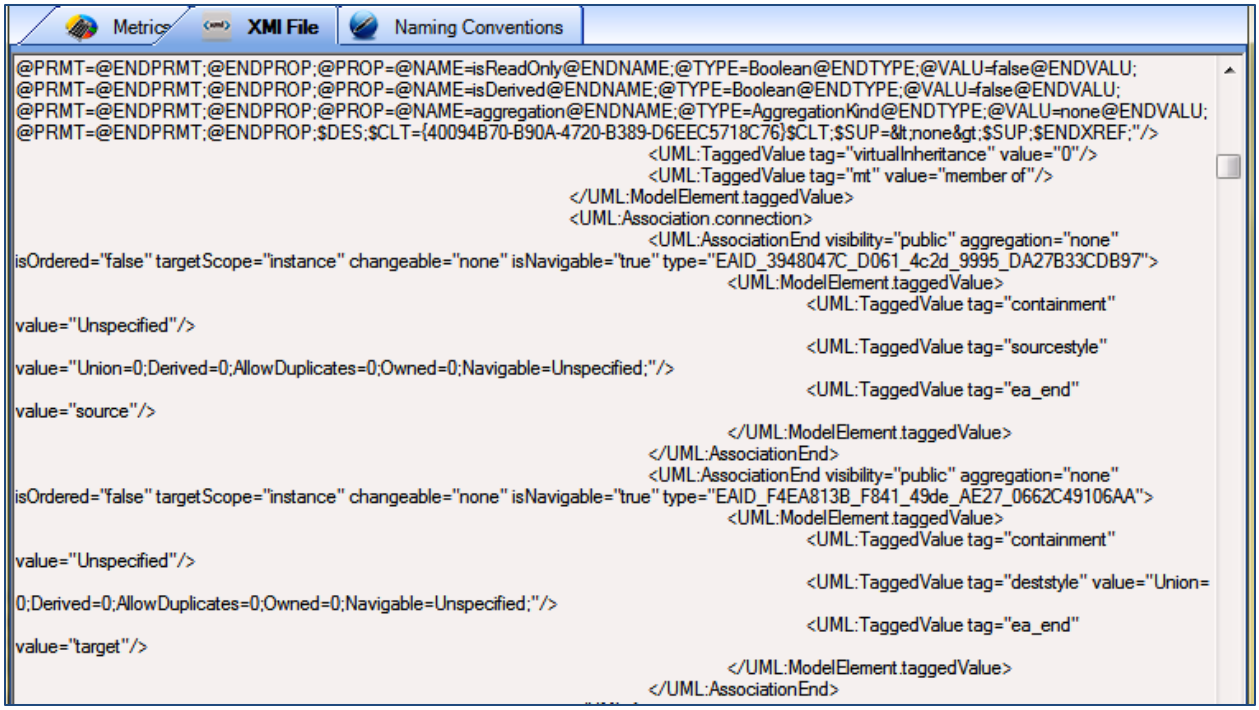


Figure (C-3) HMS as XMI document.

Design statistics can be seen in figure (C-4).

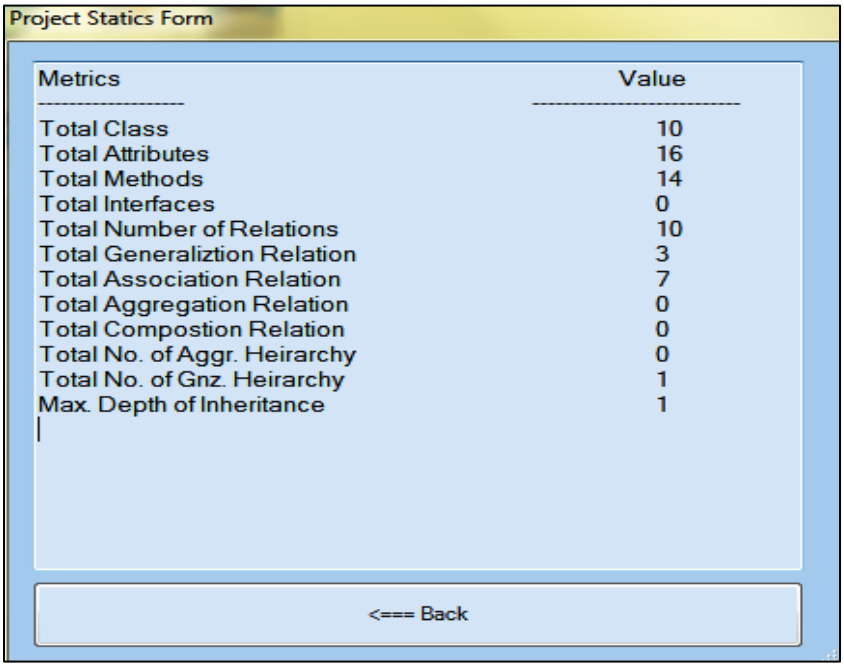


Figure (C-4) Design Statistics for HMS

Metrics are visualized in figures (C-5), (C-6), (C-7), (C-8), (C-9), (C-10), (C-11).

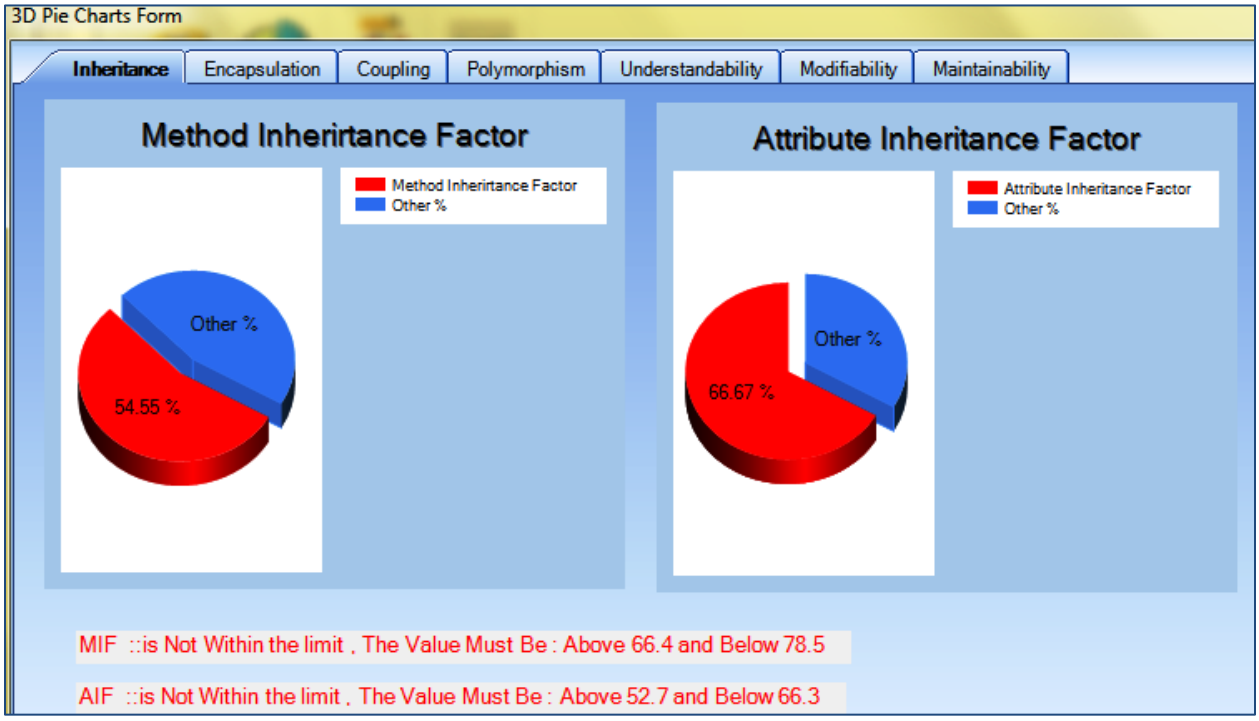


Figure (C-5) MIF and AIF 3D-Pie Chart.

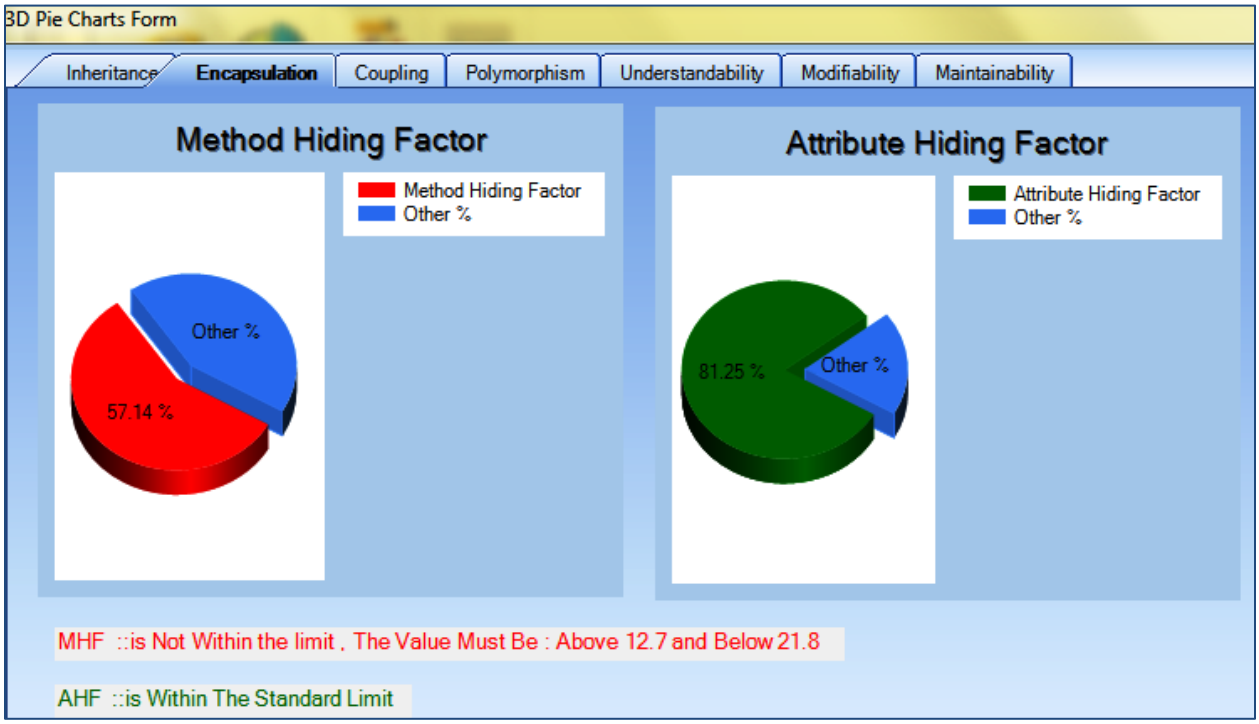


Figure (C-6) MHF and AHF 3D-Pie Chart.

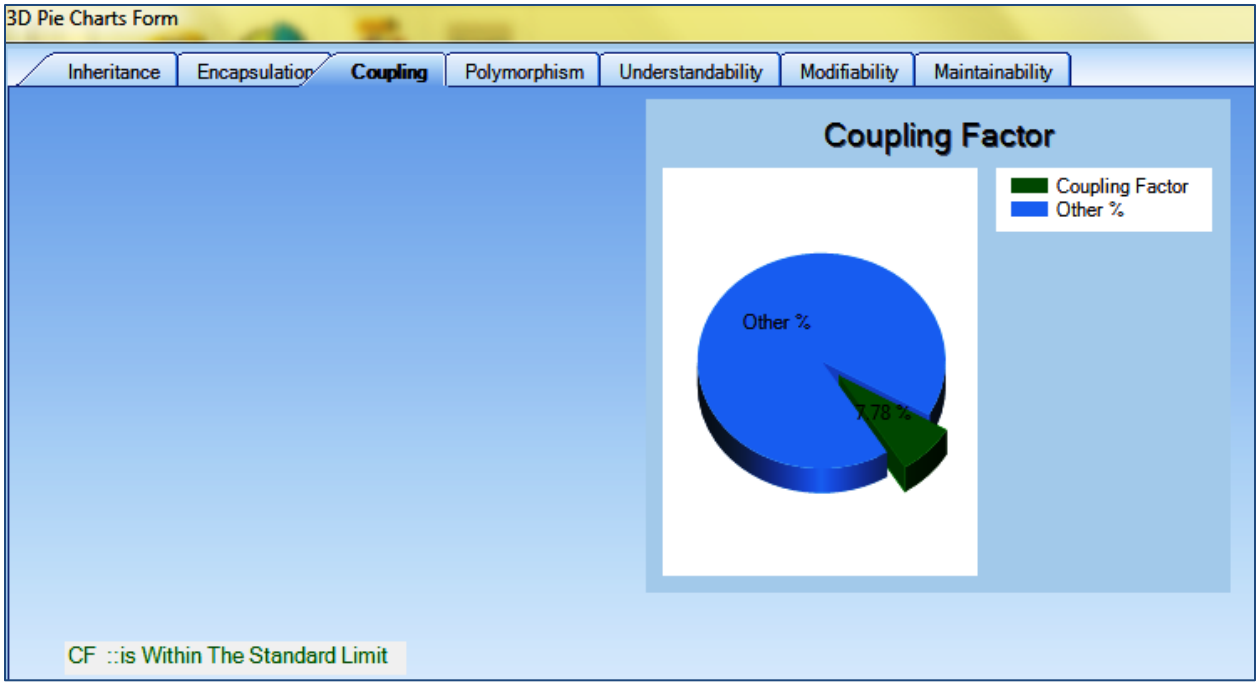


Figure (C-7) CF 3D-Pie Chart.

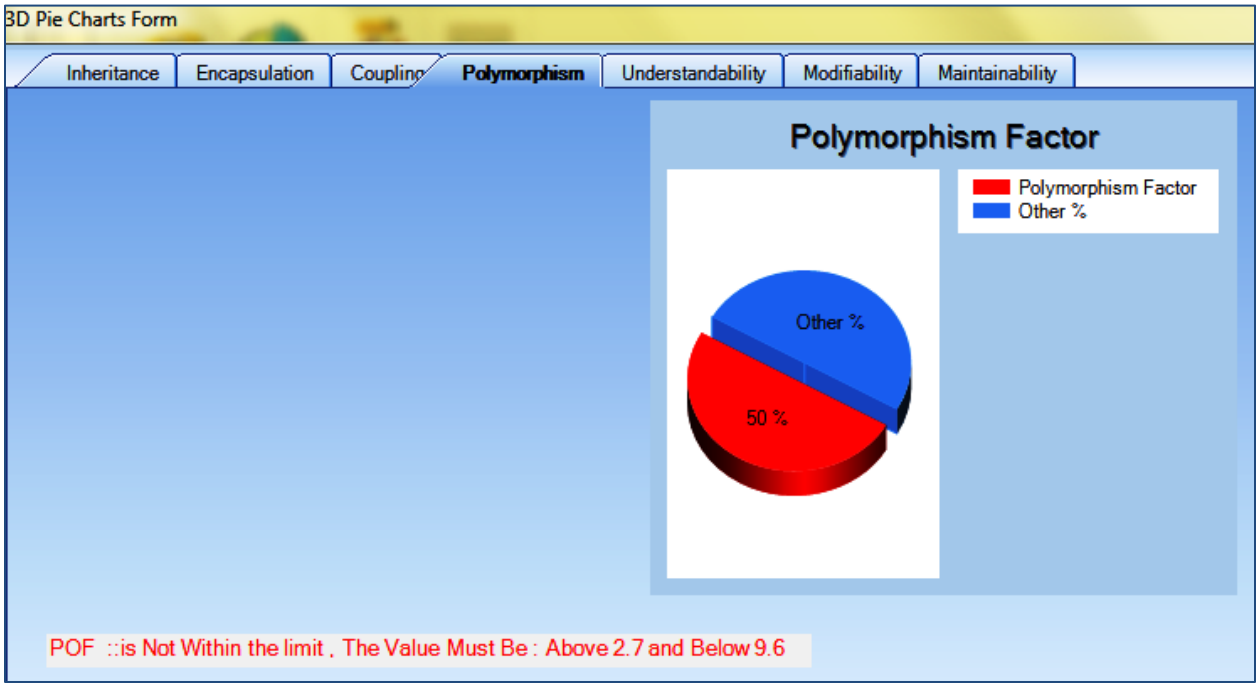


Figure (C-8) POF 3D-Pie Chart.

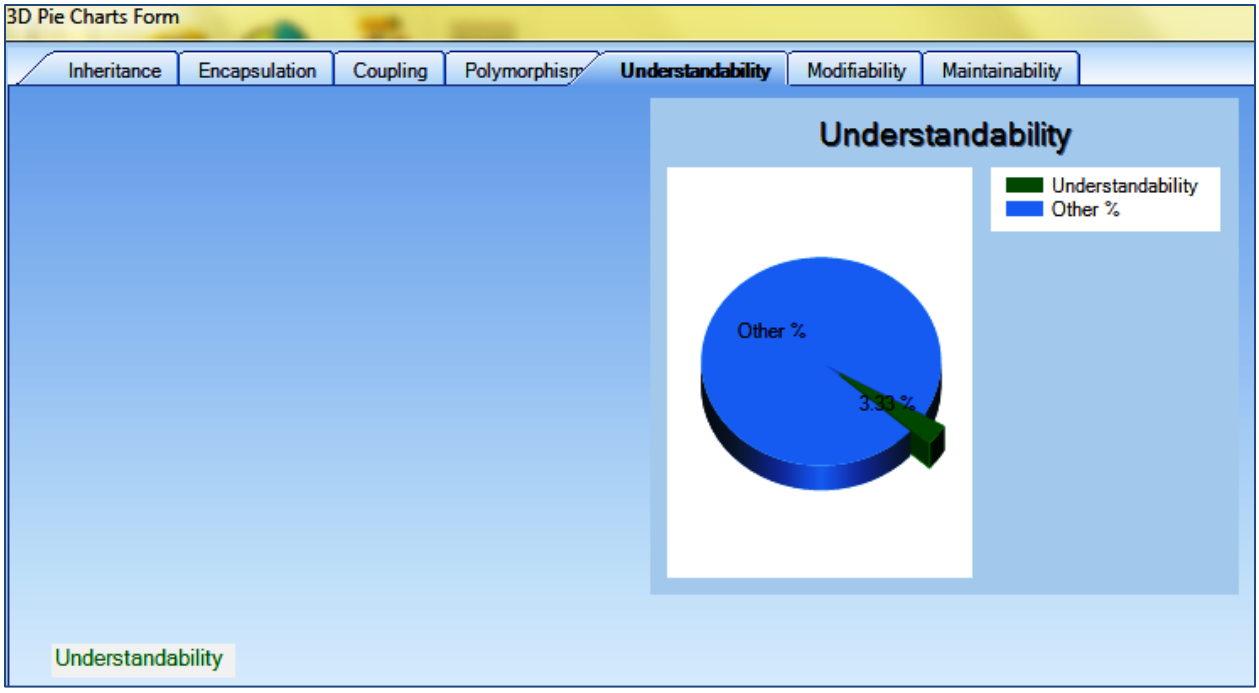


Figure (C-9) Understandability Quality Attribute 3D-Pie Chart.

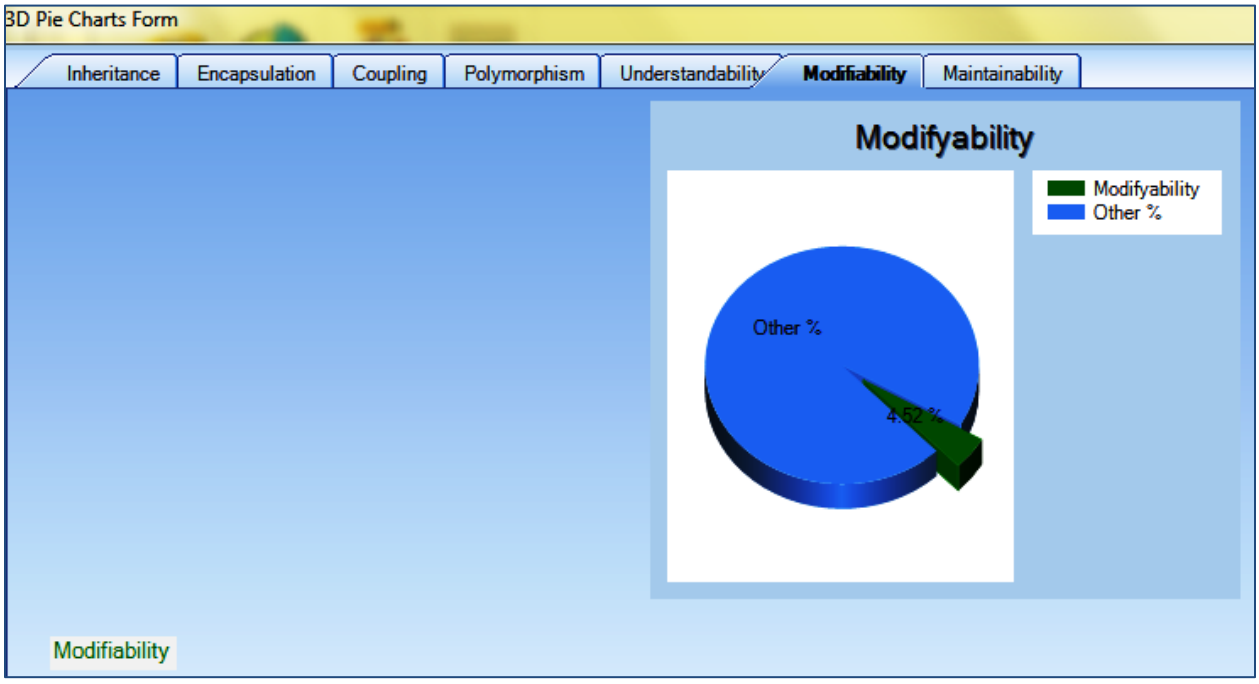


Figure (C-10) Modifiability Quality Attribute 3D-Pie Chart.

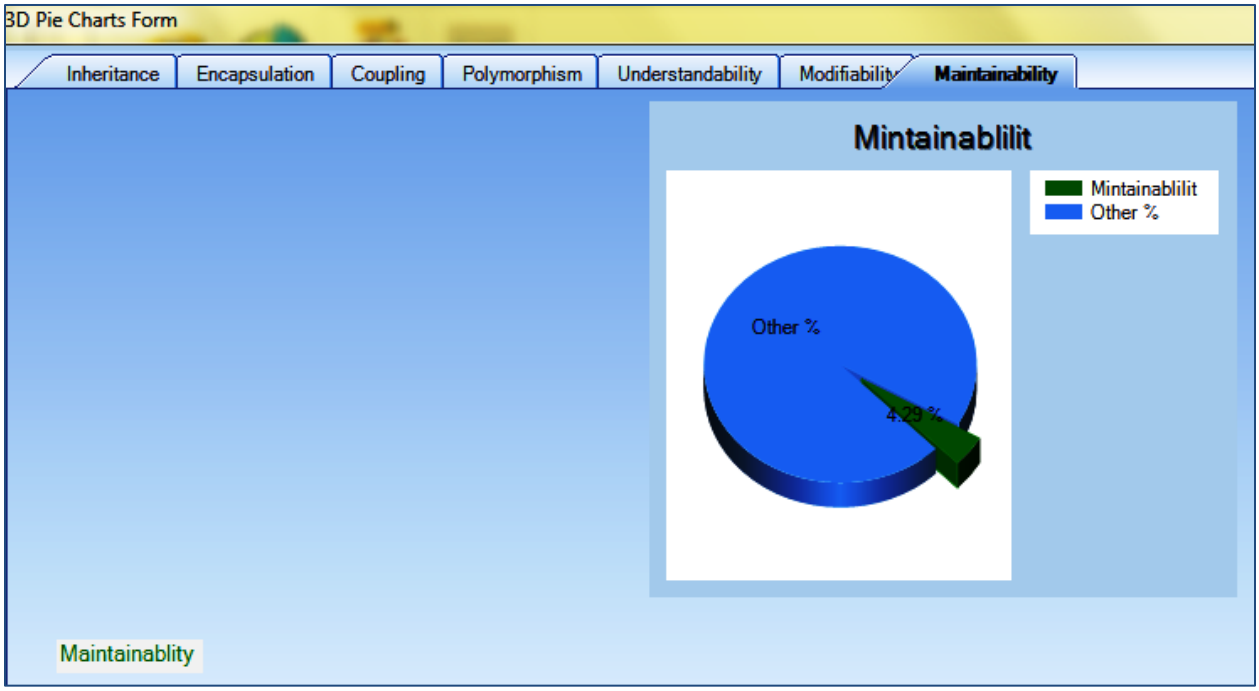


Figure (C-11) Maintainability Quality Attribute 3D-Pie Chart.

Naming conventions of the design can be seen in figure (C-12).

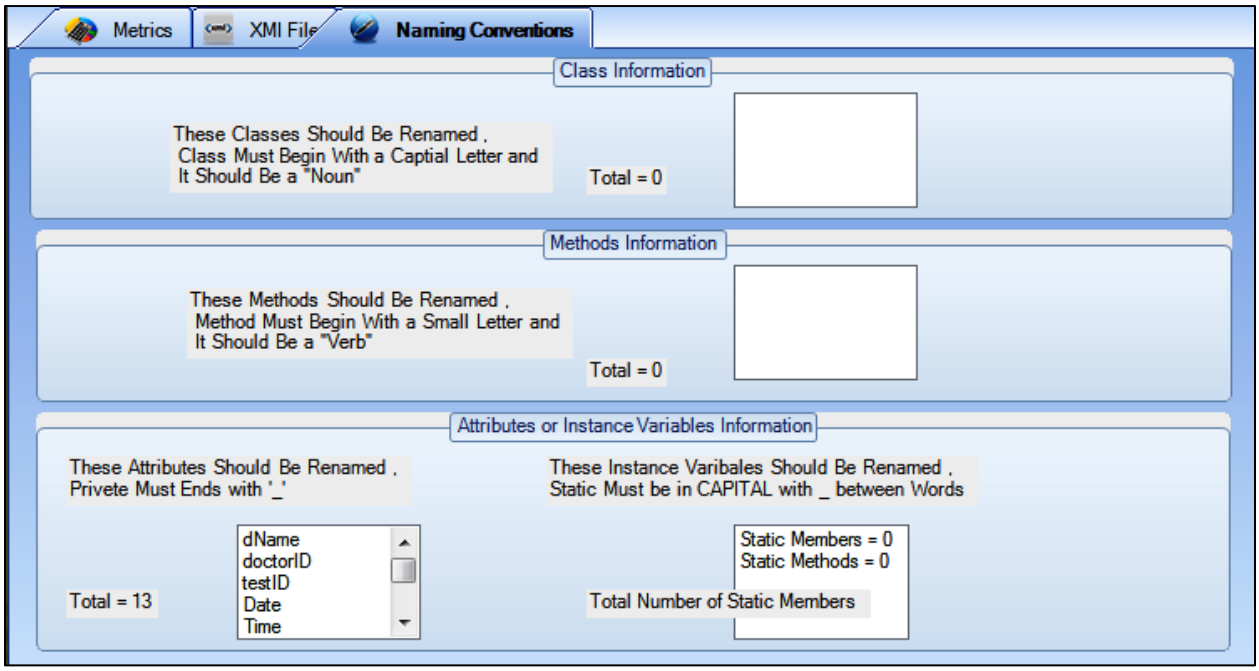


Figure (C-12) Naming Conventions Tab.

XML document generated by KDM tool can be seen in (C-13).

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<!-- CLass Diagram Metrics  -->
- <Metrics>
    <Metric Id="3" />
    <DesignerName>Conrad Mayson</DesignerName>
    <ModelName>Hospital Management Sys.</ModelName>
    <MHF>57.14</MHF>
    <AHF>81.25</AHF>
    <MIF>54.55</MIF>
    <AIF>66.67</AIF>
    <CF>7.78</CF>
    <POF>50</POF>
    <Understadability>3.33</Understadability>
    <Modifiability>4.52</Modifiability>
    <Maintainability>4.29</Maintainability>
  </Metrics>
```

Figure (C-13) XML Document Generated by KDM Tool

XML document is used as input to KRS tool, the output is crystal report, see figure (C-14).

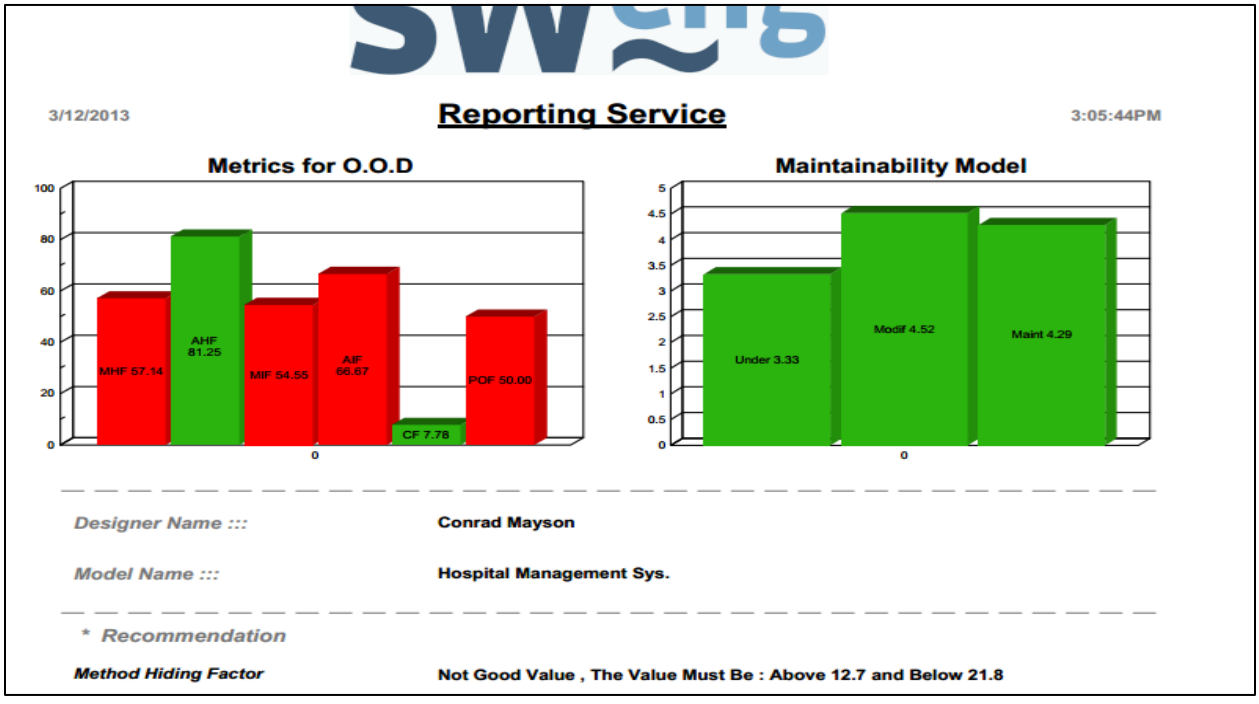


Figure (C-14) Crystal Report for HMS

XML document is used as input to KDB tool, and the metrics are stored in the database. See figure (C-15).

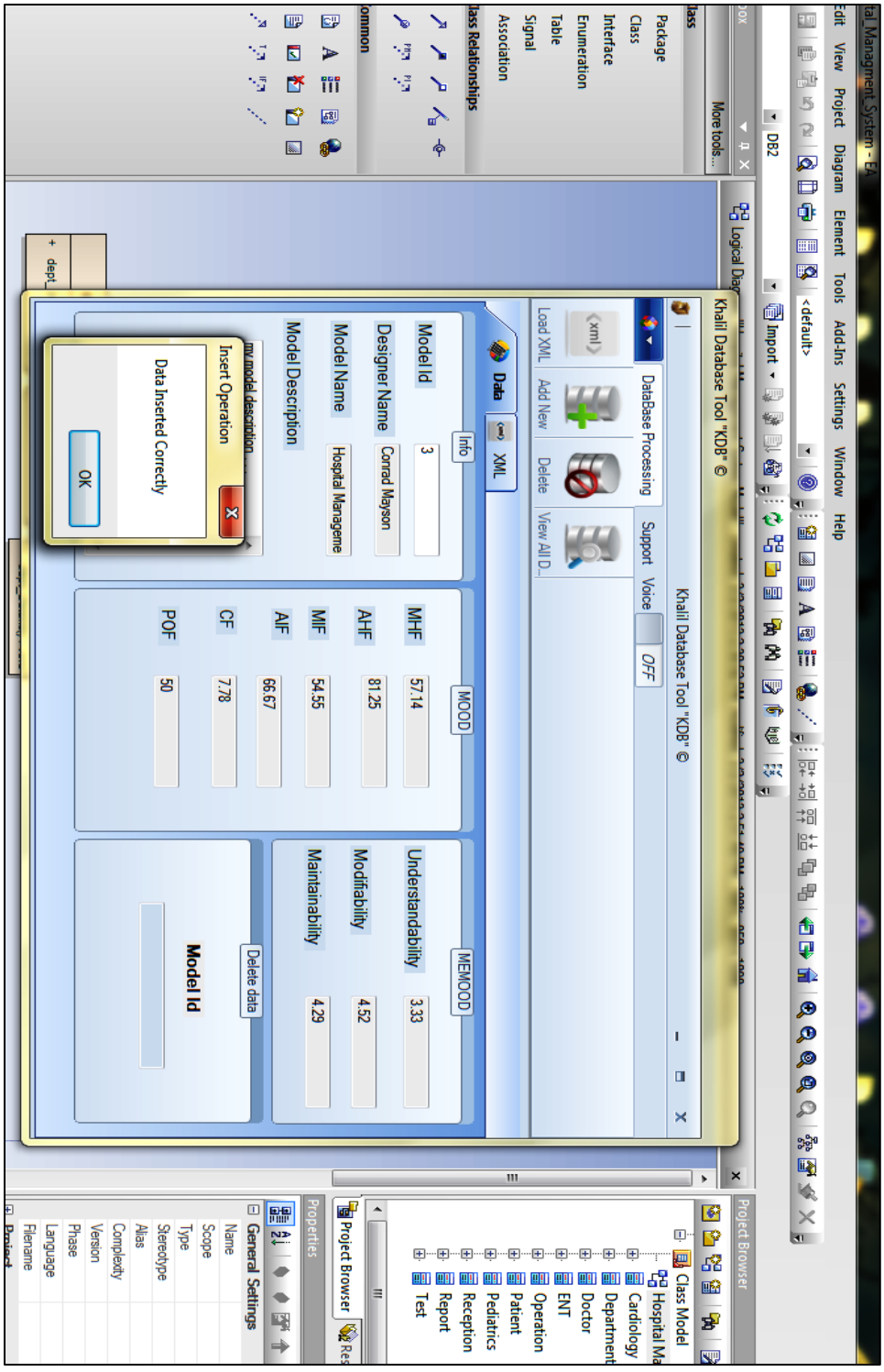


Figure (C-15) KDB tool

All metrics in the database can be seen by pressing of View All Data in KDB to ribbon bar. See figure (C-16).

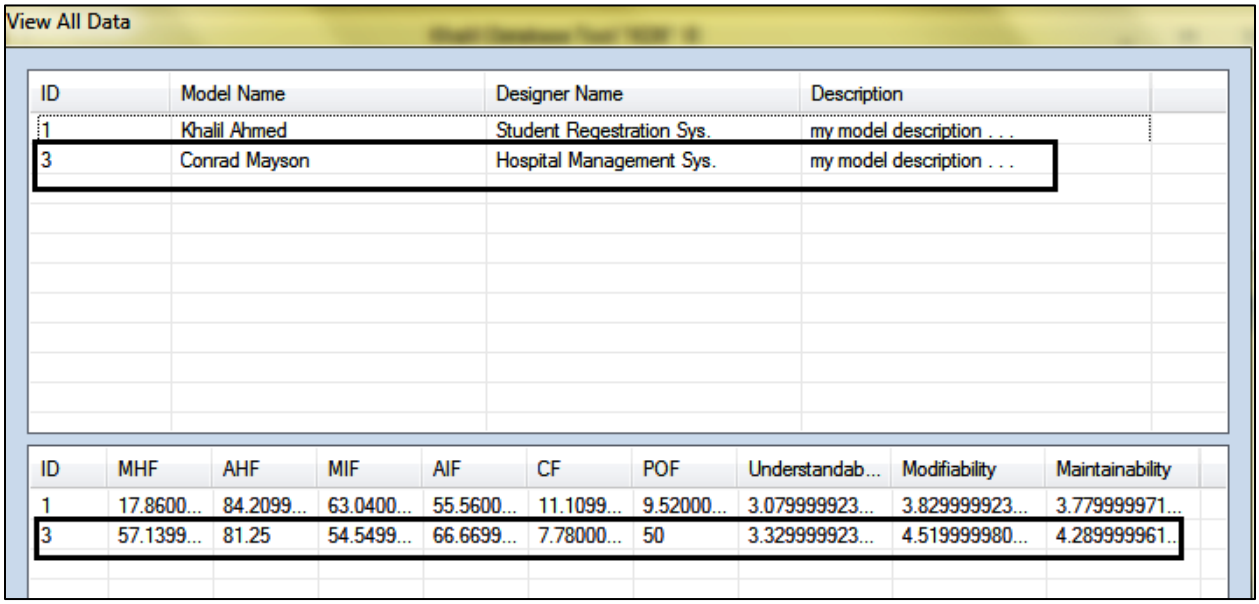


Figure (C-16) Metrics in the database

Hotel Management System (HOM)

Class diagram for HOM system is depicted in figure (C-17).

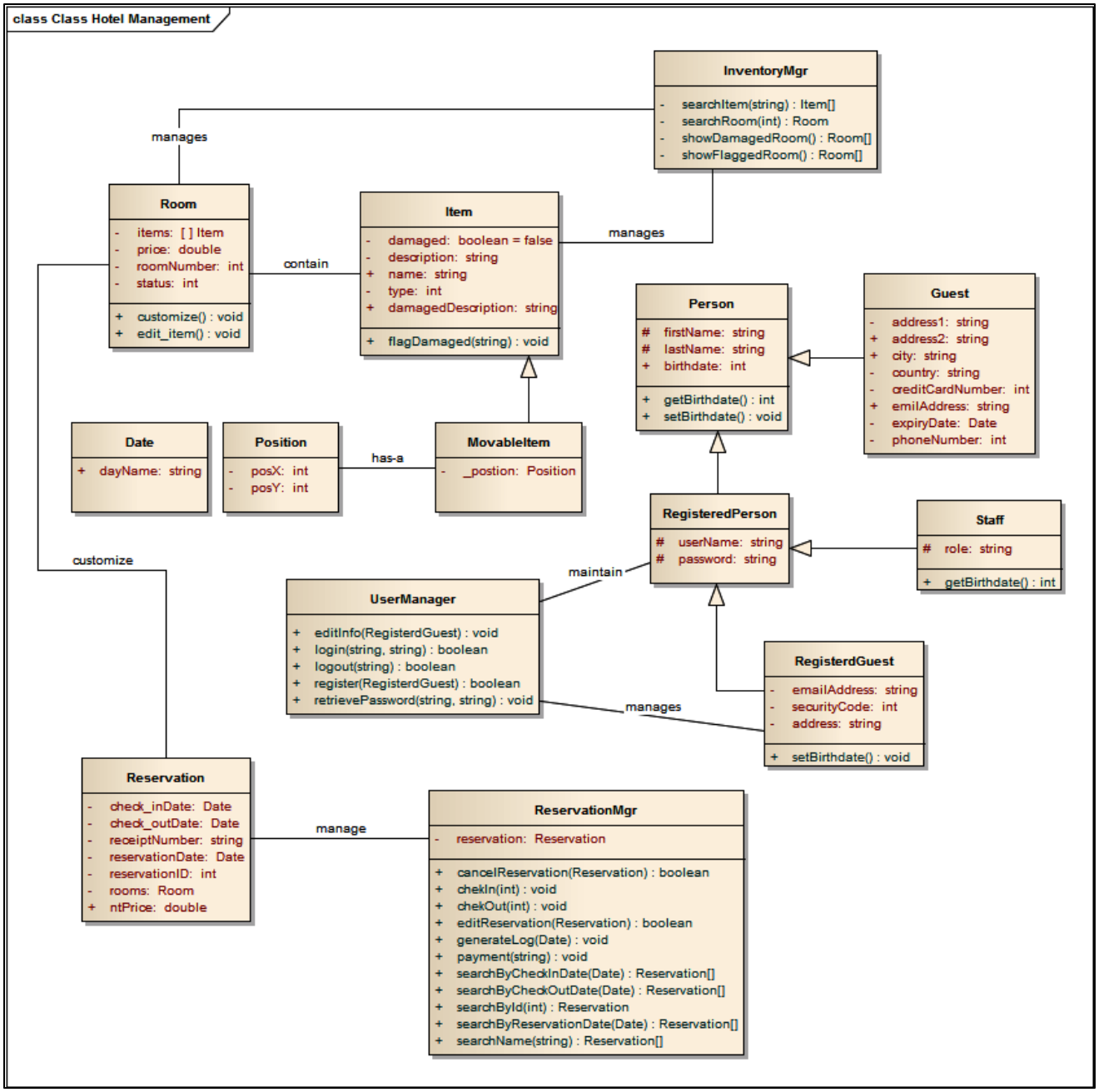

class Class Hotel Management
InventoryMgr
- searchItem(string) : Item[]
- searchRoom(int) : Room
- showDamagedRoom() : Room[]
- showFlaggedRoom() : Room[]
manages
Room
- items: [ ] Item
- price: double
- roomNumber: int
- status: int
+ customize() : void
+ edit_item() : void
contain
Item
- damaged: boolean = false
- description: string
+ name: string
- type: int
+ damagedDescription: string
+ flagDamaged(string) : void
manages
Person
firstName: string
lastName: string
+ birthdate: int
+ getBirthdate() : int
+ setBirthdate() : void
Guest
- address1: string
+ address2: string
+ city: string
- country: string
- creditCardNumber: int
+ emilAddress: string
- expiryDate: Date
- phoneNumber: int
Date
+ dayName: string
Position
- posX: int
- posY: int
has-a
MovableItem
- _postion: Position
RegisteredPerson
userName: string
password: string
Staff
role: string
+ getBirthdate() : int
customize
maintain
UserManager
+ editInfo(RegisterdGuest) : void
+ login(string, string) : boolean
+ logout(string) : boolean
+ register(RegisterdGuest) : boolean
+ retrievePassword(string, string) : void
RegisterdGuest
- emailAddress: string
- securityCode: int
- address: string
+ setBirthdate() : void
manages
Reservation
- check_inDate: Date
- check_outDate: Date
- receiptNumber: string
- reservationDate: Date
- reservationID: int
- rooms: Room
+ ntPrice: double
manage
ReservationMgr
- reservation: Reservation
+ cancelReservation(Reservation) : boolean
+ chekIn(int) : void
+ chekOut(int) : void
+ editReservation(Reservation) : boolean
+ generateLog(Date) : void
+ payment(string) : void
+ searchByCheckInDate(Date) : Reservation[]
+ searchByCheckOutDate(Date) : Reservation[]
+ searchById(int) : Reservation
+ searchByReservationDate(Date) : Reservation[]
+ searchName(string) : Reservation[]

Figure (C-17).

Using XMI document that is generated by EA for HOM system class diagram as input to KDM tool, KDM tool will calculate metrics see figure (C-18).

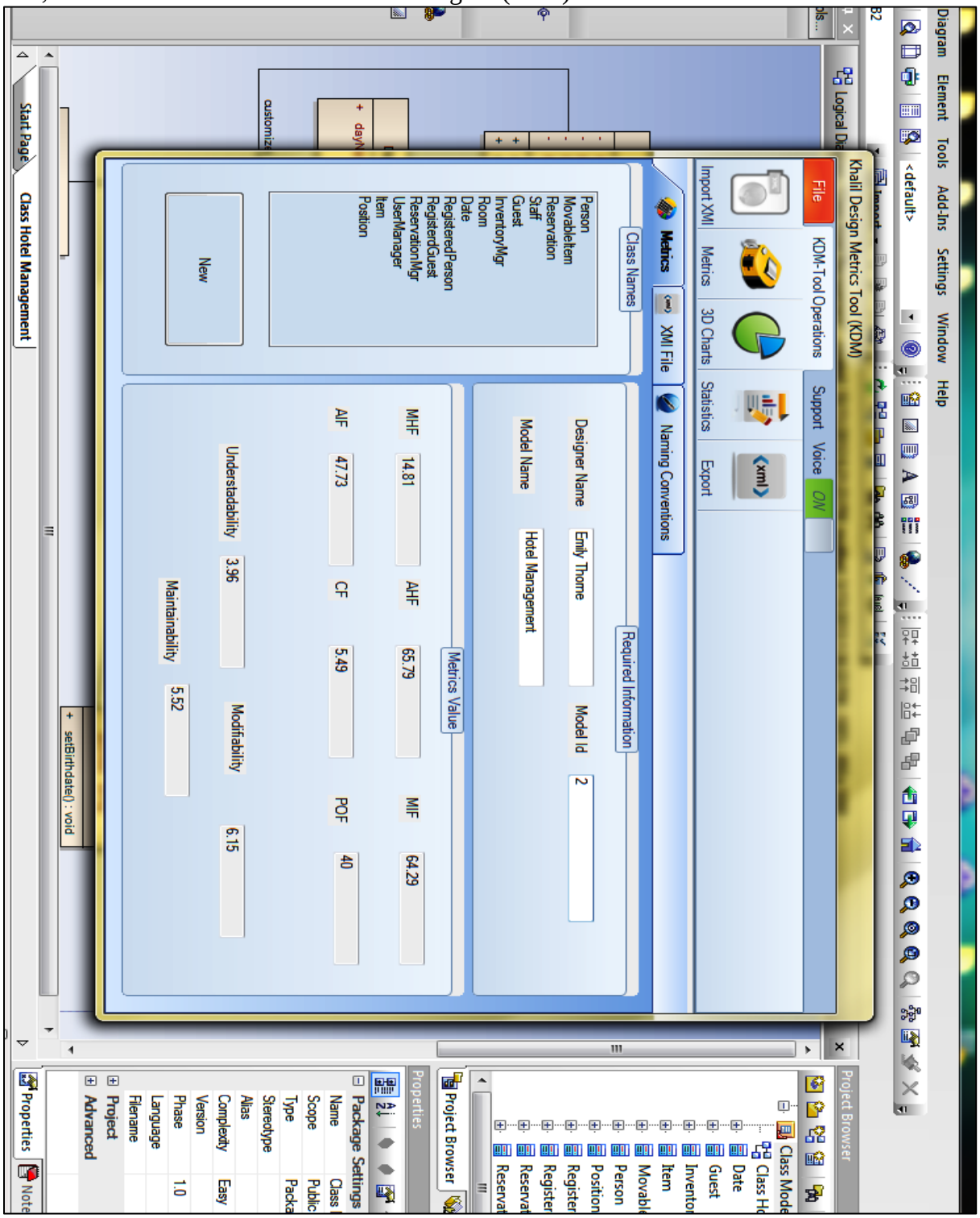


Figure (C-18) Metrics for HOM System.

XMI file is loaded in the XMI tab. See figure (C-19).

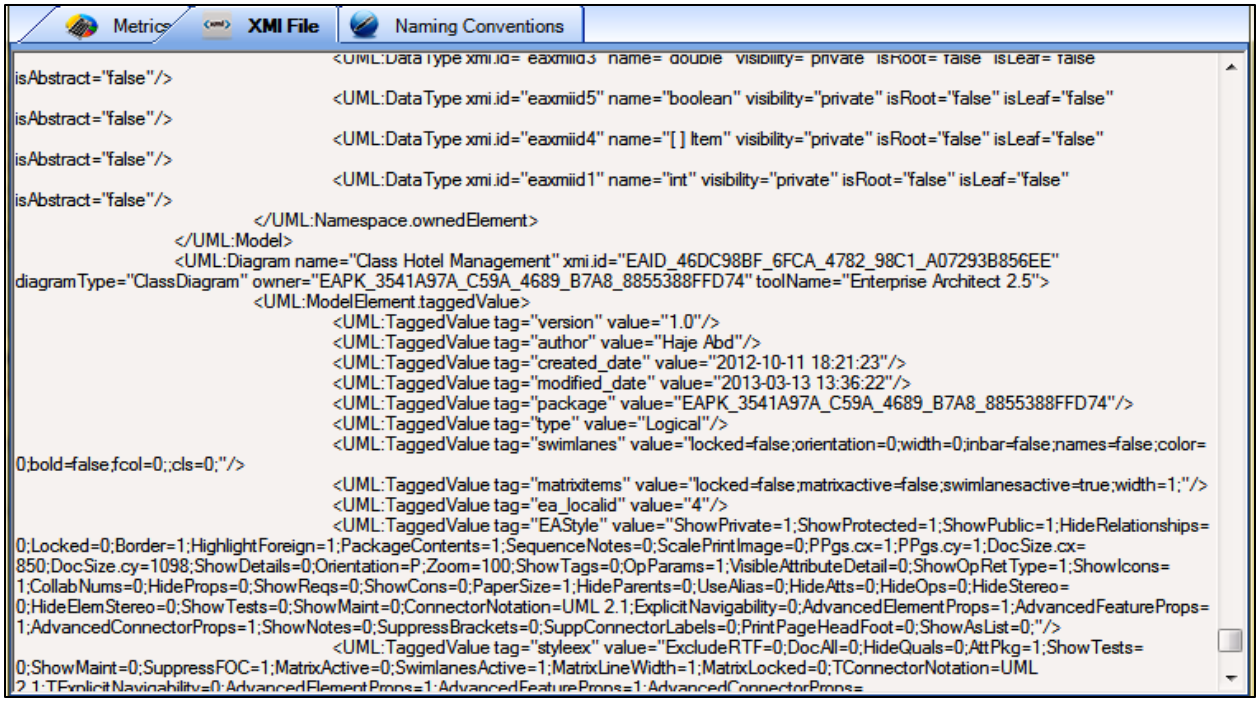


Figure (C-19) XMI Document for HOM System.

Design statistics can be seen in figure (C-20).

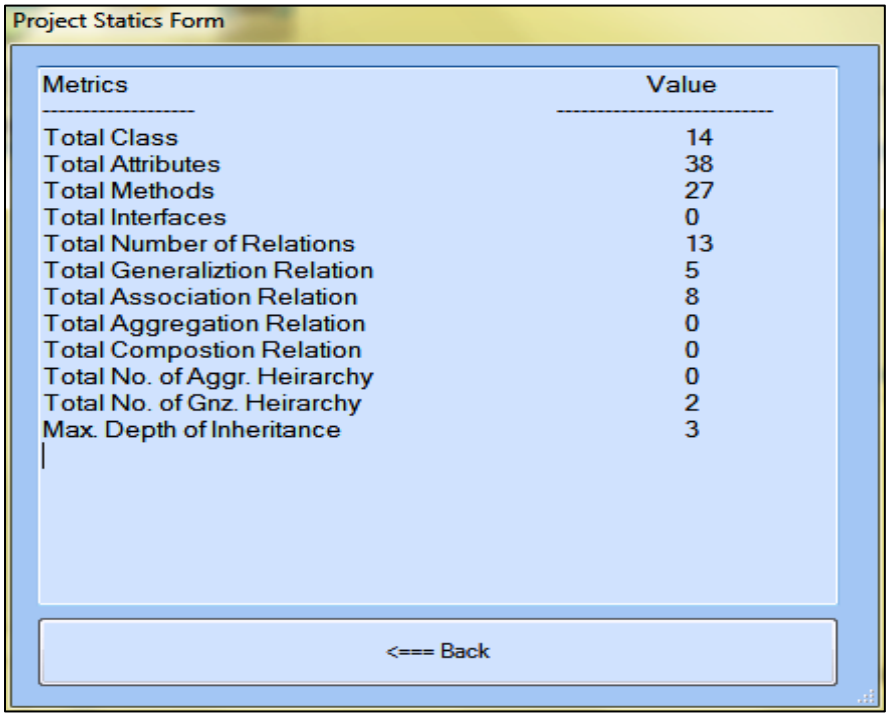


Figure (C-20) Design Statistics for HOM System.

Metrics are visualized in figures (C-21), (C-22), (C-23), (C-24), (C-25), (C-26), and (C-27).

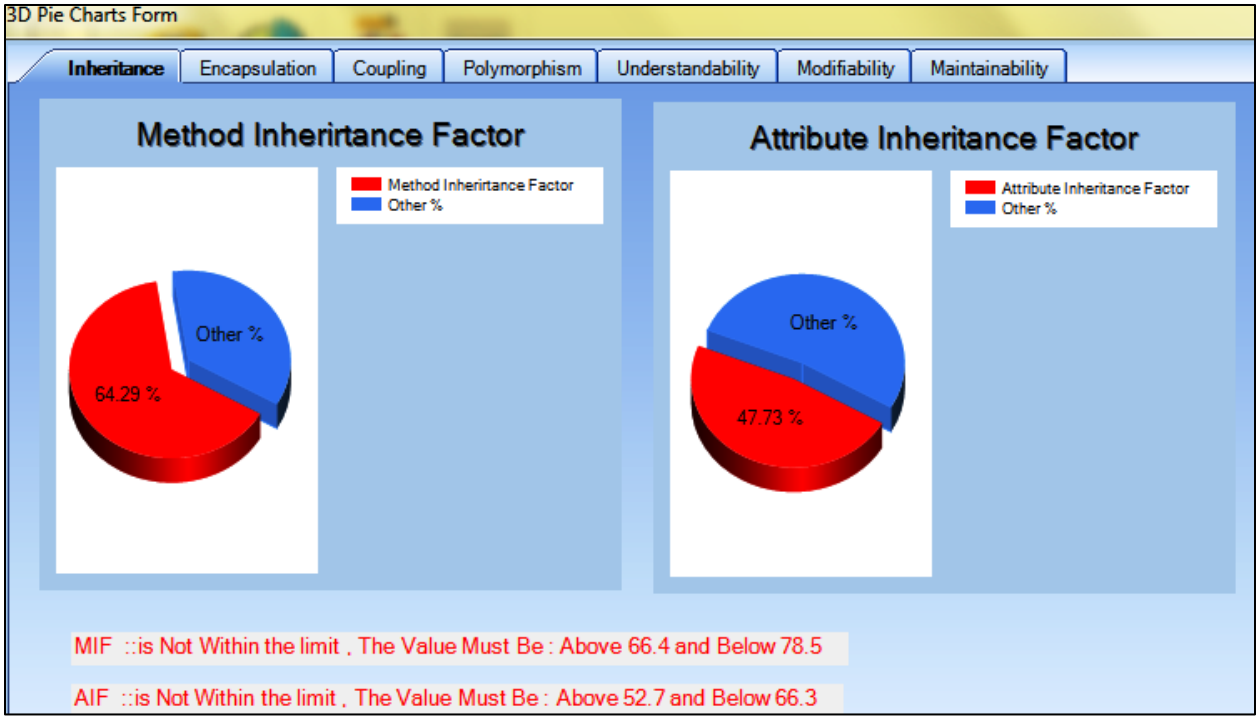


Figure (C-21) MIF and AIF 3D-Pie Chart.

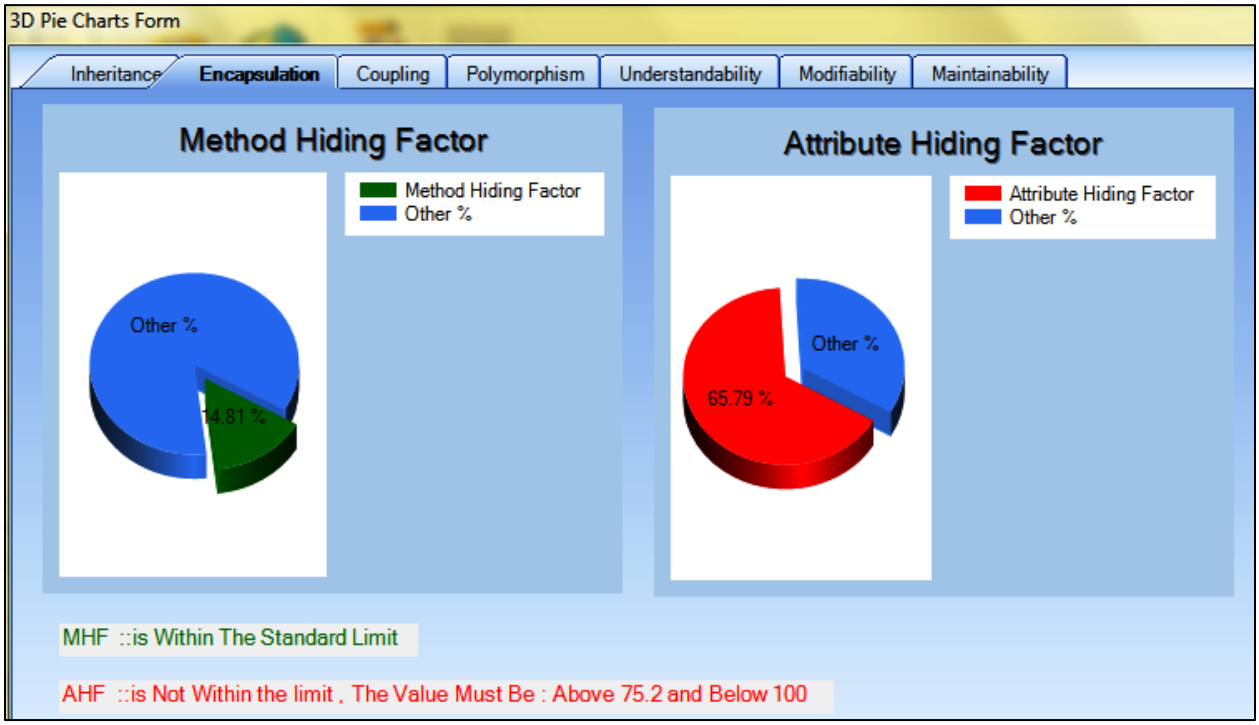


Figure (C-22) MHF and AHF 3D-Pie Chart.

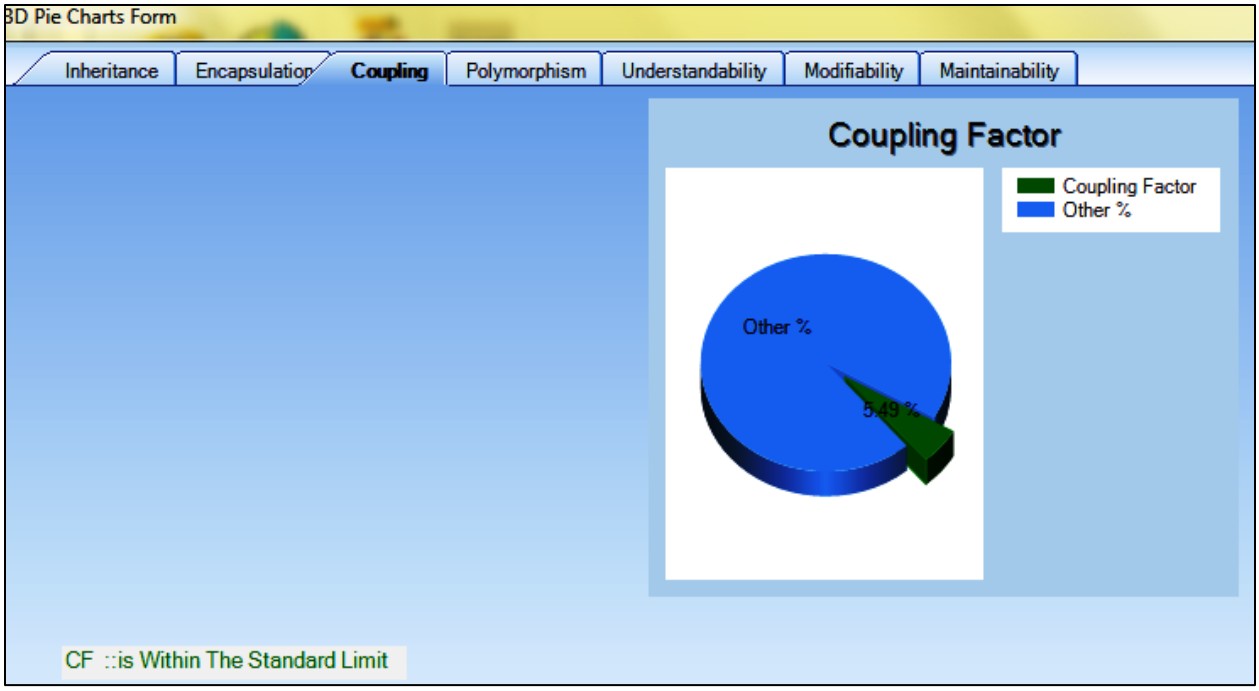


Figure (C-23) CF 3D-Pie Chart.

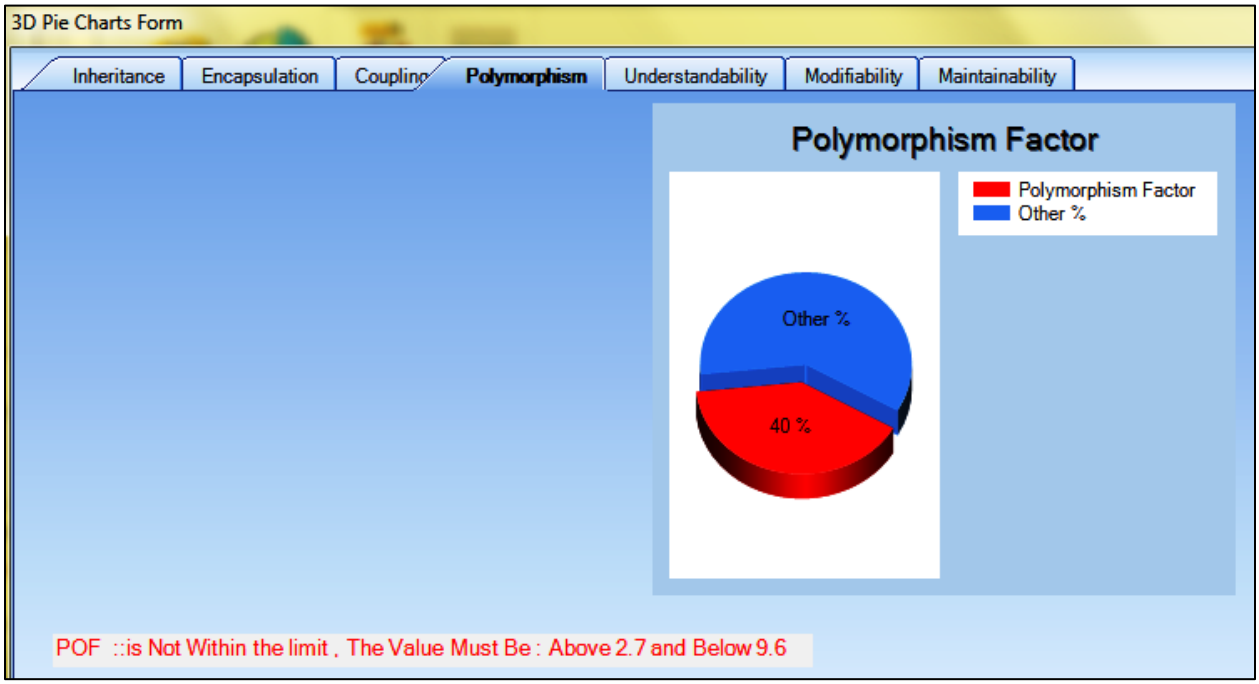


Figure (C-24) POF 3D-Pie Chart.

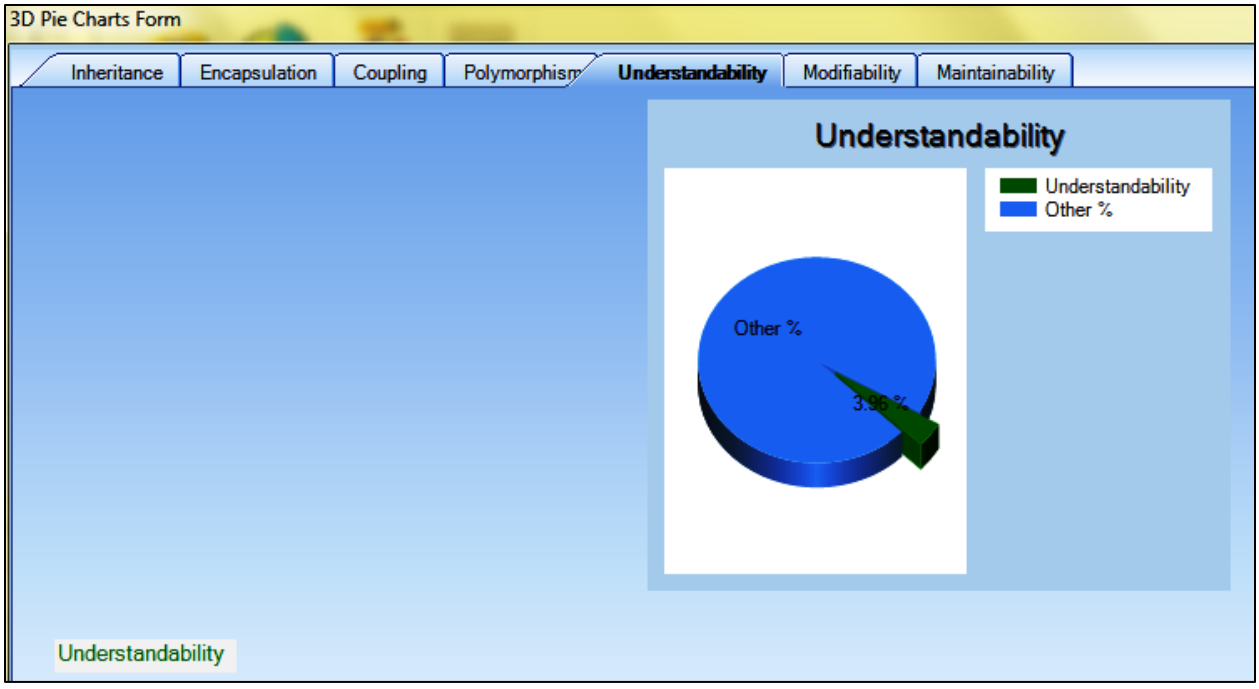


Figure (C-25) Understandability Quality Attribute 3D-Pie Chart.

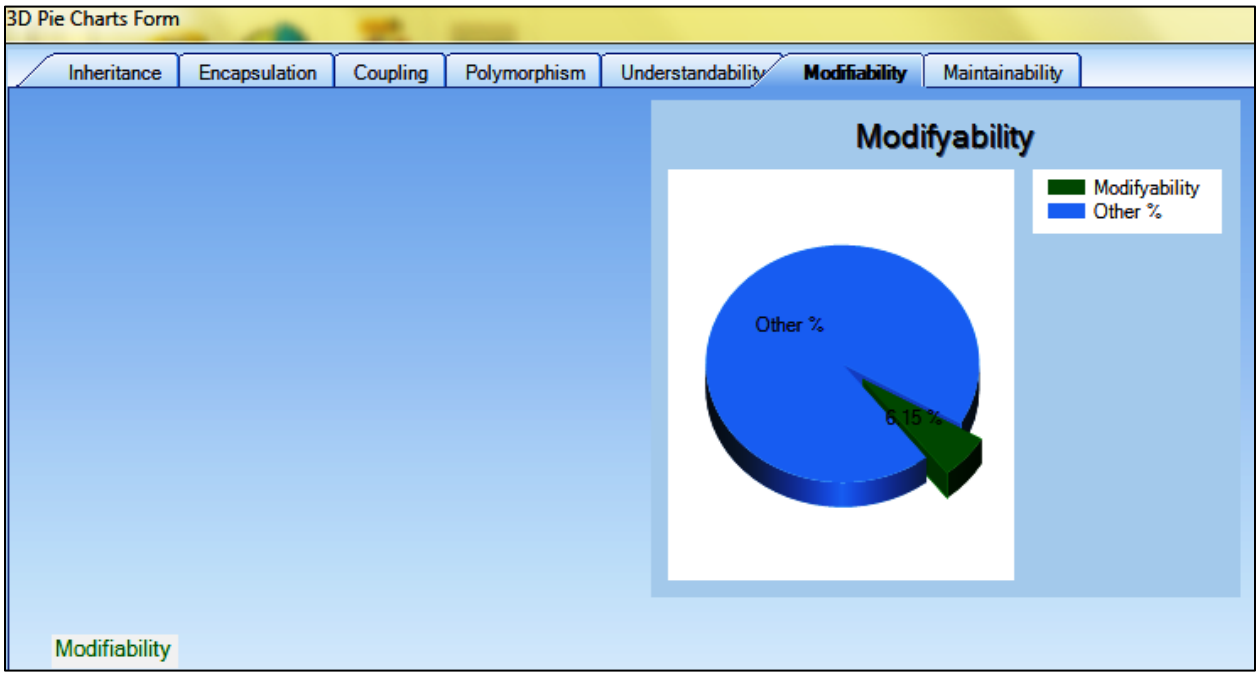


Figure (C-26) Modifiability Quality Attribute 3D-Pie Chart.

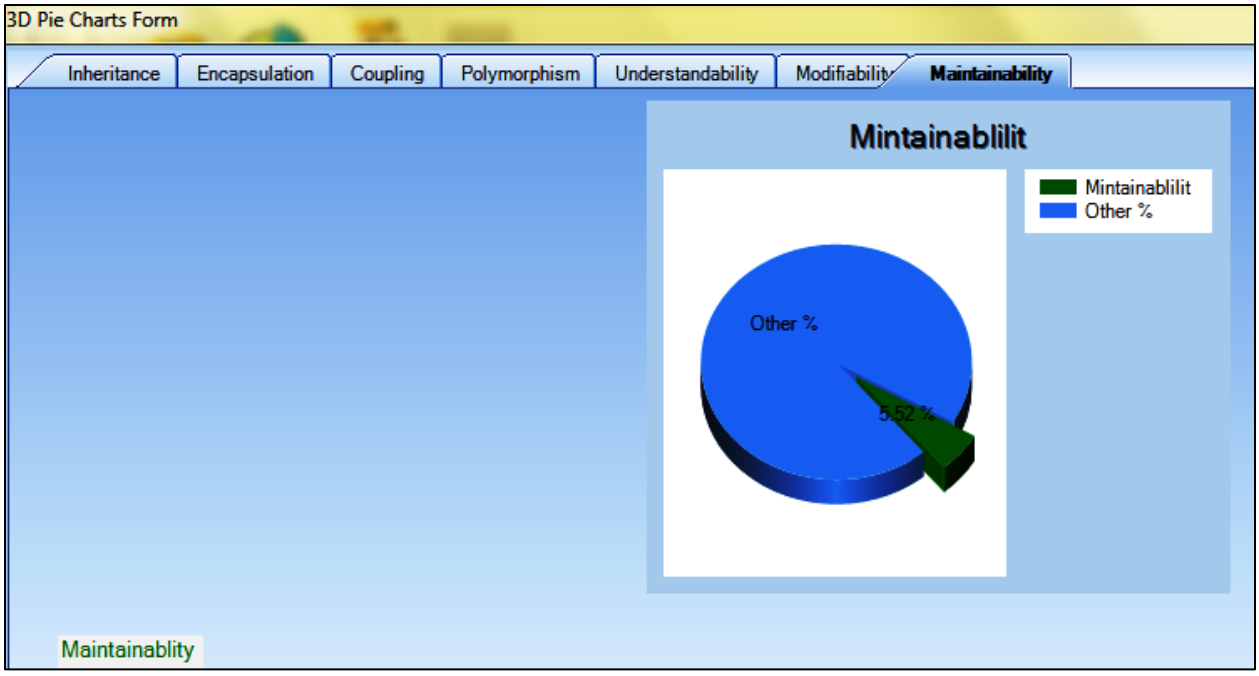


Figure (C-27) Maintainability Quality Attribute 3D-Pie Chart.

Naming convention for HOM system can be seen in figure (C-28).

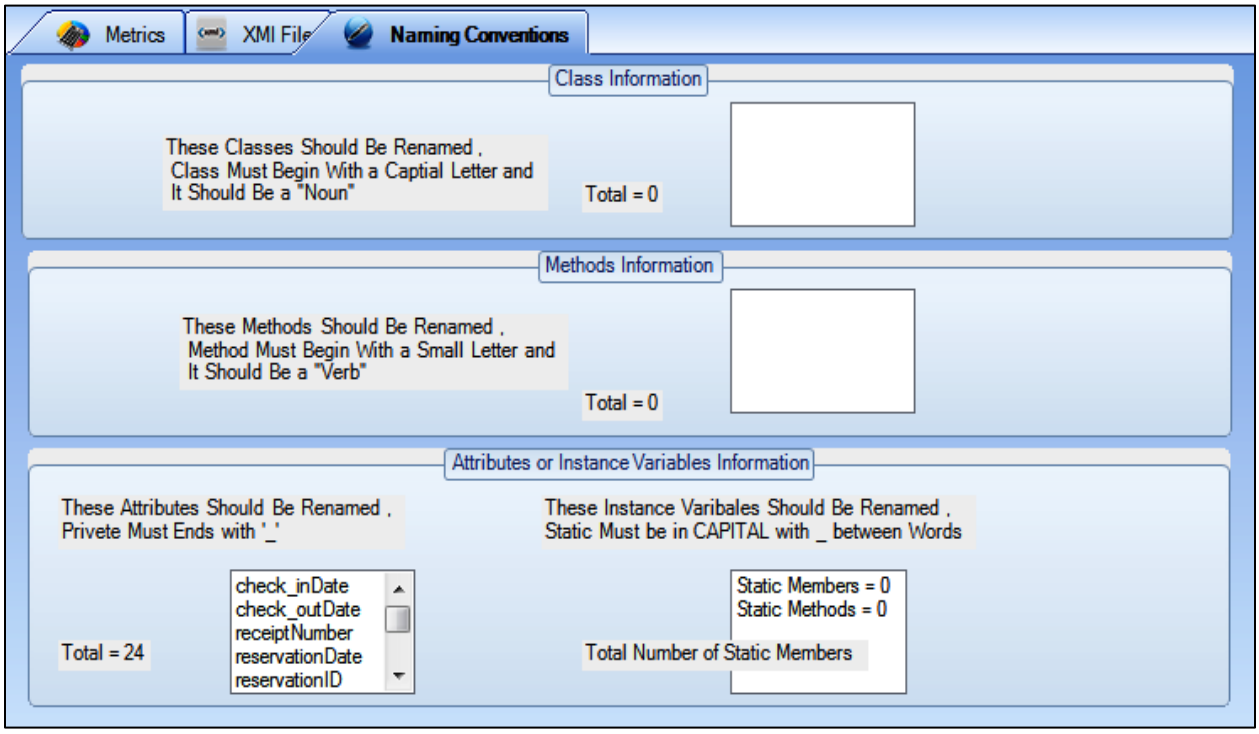


Figure (C-28) Design Naming Conventions for HOM System.

XML document generated by KDM tool can be seen in (C-29).

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<!-- CLass Diagram Metrics  -->
- <Metrics>
    <Metric Id="2" />
    <DesignerName>Emily Thorne</DesignerName>
    <ModelName>Hotel Management</ModelName>
    <MHF>14.81</MHF>
    <AHF>65.79</AHF>
    <MIF>64.29</MIF>
    <AIF>47.73</AIF>
    <CF>5.49</CF>
    <POF>40</POF>
    <Understadability>3.96</Understadability>
    <Modifiability>6.15</Modifiability>
    <Maintainability>5.52</Maintainability>
  </Metrics>
```

Figure (C-29) XML Document Generated By KDM Tool.

XML document is used as input to KRS tool, the output is crystal report, see figure (C-30).

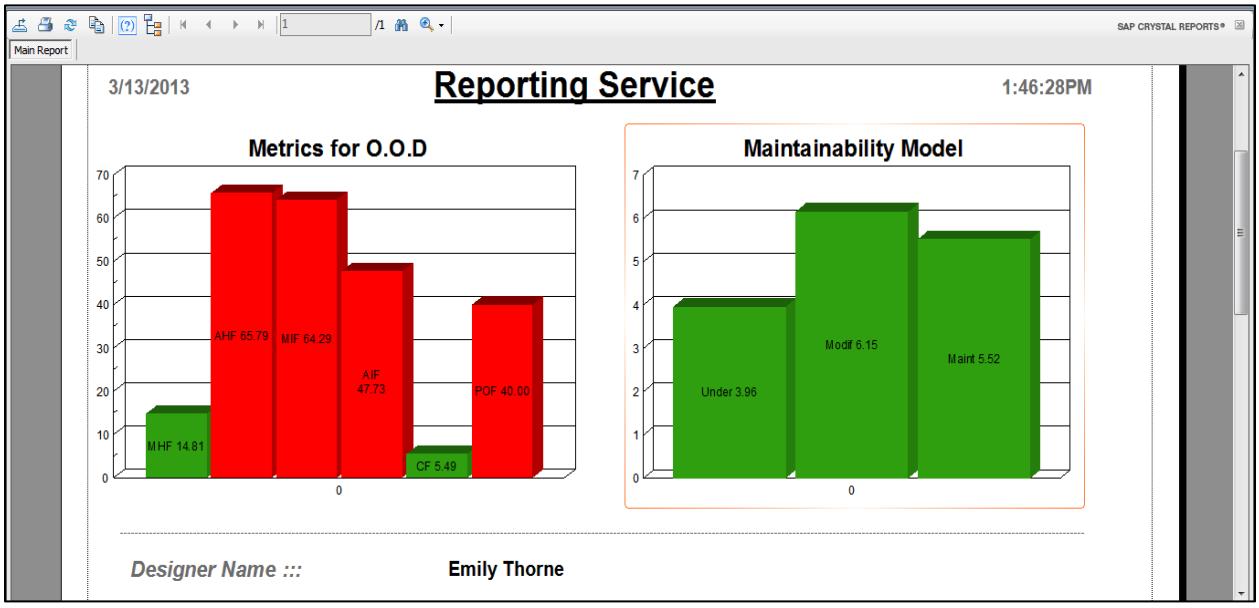


Figure (C-30) Crystal Report Generated By KRS Tool.

XML document is used as input to KDB tool, and the metrics are stored in the database. See figure (C-31).

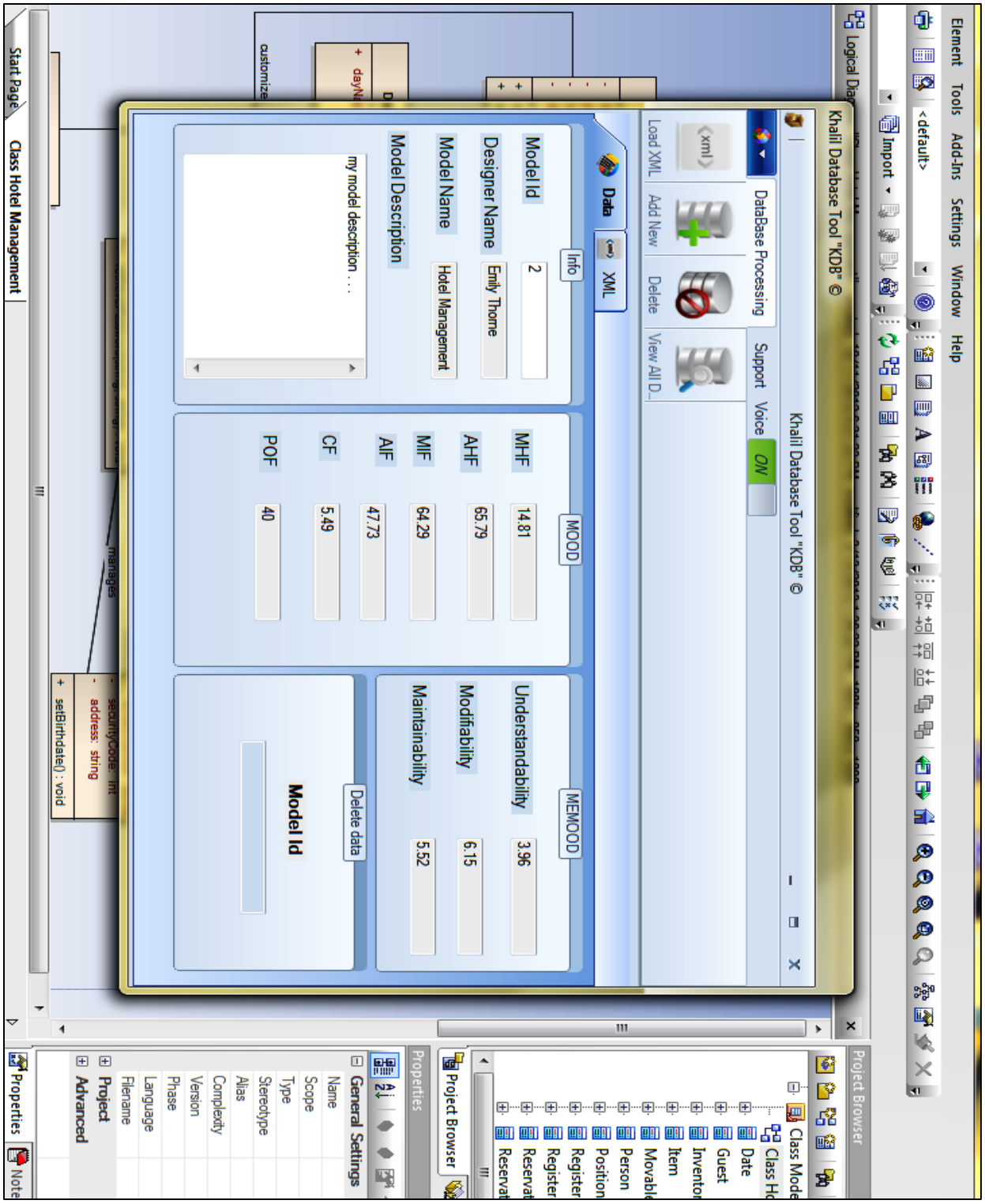


Figure (C-31) Metrics of HOM System Stored in Database.

All metrics in the database can be seen by pressing of View All Data in KDB to ribbon bar. See figure (C-32).

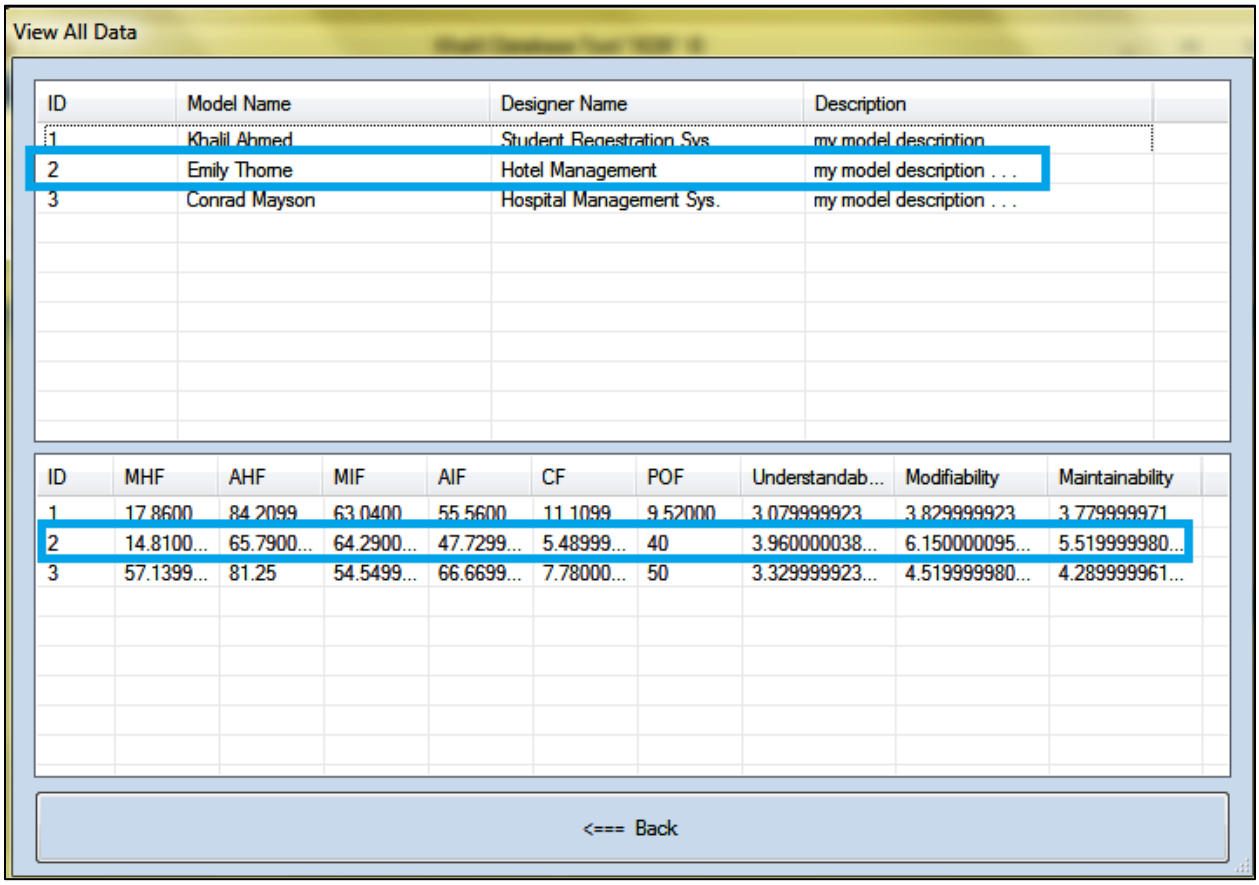


Figure (C-32) Metrics in database.

Sequence Diagrams for MOOD and MEMOOD Algorithms

In this appendix, a sequence diagram is represented for each algorithm in chapter 4. Figure (D-1), (D-2), (D-3), (D-4), (D-5), (D-6), (D-7) are for MOOD and Figures (D-8), (D-9), (D-10) are for MEMOOD.

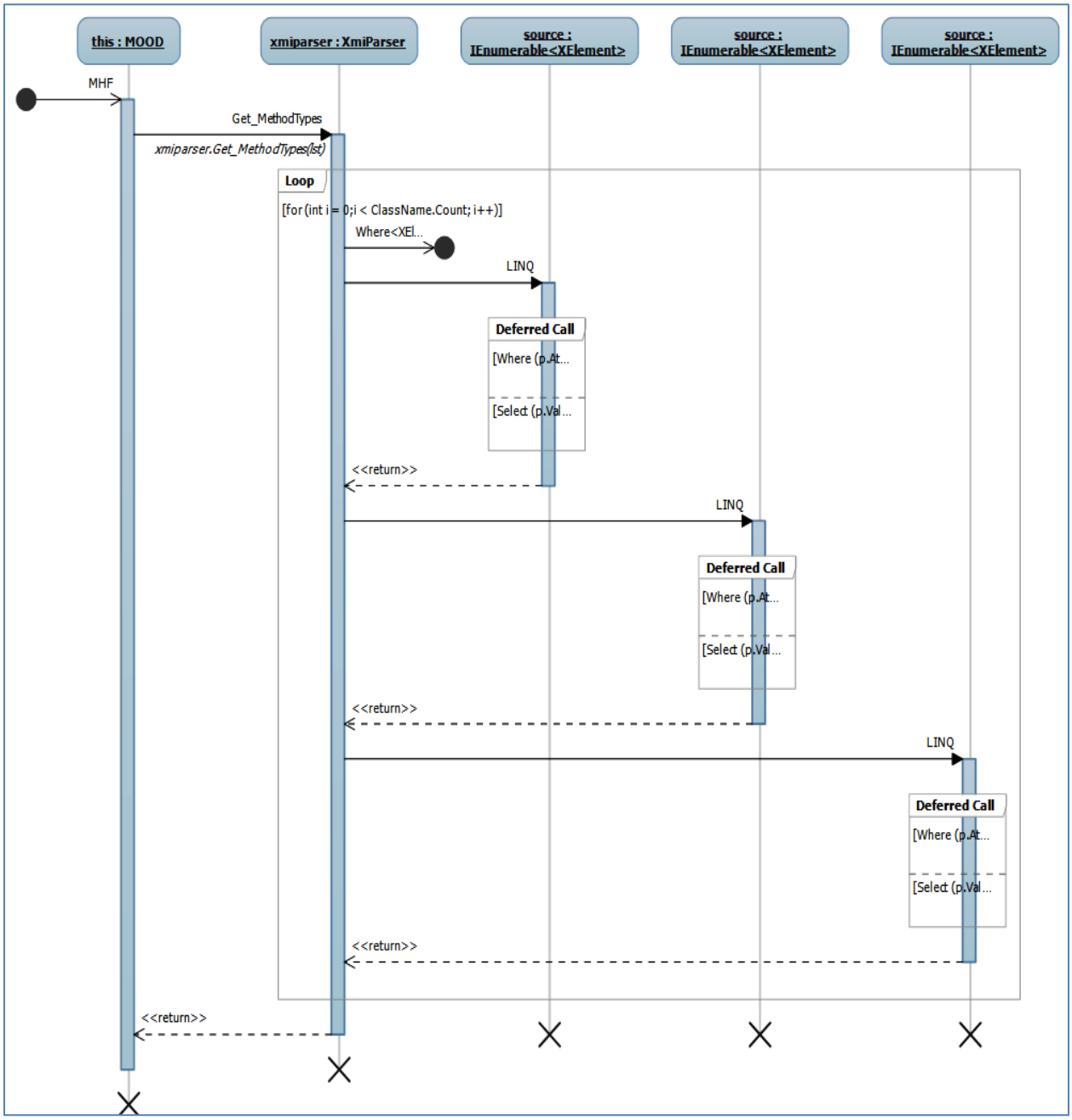


Figure (D-1) Sequence Diagram for MHF Algorithm.

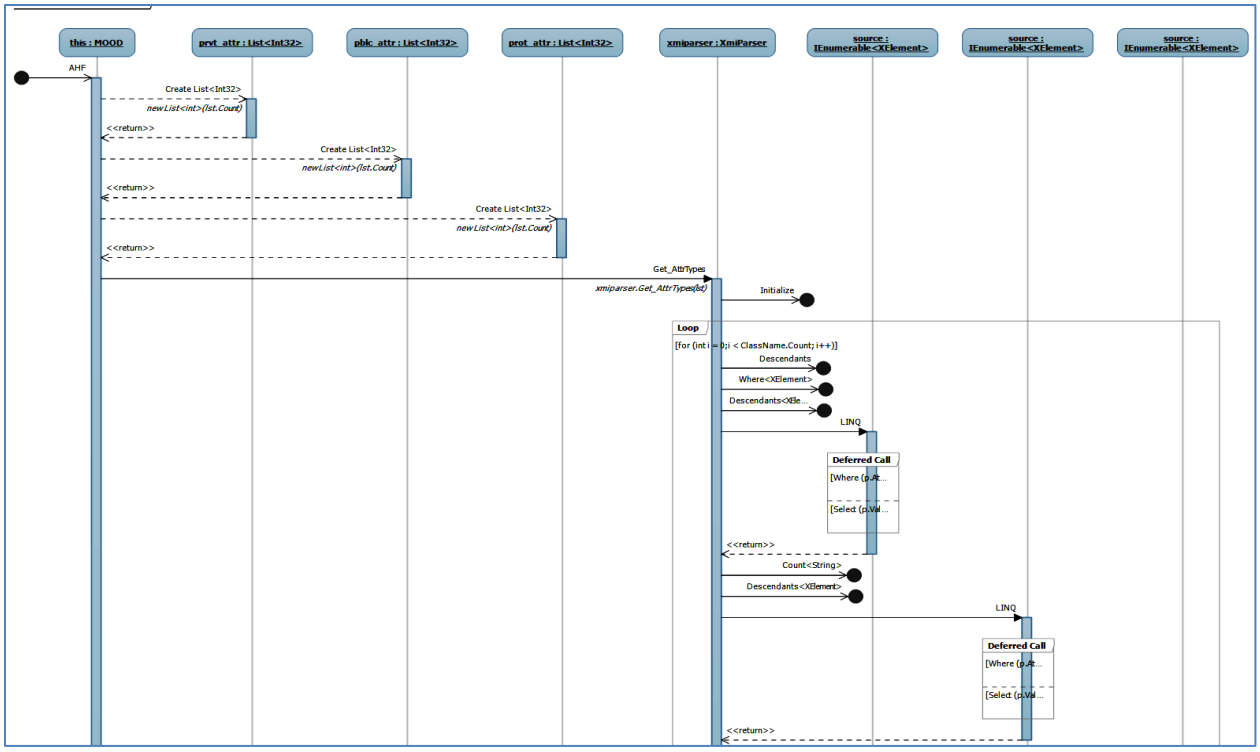


Figure (D-2) Sequence Diagram for AHF Algorithm (a).

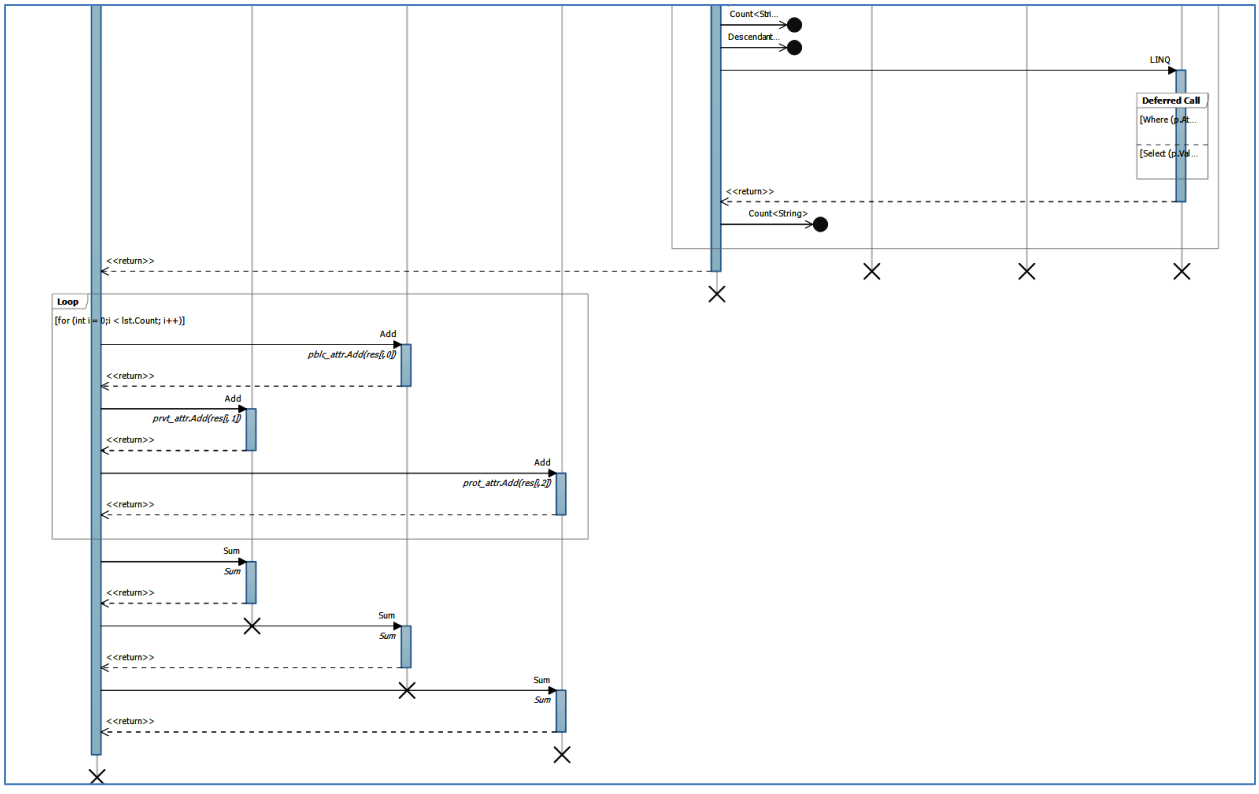


Figure (D-2) Sequence Diagram for AHF Algorithm (b).

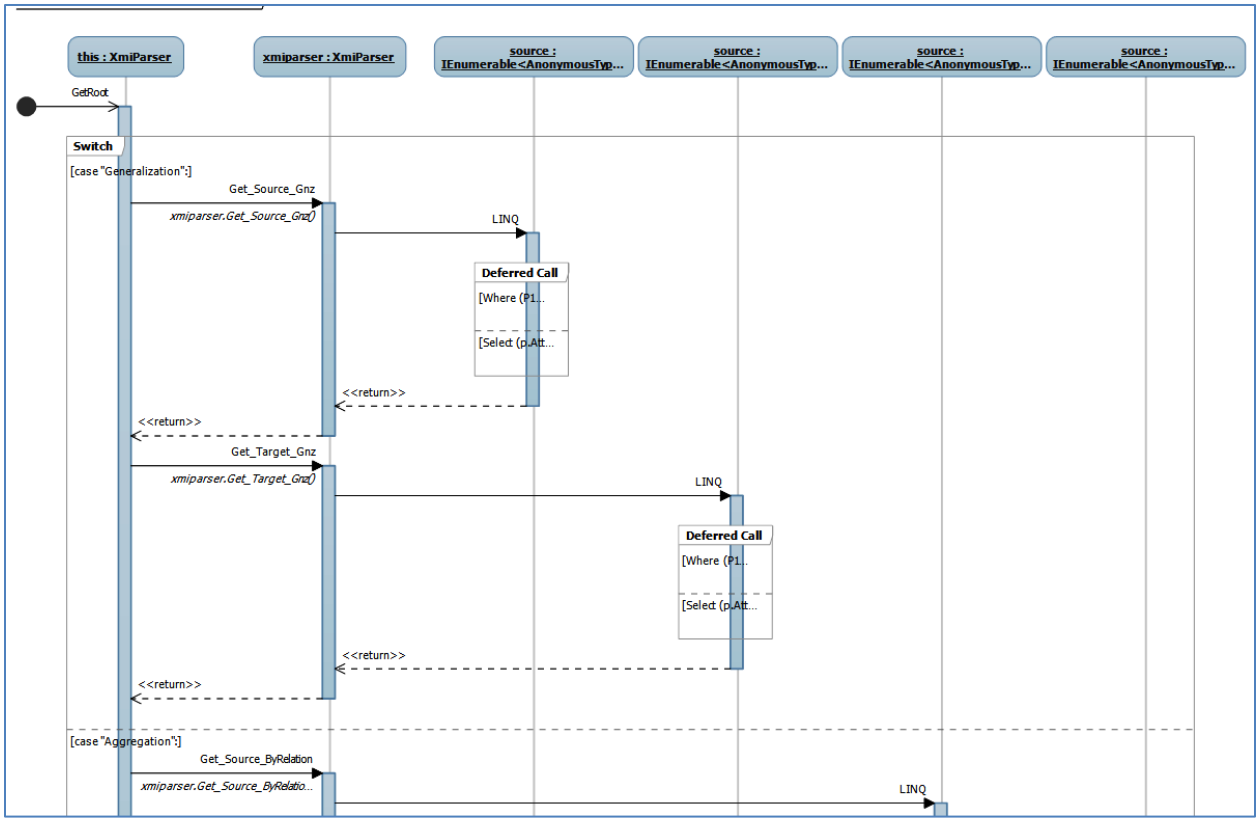


Figure (D-3) Root Algorithm (a).

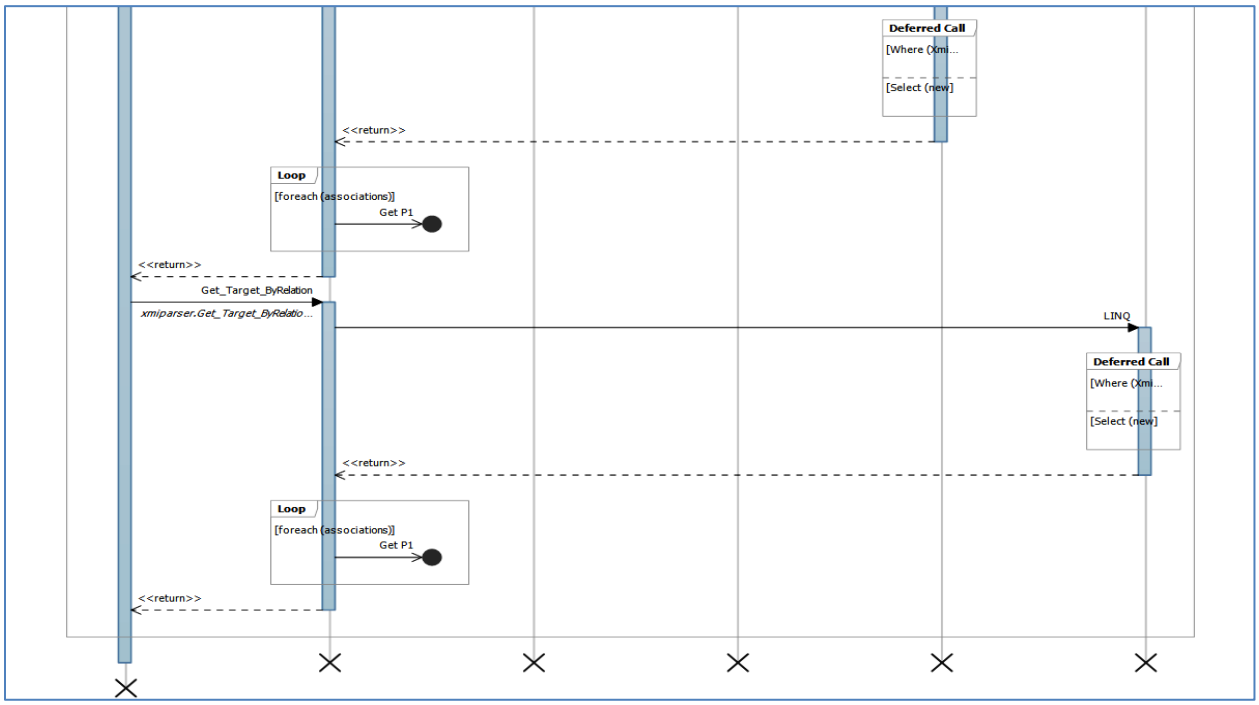


Figure (D-3) Root Algorithm (b).

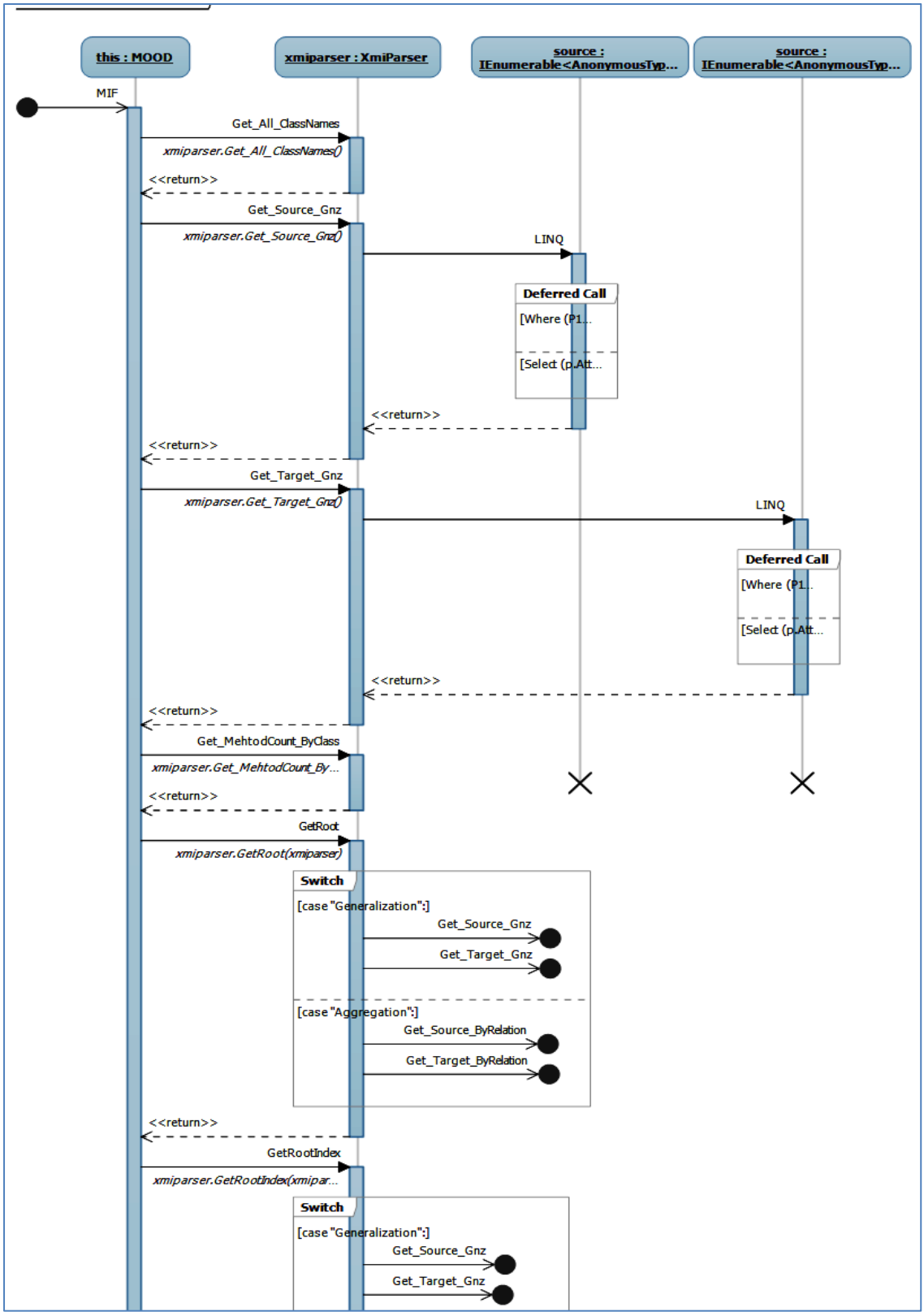


Figure (D-4) Sequence Diagram for MIF Algorithm (a)

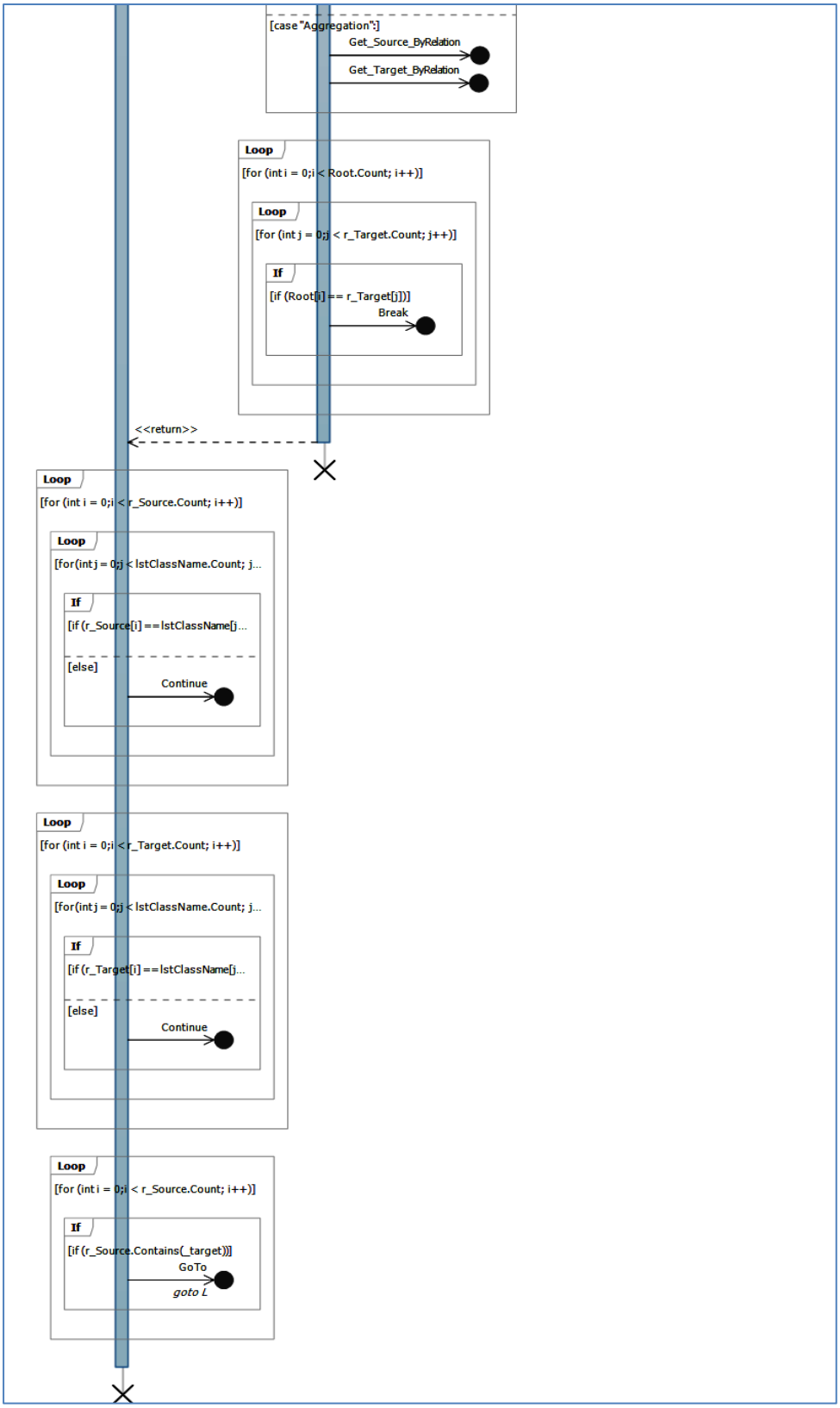


Figure (D-4) Sequence Diagram for MIF Algorithm (b)

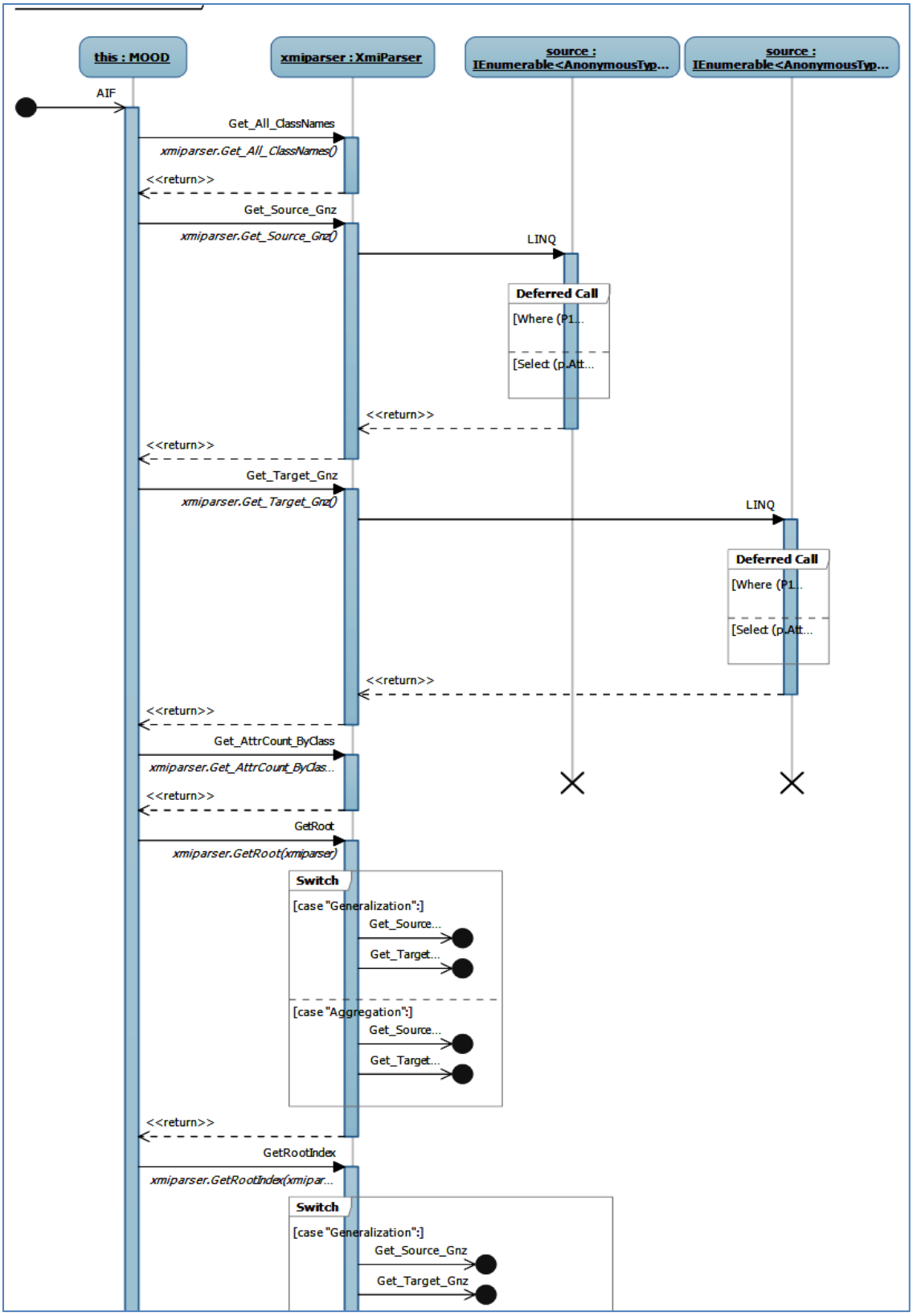


Figure (D-5) Sequence Diagram for AIF Algorithm (a)

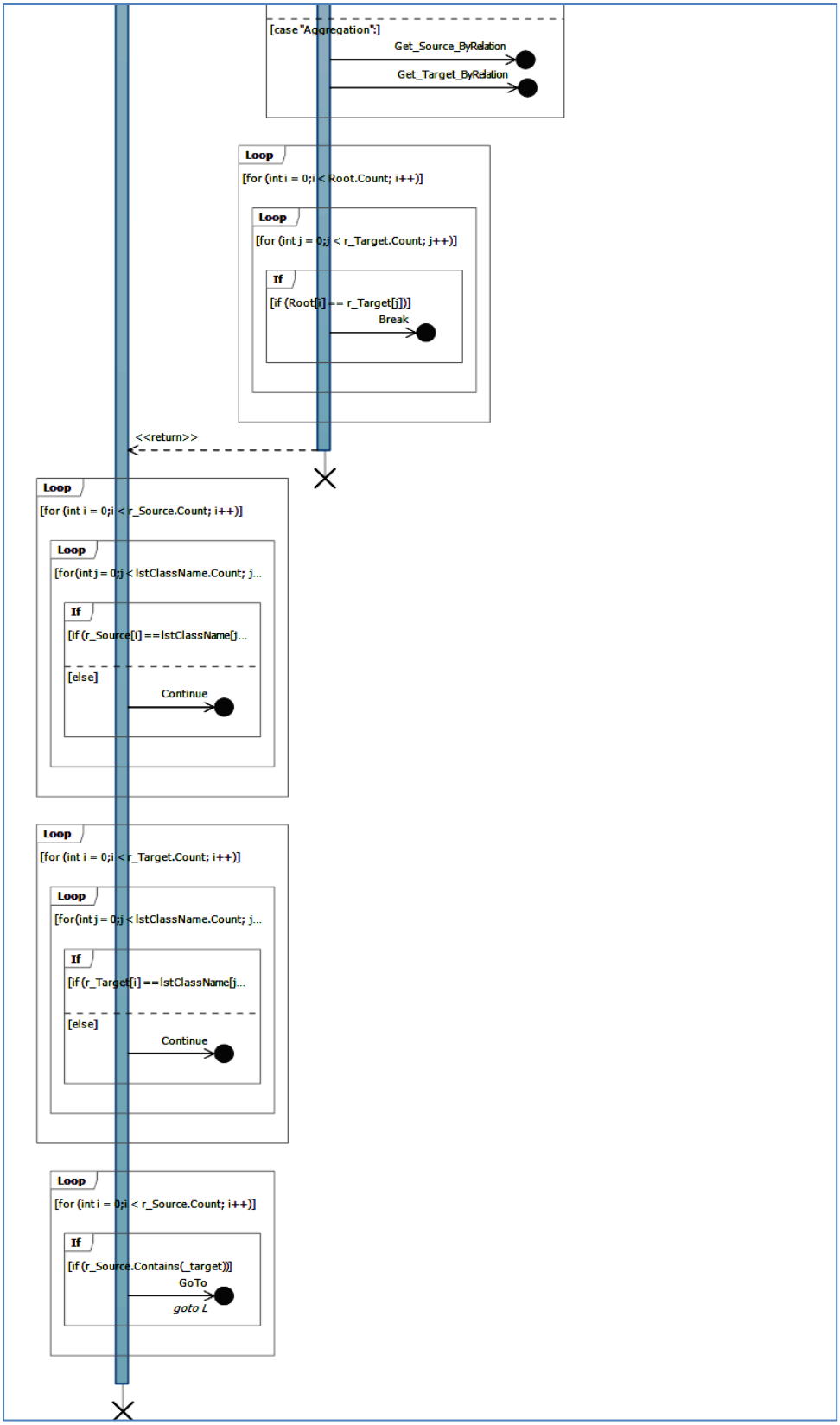


Figure (D-5) Sequence Diagram for AIF Algorithm (b)

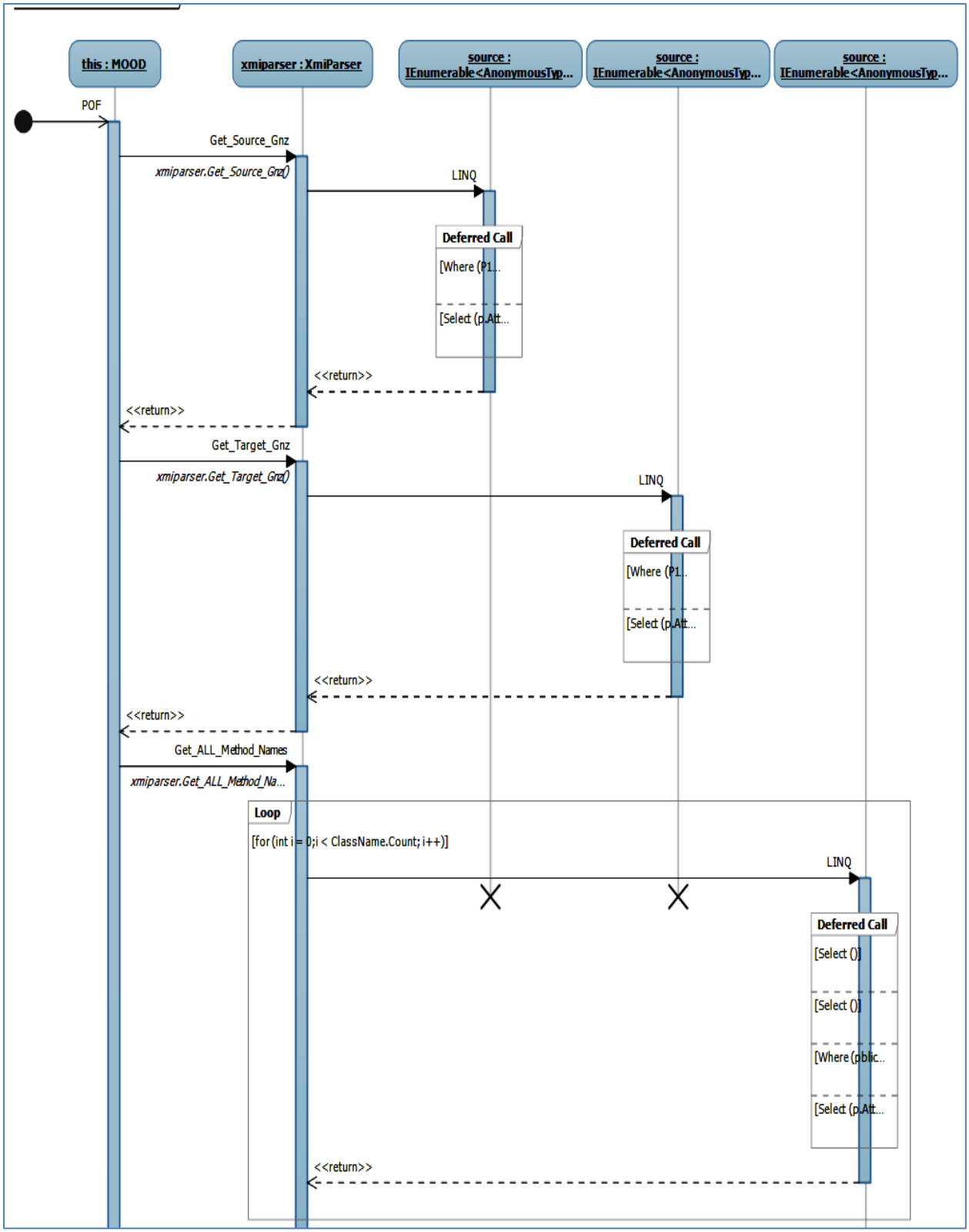


Figure (D-6) Sequence Diagram for POF Algorithm (a)

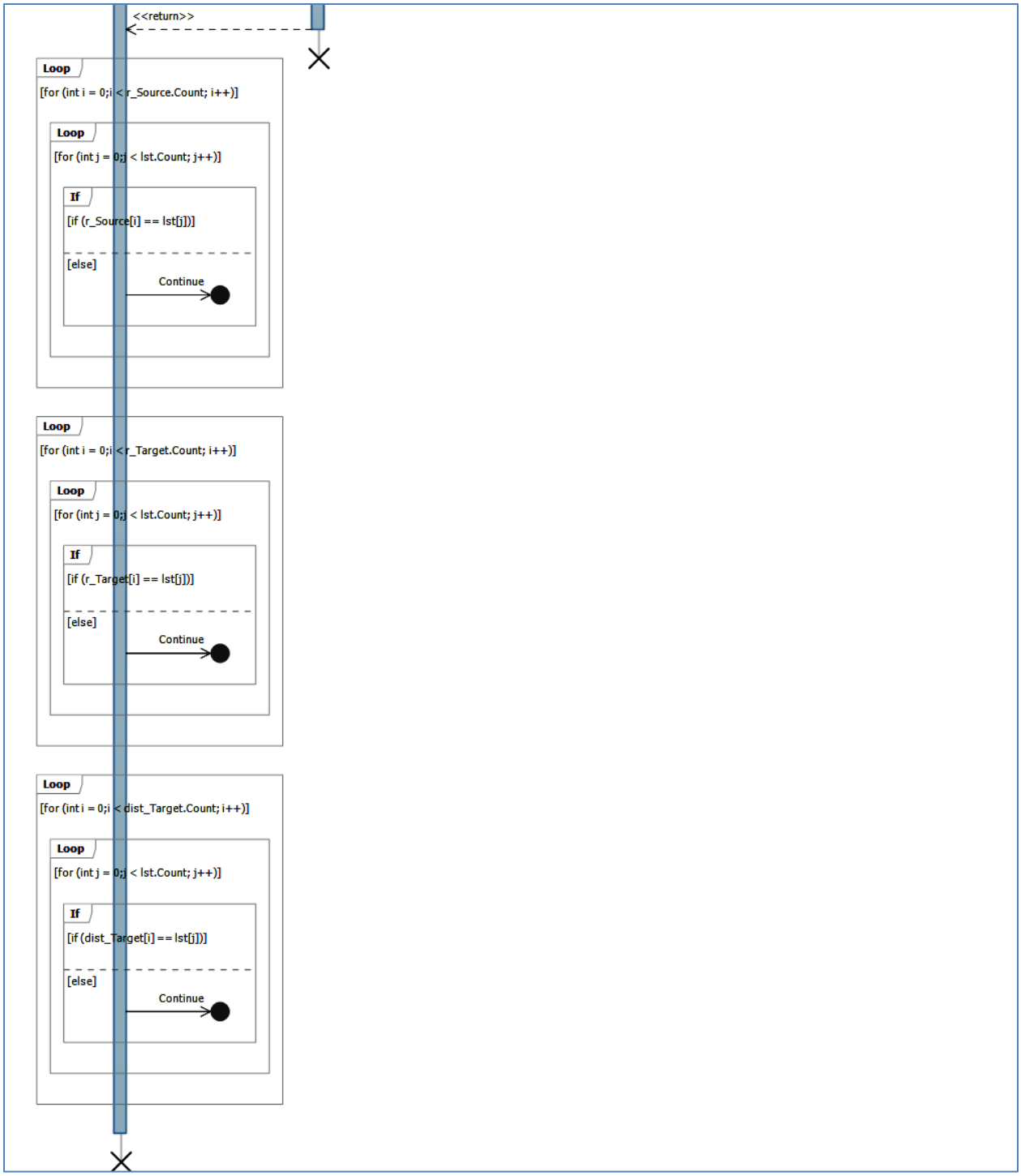


Figure (D-6) Sequence Diagram for POF Algorithm (b)

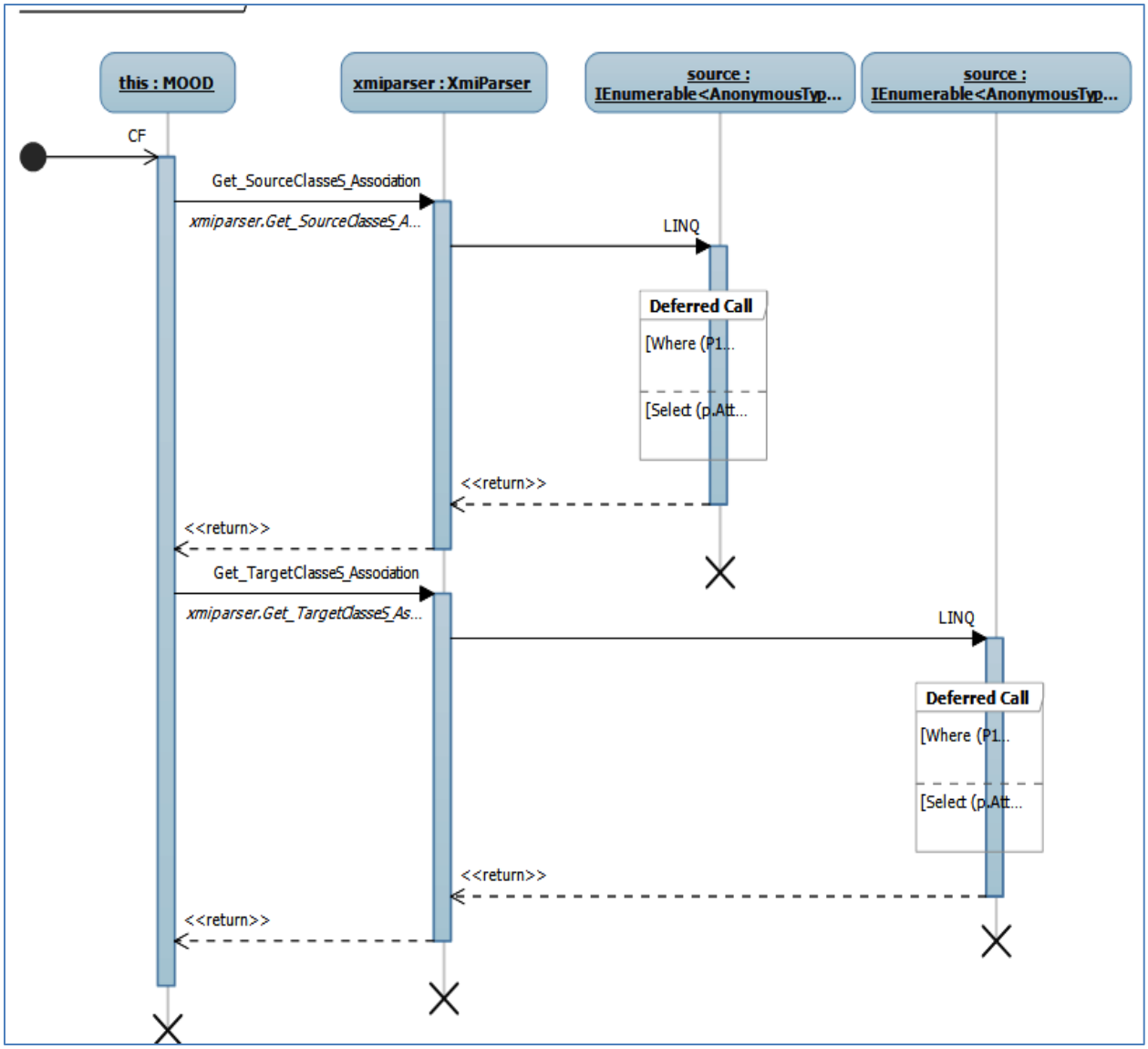


Figure (D-7) Sequence Diagram for CF Algorithm

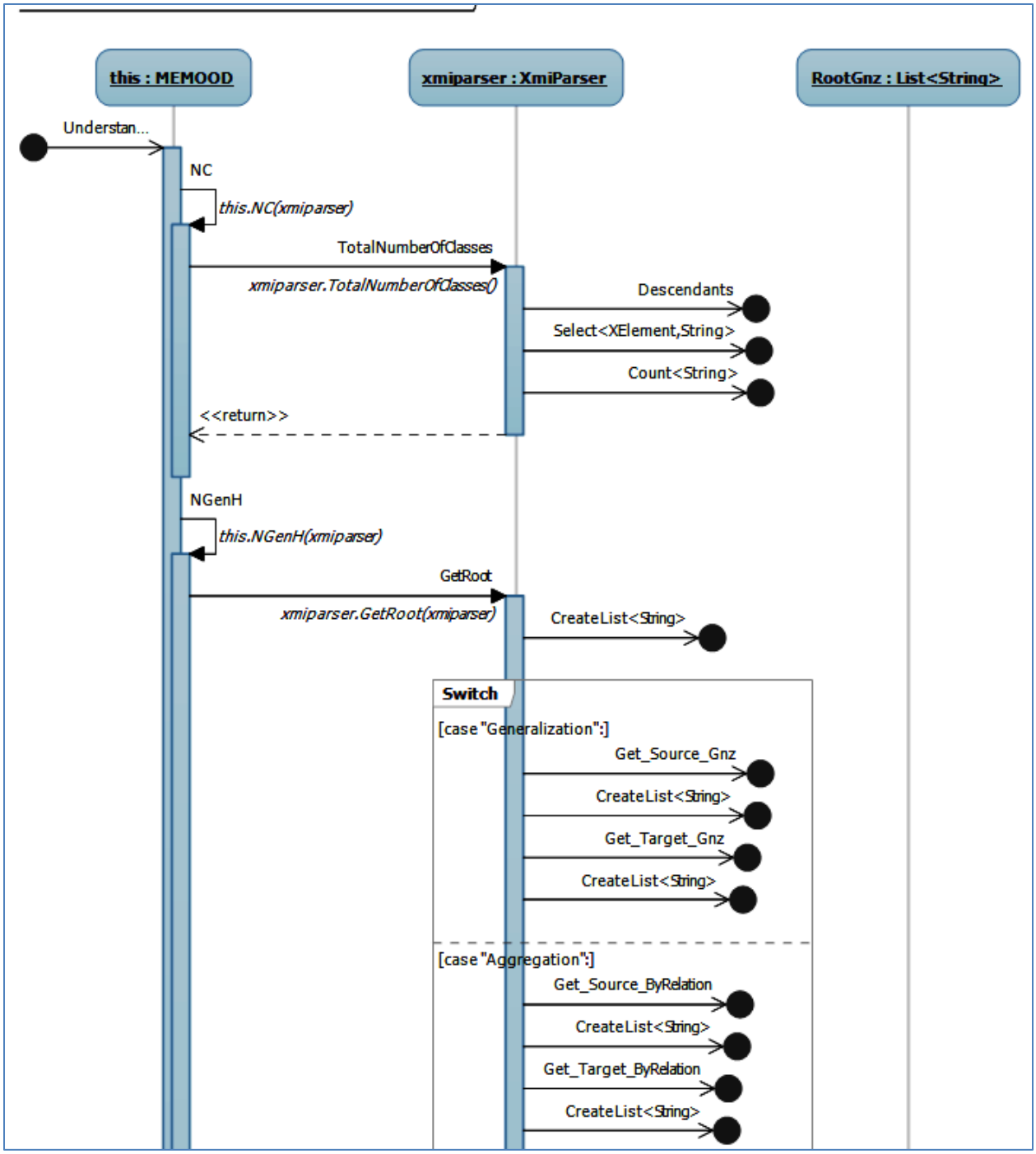


Figure (D-8) Sequence Diagram for Understandability Algorithm (a).

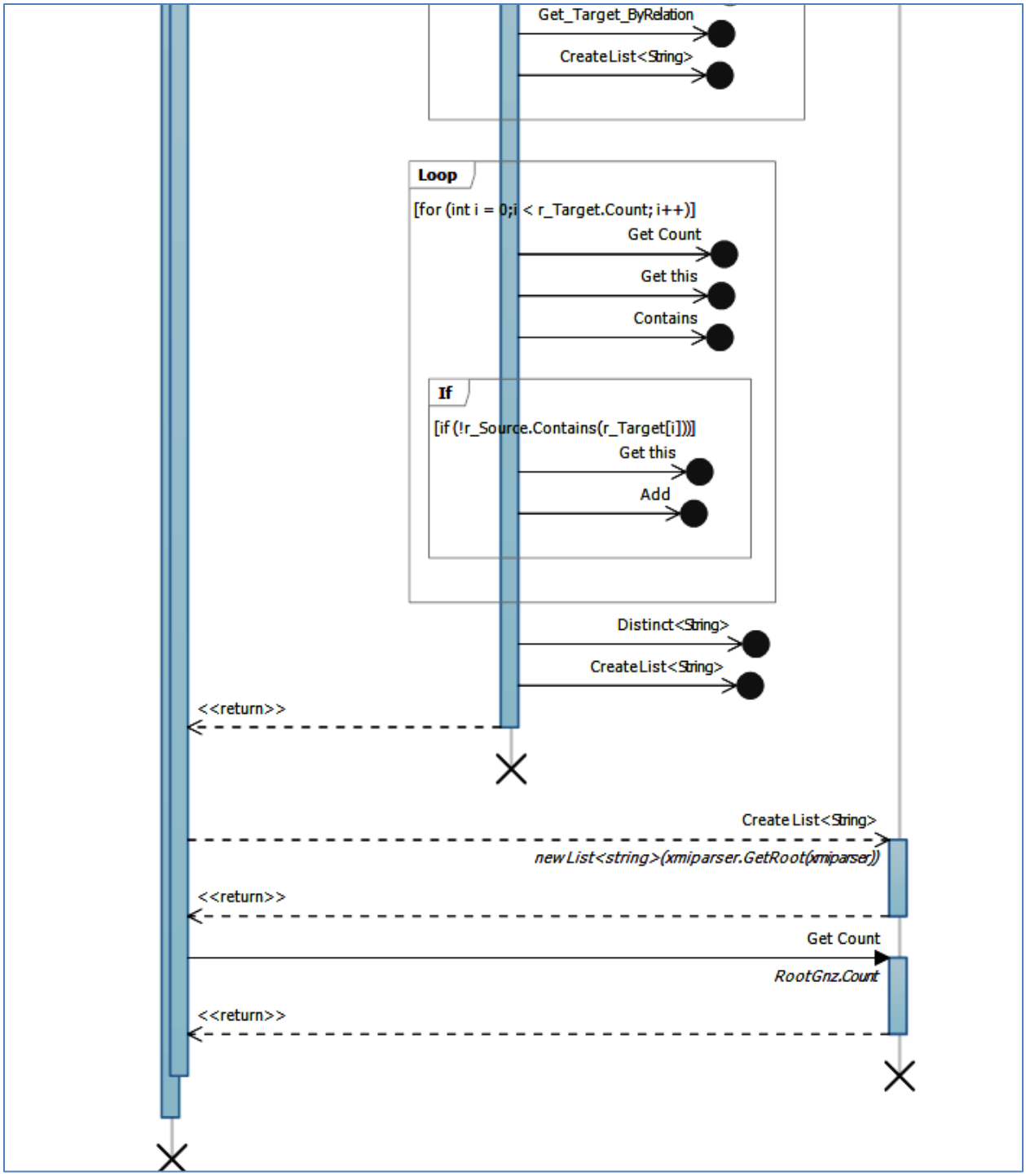


Figure (D-8) Sequence Diagram for Understandability Algorithm (b)

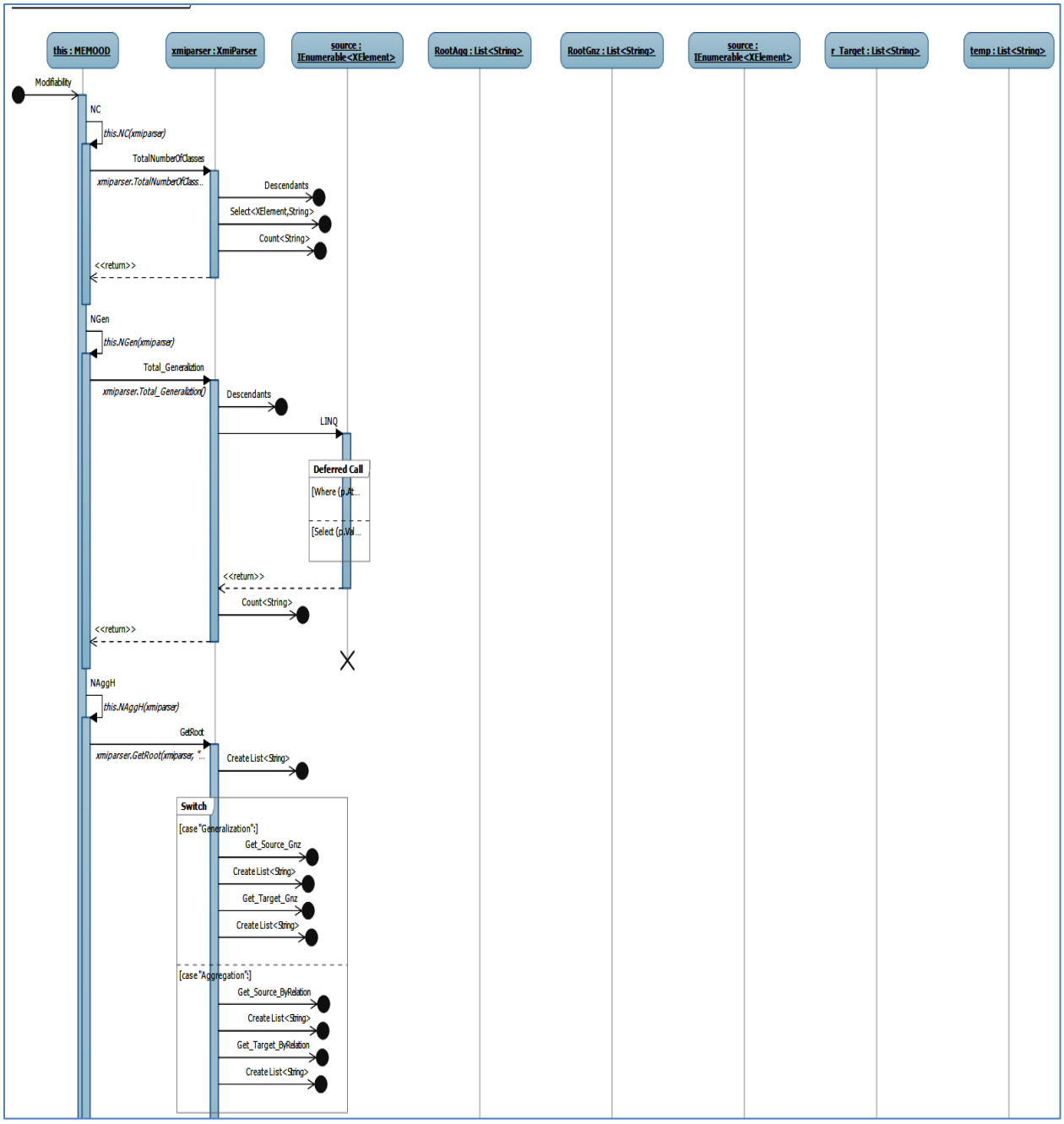


Figure (D-9) Sequence Diagram for Modifiability Algorithm (a)

Figure (D-9) Sequence Diagram for Modifiability Algorithm (b)

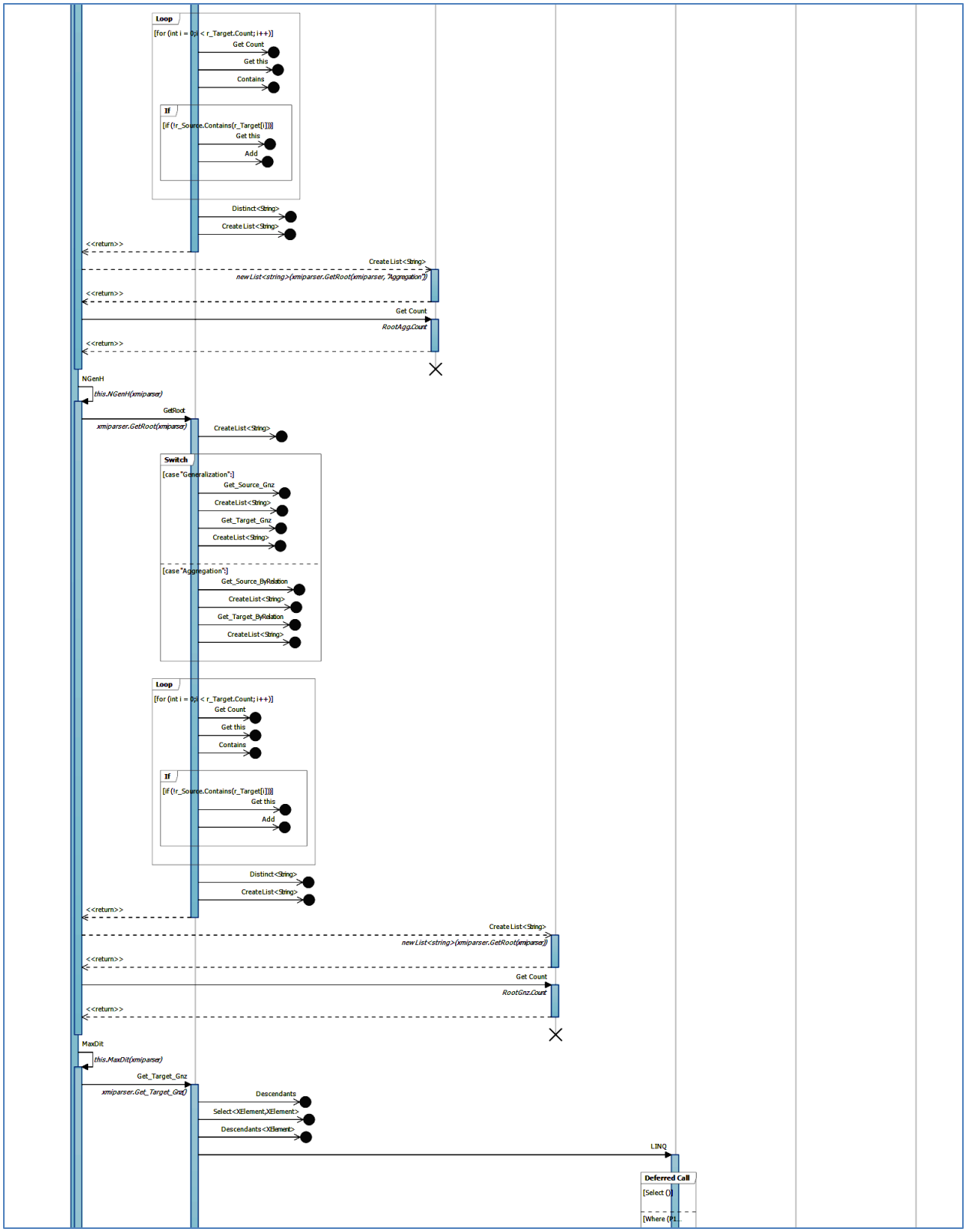


Figure (D-9) Sequence Diagram for Modifiability Algorithm (c)

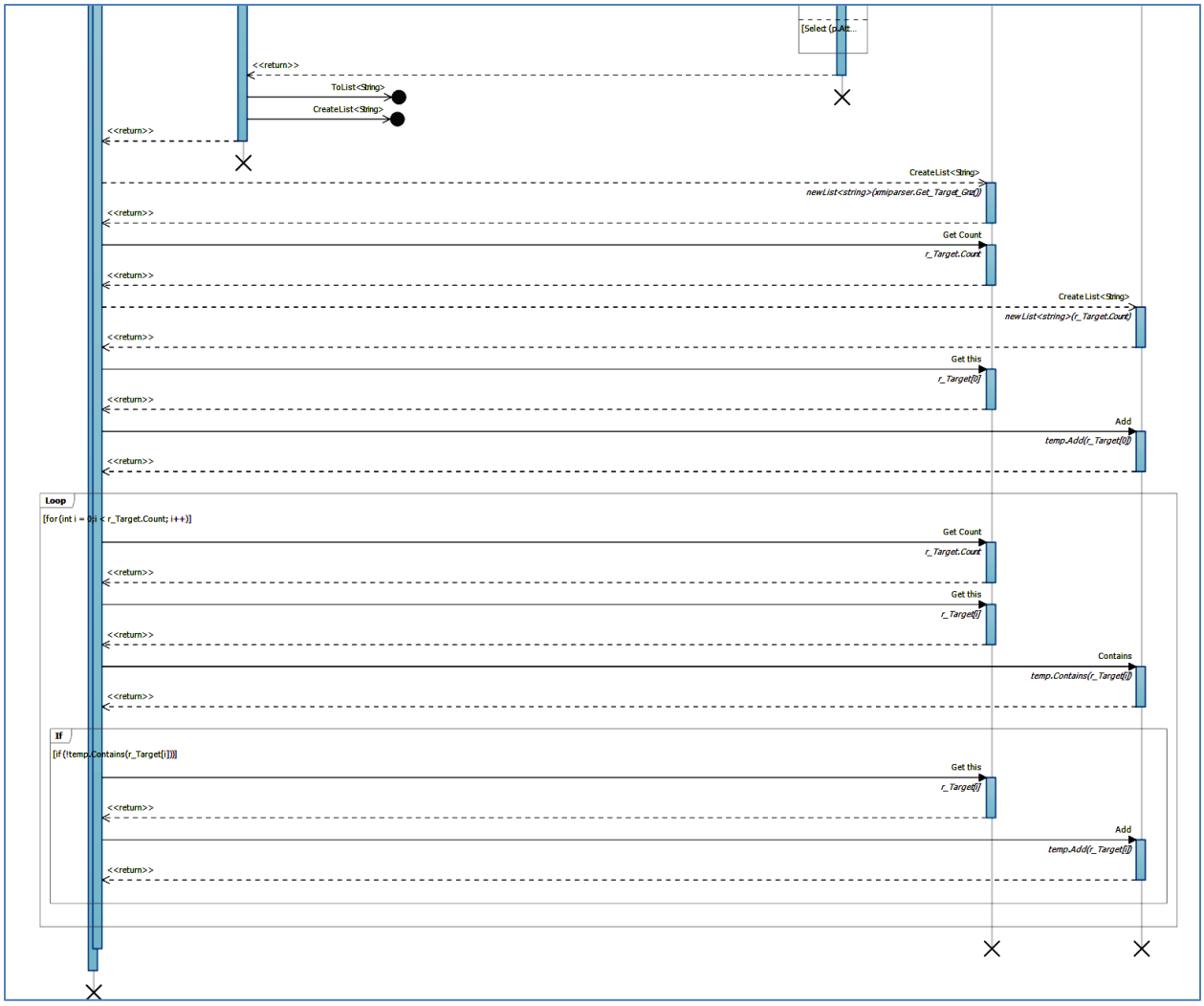


Figure (D-9) Sequence Diagram for Modifiability Algorithm (d)

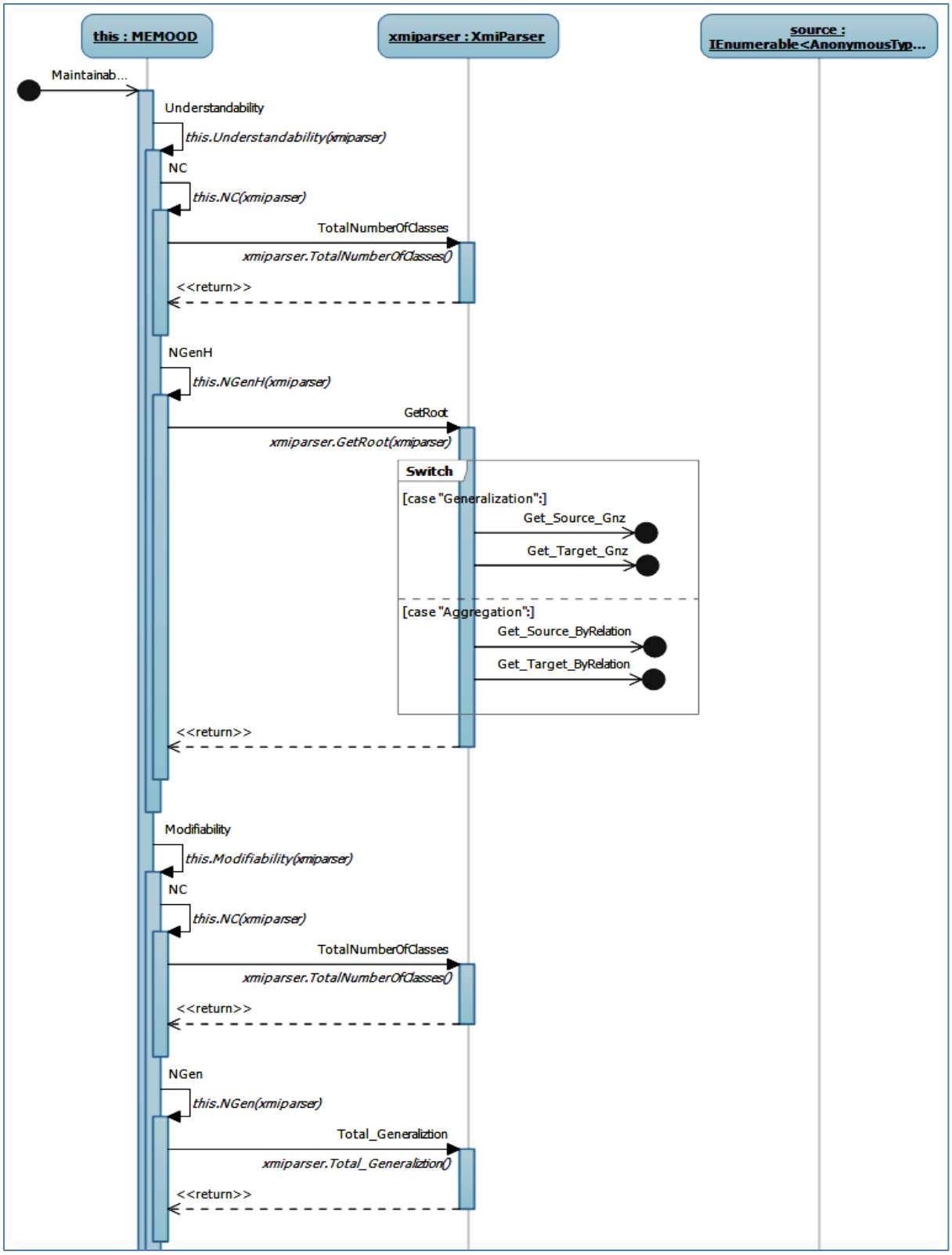


Figure (D-10) Sequence Diagram for Maintainability Algorithm (a)

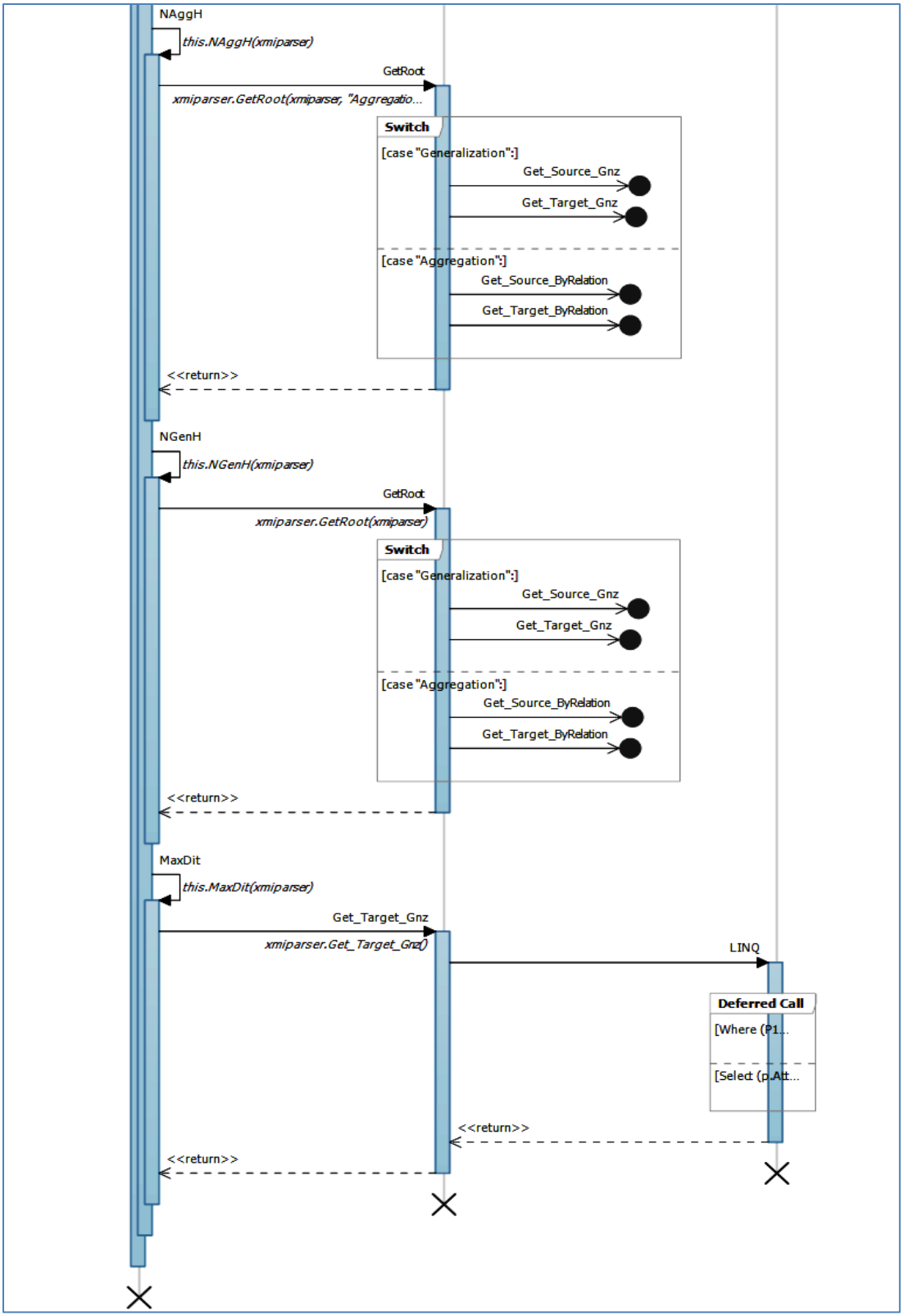


Figure (D-10) Sequence Diagram for Maintainability Algorithm (b)

Class Diagrams

Class diagrams (detailed) for KDM tool are depicted in figure (D-11), (D-12), (D-13), (D-14), and (D-15).

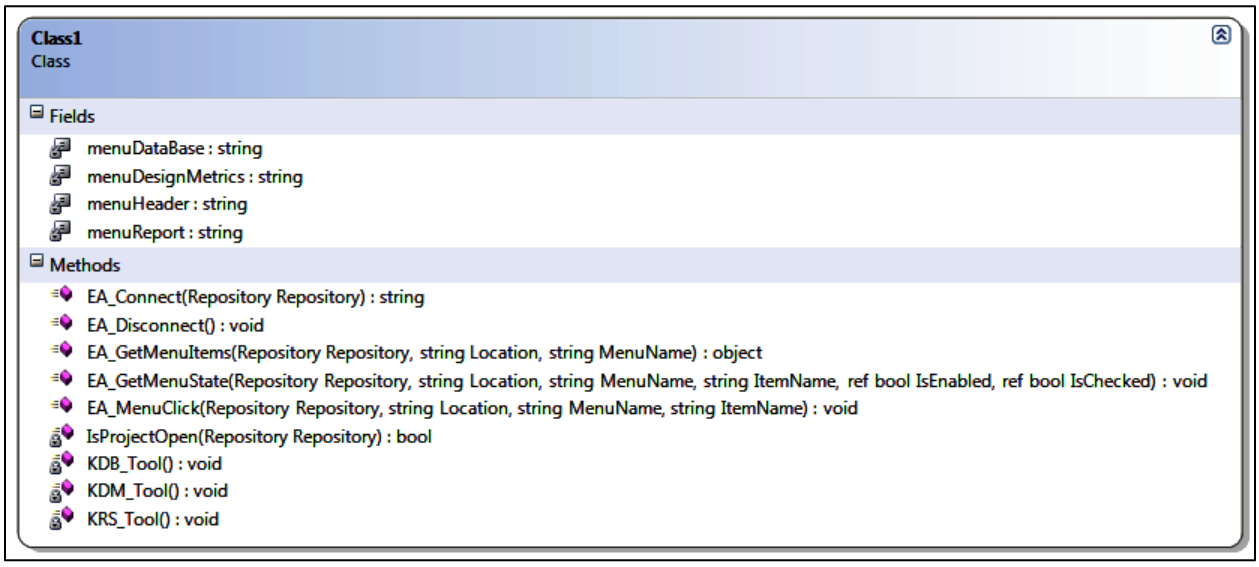


Figure (D-11) Add-In Class Diagram

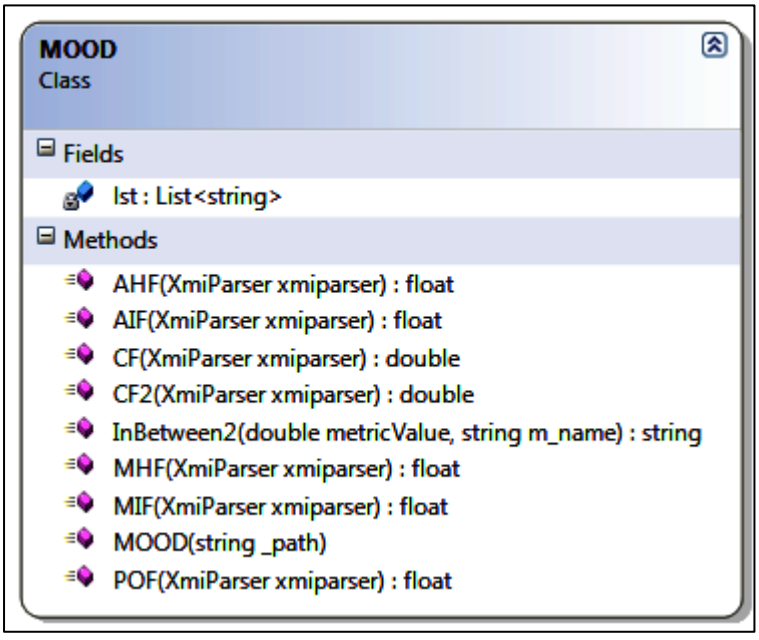


Figure (D-12) MOOD Metrics Class Diagram

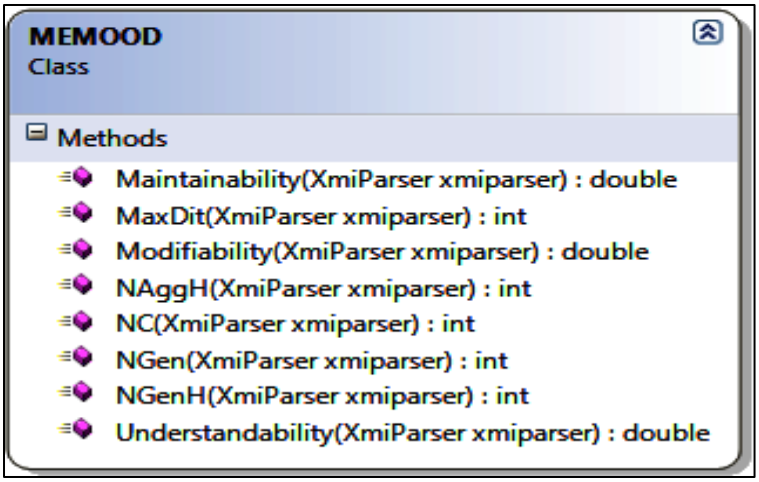


Figure (D-13) Class Diagram for MEMOOD metrics

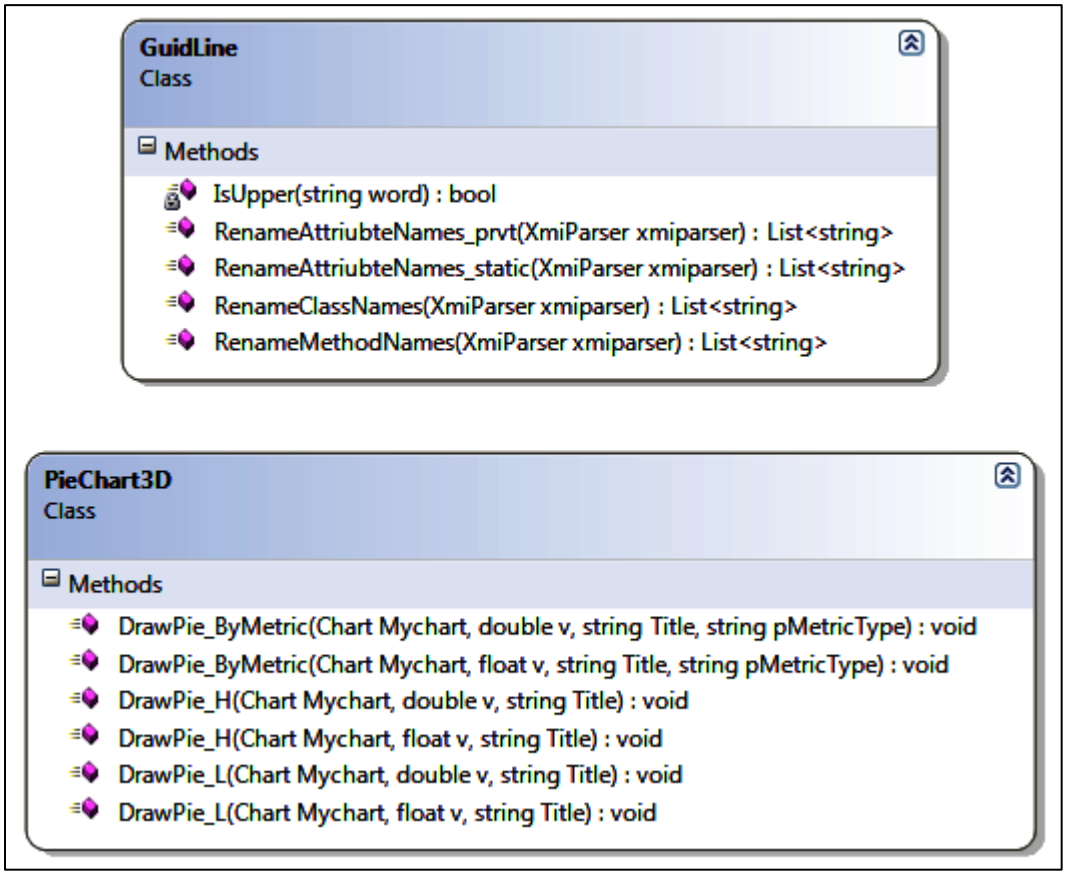


Figure (D-14) Class Diagram for 3D-Pie Chart and Naming Conventions

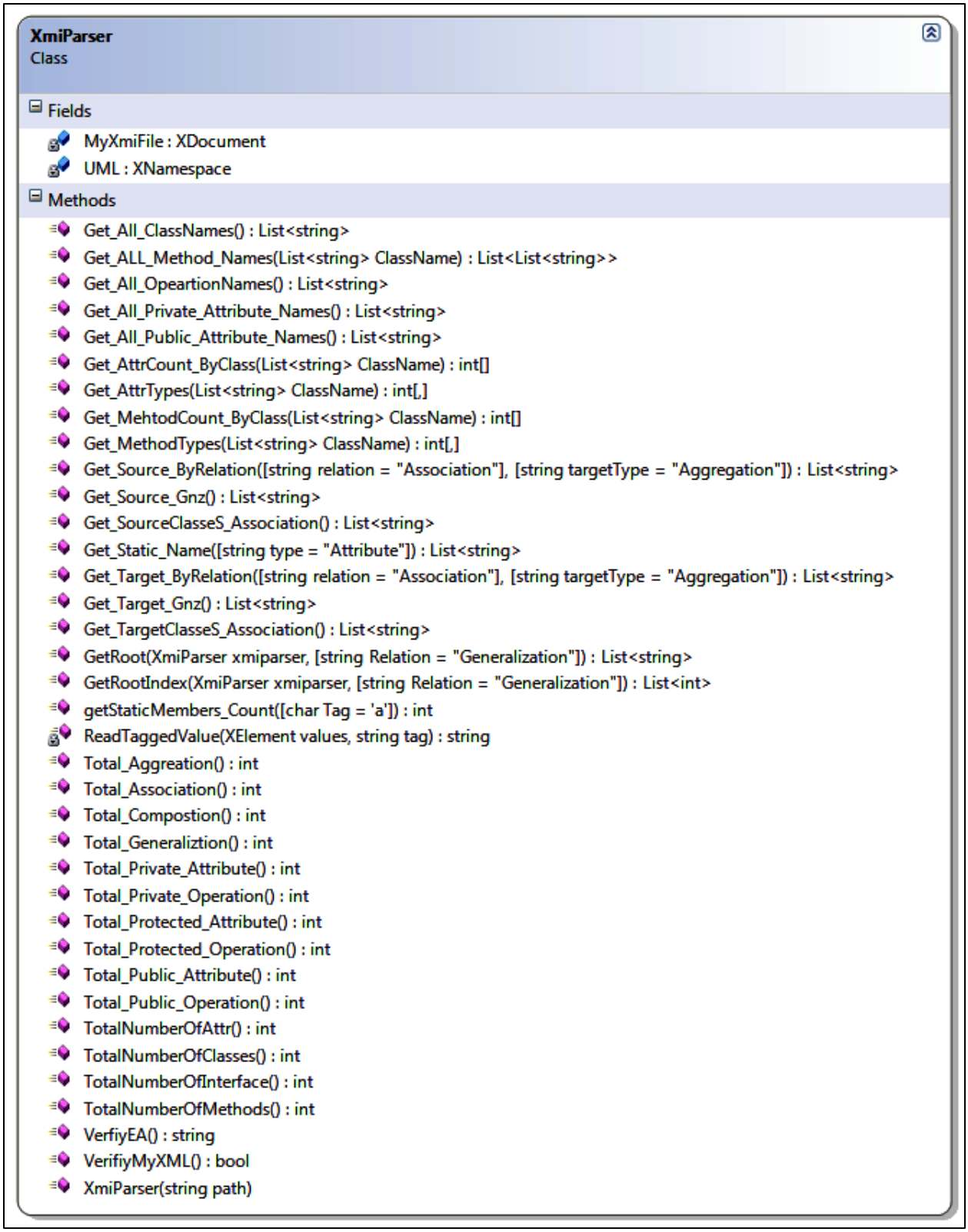


Figure (D-15) Class Diagram for XMI Parser

NC	NGenH	NGen	NAggH	MaxDIT	Understandability	Modifiability	Maintainability
5	1	1	0	1	2.052	2.511	2.458
9	1	1	0	1	3.076	4.395	4.06431
12	1	1	0	1	3.844	5.808	5.2689
15	1	1	0	1	4.612	7.221	6.47
---	---	---	---	---	---	---	---
5	2	1	0	1	1.658	1.815	1.854
9	2	1	0	1	2.682	3.699	3.46
---	---	---	---	---	---	---	---
9	1	1	1	1	3.076	3.779	3.755
12	1	1	1	1	3.844	5.192	4.959
15	1	1	1	1	4.612	6.605	6.164
---	---	---	---	---	---	---	---
10	1	1	1	1	3.332	4.25	4.156
20	1	1	1	1	5.89	8.96	8.1722
30	1	1	1	1	8.452	13.67	12.18788
---	---	---	---	---	---	---	---
10	2	2	2	2	2.938	3.161	3.355
20	2	2	2	2	5.498	7.871	7.3714
30	2	2	2	2	8.058	12.581	11.387
---	---	---	---	---	---	---	---
10	3	3	3	3	2.544	2.072	2.55
20	3	3	3	3	5.104	6.782	6.5704
30	3	3	3	3	7.664	11.942	10.586
---	---	---	---	---	---	---	---
10	4	4	4	4	2.15	0.98	1.75
20	4	4	4	4	4.17	5.693	5.769
30	4	4	4	4	7.22	10.403	9.785
----	---	---	---	---	---	---	---
10	2	1	1	1	2.938	3.554	3.5531
10	3	1	1	1	2.54	2.85	2.94
10	1	2	1	1	3.3	4.077	4.069
10	1	3	1	1	3.3	3.904	3.982
10	1	1	2	1	3.3	3.63	3.84
10	1	1	3	1	3.3	3.018	3.538
10	1	1	1	2	3.3	4.646	4.335
10	1	1	1	3	3.3	5.042	4.55
---	---	---	---	---	---	---	---
5	1	3	0	1	2.052	2.165	2.28
5	1	3	0	2	2.052	2.56	2.48
5	1	3	1	1	2.052	1.549	1.97
5	1	3	1	2	2.052	1.94	2.173
---	---	---	---	---	---	---	---
9	1	3	0	1	3.076	4.049	3.89

9	1	3	0	2	3.076	4.445	4.089
9	1	3	1	1	3.076	3.433	3.581
9	1	3	1	2	3.076	3.829	3.7801
---	---	---	---	---	---	---	---
12	1	3	1	1	3.844	4.846	3.78
12	1	3	1	2	3.844	5.242	4.98
---	---	---	---	---	---	---	---
15	1	3	1	1	4.612	6.259	5.9907
15	1	3	1	2	3.612	6.655	6.189

Based on above experiments, the following conclusions are made:

1. When NC increases, Understandability increases.
2. When NGenH increases, Understandability decreases.
3. When NC increases, all increases.
4. Modifiability and Maintainability decrease When NGenH increases.
5. Modifiability and Maintainability decrease minimally When NAggH increases. Also the affect increases when NC increases.
6. When NGen increases, Modifiability and Maintainability decrease.
7. NC is extrusive with Maintainability.
8. NGen and NGenH are inversive with Maintainability.
9. NAggH and MaxDIT have minimal effect on Maintainability.

Questionnaire Form

A questionnaire has been conducted of a twenty person in the field (Computer Science and Software Engineering). The questionnaire contains four sections as in the table below.

Table (F-1) Questionnaire Form (a).

ت	المحاور		فقرات المحـــــاور	غير موافق	موافق	موافق جدا
1	تقييم الادوات بشكل عام	1	ان عملية تثبيت الادوات KDM,KRS,KDB سهلة			
		2	هل اضافت الادوات خصائص زادت من امكانيات Enterprise Architect(EA)			
		3	ان الادوات ذات كفاءة، اذ توفر بيئة مثالية للمستخدم في تمثيل المقاييس.			
		4	الادوات ذات وثوقية عالية (High Reliability) في الاستخدام.			
		5	واجهات الادوات سهلة الاستخدام من قبل المستخدم.			
		6	تضمنت واجهات الادوات كافة الخيارات الضرورية لمهندس البرمجيات ضمن طوري التحليل والتصميم من ناحية الجودة.			
		7	وفرت الادوات user guide تعريفية للمستخدمين مما سهل عملية استخدام الادوات الثلاثة بشكل عام.			
2	تقييم الأداة KDM	1	الأداة سريعة في تحليل XMI document الخاصة بوصف التصميم.			
		2	وفرت الاداة الجهد اللازم للمستخدم في الحصول على المقاييس بطريقة تلقائية عوضا عن الطريقة اليدوية.			
		3	وفرت الاداة قيم دقيقة في حساب المقاييس.			
		4	وفرت الاداة اقتراح التسمية للتصميم Naming Conventions.			
		5	شملت الاداة المقاييس الاكثر استخداما في التصميم الكائني المنحى OOD.			
		6	قدمت الاداة رسوم بيانية Charts لتكون رؤية واضحة للمستخدم تمكنه من فهم المقاييس.			

Table (F-1) Questionnaire Form (b).

ت	المحاور		فقرات المحـــــاور	غير موافق	موافق	موافق جدا
3	تقييم الاداة KRS	1	الأداة سريعة في تحليل XML document الخاصة بوصف المقاييس.			
		2	وفرت الاداة الجهد اللازم للمستخدم في توثيق المقاييس.			
		3	قدمت الاداة التقارير Reports لتكون رؤية واضحة للمستخدم في توثيق المقاييس			
		4	اعطى التقرير زيادة في قابلية الفهم لدى رئيس الفريق او مدير المشروع.			
4	تقييم الاداة KDB	1	الأداة سريعة في تحليل XML document الخاصة بوصف المقاييس.			
		2	وفرت الاداة الجهد اللازم للمستخدم في خزن واسترجاع المقاييس.			
		3	قدمت الاداة امكانية خزن البيانات لغرض المقارنة مع المشاريع المستقبلية Historical Data.			
		4	وفرت الاداة عمليات الاضافة والحذف و العرض في قاعدة البيانات مما زاد من مرونة الاستخدام.			
		5	مكنت الاداة المستخدم من استخدام قاعدة البيانات في مشاريع التنقيب عن البيانات Data Mining.			