การวัดความสามารถในการบำรุงรักษาซอฟต์แวร์เชิงวัตถุในขั้นตอนการออกแบบ โดยใช้มาตรวัดความซับซ้อนของโครงสร้างและมาตรวัดสุนทรียภาพ

นางสาวเมทินี เขียวกันยะ

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรดุษฎีบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2549

MEASURING OBJECT-ORIENTED SOFTWARE MAINTAINABILITY IN DESIGN PHASE USING STRUCTURAL COMPLEXITY AND AESTHETIC METRICS

Miss Matinee Kiewkanya

A Dissertation Submitted in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy Program in Computer Engineering
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Academic Year 2006

Thesis Title	MEASURING OBJECT-ORIENTED SOFTWARE MAINTAINABILITY IN DESIGN PHASE USING STRUCTURAL COMPLEXITY AND AESTHETIC METRICS
By	Miss Matinee Kiewkanya
Field of Study	Computer Engineering
Thesis Advisor	Associate Professor Pornsiri Muenchaisri, Ph.D.

Accepted by the Faculty of Engineering, Chulalongkorn University in Partial Fulfillment of the Requirements for the Doctoral Degree

DL Lavansiri .... Dean of the Faculty of Engineering

(Professor Direk Lavansiri, Ph.D.)

THESIS COMMITTEE

P. Chongstitvatana .... Chairman

(Associate Professor Prabhas Chongstitvatana, Ph.D.)

Pornsiri Muenchaisri .... Thesis Advisor

(Associate Professor Pornsiri Muenchaisri, Ph.D.)

.... Member

(Associate Professor Wanchai Rivepiboon, Ph.D.)

.... Member

(Associate Professor Boonserm Kijsirikul, Ph.D.)

.... Member

(Songsak Rongviriyapanich, Ph.D.)

เมทินี เขียวกันยะ : การวัดความสามารถในการบำรุงรักษาซอฟต์แวร์เชิงวัตถุในขั้นตอนการออกแบบโดยใช้มาตรวัดความซับซ้อนของโครงสร้างและมาตรวัดสุนทรีภาพ. (MEASURING OBJECT-ORIENTED SOFTWARE MAINTAINABILITY IN DESIGN PHASE USING STRUCTURAL COMPLEXITY AND AESTHETIC METRICS) อ. ที่ปรึกษา: รศ.ดร.พรศิริ หมื่นไชยศรี, 125 หน้า.

คุณภาพของการออกแบบซอฟต์แวร์มีผลกระทบอย่างยิ่งต่อคุณภาพของซอฟต์แวร์ที่ผลิตขึ้นในภายหลัง ประโยชน์ข้อหนึ่งของรูปแบบเชิงวัตถุคือความง่ายของการบำรุงรักษา การวัดความสามารถในการบำรุงรักษาในขั้นตอนของการออกแบบจะช่วยให้นักออกแบบซอฟต์แวร์ตัดสินใจได้ว่าการออกแบบซอฟต์แวร์นั้นควรจะมีการปรับปรุงหรือไม่ เพื่อเป็นการลดค่าใช้จ่ายในขั้นตอนของการบำรุงรักษาซอฟต์แวร์ งานวิจัยนี้นำเสนอการทดลองที่จัดทำขึ้นเพื่อสำรวจว่ามาตรวัดความซับซ้อนของโครงสร้างและมาตรวัดสุนทรียภาพสามารถใช้เป็นตัวบ่งชี้ความสามารถในการบำรุงรักษาของแผนภาพคลาสและแผนภาพซีเควนซ์ได้หรือไม่ และเพื่อทำการสร้างโมเดลความสามารถในการบำรุงรักษาจากมาตรวัดทั้งสองกลุ่ม การสร้างโมเดลความสามารถในการบำรุงรักษาจะใช้เทคนิค 3 เทคนิค ได้แก่ การวิเคราะห์จำแนกกลุ่ม ต้นไม้ตัดสินใจ และโครงข่ายประสาทเทียมเพอร์เซบตรอนแบบหลายชั้น เพื่อหาโมเดลที่ดีที่สุด ผลการทดลองเบื้องต้นแสดงว่าโมเดลความสามารถในการบำรุงรักษาที่สร้างขึ้นโดยใช้โครงข่ายประสาทเทียมเพอร์เซบตรอนแบบหลายชั้น เป็นโมเดลที่ดีที่สุดเมื่อพิจารณาจากความแม่นยำของโมเดล นอกจากนั้น งานวิจัยนี้มีการสร้างเครื่องมืออัตโนมัติสำหรับทำนายความสามารถในการบำรุงรักษาโดยใช้โมเดลความสามารถในการบำรุงรักษา และยังได้นำเสนอมาตรวัดความซับซ้อนของโครงสร้างชุดใหม่สำหรับวัดความสามารถในการบำรุงรักษาจากแผนภาพคลาสอีกด้วย

ภาควิชา.....วิศวกรรมคอมพิวเตอร์.... ลายมือชื่อนิสิต ....
สาขาวิชา....วิศวกรรมคอมพิวเตอร์... ลายมือชื่ออาจารย์ที่ปรึกษา..
ปีการศึกษา..........2549...............

4471833621 : MAJOR COMPUTER ENGINEERING

KEY WORD: MAINTAINABILITY / PREDICTION MODEL / SOFTWARE QUALITY / DESIGN METRICS / OBJECT-ORIENTED

MATINEE KIEWKANYA : MEASURING OBJECT-ORIENTED SOFTWARE MAINTAINABILITY IN DESIGN PHASE USING STRUCTURAL COMPLEXITY AND AESTHETIC METRICS. THESIS ADVISOR : ASSOC.PROF. PORNSIRI MUENCHAISRI, Ph.D., 125 pp.

Qualities of software design heavily affect on qualities of software ultimately developed. One of the claimed advantages of object-oriented paradigm is the ease of maintenance. Assessing maintainability in design phase will help software designers to decide that the design of software should be altered or not. This will help reducing cost of software maintenance in later phases. This research presents a controlled experiment carried out to investigate whether structural complexity and aesthetic metrics can be indicators of class and sequence diagrams maintainability and to establish maintainability models from both metric sets. Maintainability models are constructed by using three techniques: Discriminant analysis, Decision tree and MLP neural network in order to find the best one among these models. The preliminary result shows that the maintainability models constructed by applying MLP neural network are the best models in terms of the model accuracy. Moreover, this research constructs an automated tool for predicting maintainability using the maintainability models and also proposes a new set of structural complexity metrics for measuring maintainability from class diagrams.

Department ....Computer Engineering... Student's Signature ....
Field of study ..Computer Engineering... Advisor's Signature ...P. Muenchaisri.......
Academic year ....2006.................

ACKNOWLEDGEMENTS

I am especially deeply grateful to my thesis advisor, Associate Professor Dr. Pornsiri Muenchaisri, who provided me the great deal of guidance in the area of software measurement. She always motivates me throughout this research period and taught me how to be a good advisor. Without her, my thesis would have never been accomplished. Moreover, she also gave me an opportunity to get the fund to participate international conferences in oversea. This opportunity made me open my vision widely.

I also wish to express my special thanks to the thesis committee, Associate Professor Dr. Prabhas Chongstitvatana, Associate Professor Dr. Wanchai Rivepiboon, Associate Professor Dr. Boonserm Kijsirikul and Dr. Songsak Rongviriyapanich, for their valuable advices, reading and criticizing the manuscript. They helped me focus on my research activities. Thanks are also extended to Lecturer Nakornthip Prompoon and Lecturer Chate Patanothai.

I wish to thank the graduate students within the Computer Engineering Department at Chulalongkorn University who spent their time to participate in my experiment. This research work is the part of “Object-Oriented Software Metrics” project, financed by Chulalongkorn University - Stock Exchange of Thailand (SET) Linkage Research Fund. I would like to thank the Royal Thai Government Scholarship for the financial support during my study. This fund also gave me a chance to have research collaboration at National ICT Australia (NICTA). I also would like to thank Professor Dr. Ross Jeffery and Elsa Colella who looked after me when I did my research at NICTA.

I would like to thank all my colleagues and friends at the Software Engineering Laboratory, especially Nongyao Jindasawat, for helps and discussions on my thesis. I would like to thank my best friend, Benjamas Panyangam, for her everlasting friendship.

My thanks also go to all my past and present lecturers for their valuable lectures and instructions. Finally, I would like to express my sincere gratitude to my parents and family members for their everlasting love, supports and hearty encouragements throughout my life.

TABLE OF CONTENTS

Page

Chapter

Chapter Page

Chapter Page

LIST OF TABLES

Table Page

Table Page

LIST OF FIGURES

Figure Page

CHAPTER I

INTRODUCTION

1.1 Motivation

Software maintenance phase is frequently concerned as less important than the design and development phases of the system life cycle. In fact, many researches report that 50-70% of the total software life cycle is spent on software maintenance [1,2,3].

Object-oriented techniques have become increasingly popular as a software developing methodology. More empirical research is needed to investigate that objected-oriented techniques provide significant advantages over other techniques. One particular area which warrants immediate investigations is maintainability of object-oriented software. Object-oriented development techniques are promised to increase maintainability through the better data encapsulation. If this statement is true, an organization switching to object-oriented techniques will be likely to save large amounts of money through the lifetime of an object-oriented software. Usually, maintainability is the external quality characteristic that can be evaluated once the software is finished or nearly finished. In order to improve the quality and reduce the increasing high cost of software maintenance, measuring maintainability should be done at the early phase.

Software metrics can be used to capture software maintainability. Bandi and his colleagues reported the experimental result of validating three existing object-oriented design complexity metrics. The result showed that each of the three metrics can be a useful indicator for predicting maintenance performance [4]. Genero et al. investigated the possibility that structural complexity and size metrics can be used as good predictors of class diagram maintainability by constructing maintainability prediction models based on metrics of UML class diagrams [5]. Some software maintainability metrics which can be applied to UML specification are also proposed by [6,7,8].

The Unified Modeling Language (UML) [9] is accepted as an industrial standard for modeling object-oriented design. It defines notations and semantics of modeling elements and the relationships among these elements. In its current form, class and sequence diagrams are two major artifacts acted as the blueprints of object-oriented software. Class diagram, a conceptual model of object-oriented software, shows the classes of the system, their inter-relationships, and the operations and attributes of the classes. Meanwhile, class diagram represents static structure, dynamic structure of software is represented by sequence diagram. Sequence diagram is utilized for modeling software behavior in each scenario. Therefore, the quality of object-oriented software ultimately implemented is heavily dependent on the quality of both diagrams. From now on, the term UML class and sequence diagrams will be interchanged with the term software design model.

This thesis proposes methodology for constructing maintainability models from metrics called structural complexity metrics and aesthetics metrics. These metrics can be measured from class and sequence diagrams produced at early phase of software life cycle. Software developers can utilize the models to identify 3 maintainability levels of software design model: easy, medium and difficult. When a software design model is categorized into medium or difficult level, software developers can decide whether to redesign it or not in order to improve its maintainability. An automated tool for predicting maintainability of software design model is also constructed.

1.2 Objectives

The objectives of this research are as follows:

1. To investigate whether structural complexity metrics and aesthetic metrics can be indicators of class and sequence diagrams maintainability and its sub-characteristics: understandability and modifiability.
2. To create understandability, modifiability and maintainability models from structural complexity metrics and aesthetic metrics using classification techniques.
3. To develop an automated tool for measuring structural complexity metrics from UML class and sequence diagrams and predicting understandability, modifiability and maintainability by using the obtained prediction models.

4. To propose a new set of structural complexity metrics for measuring understandability and modifiability from class diagrams.

1.3 Scope and Limitation

1. This work focuses on only two sub-characteristics of maintainability namely understandability and modifiability.
2. Metrics used in this work are structural complexity metrics and aesthetic metrics.
3. This work will use more than 30 software design models for constructing understandability, modifiability and maintainability models.
4. Class and sequence diagrams used in this work must be constructed using Rational Rose.
5. The tool for transforming class and sequence diagrams into XML document is Unisys Rose XML Tool.
6. This work will construct an automated tool for measuring structural complexity metrics from UML class and sequence diagrams. This tool can predict understandability, modifiability and maintainability from structural complexity metrics by using the obtained prediction models.

1.4 Contribution

The outcomes of this research will be the followings:

1. Models for predicting understandability, modifiability and maintainability of UML class and sequence diagrams.
2. An automated tool for measuring structural complexity metrics and predicting understandability, modifiability and maintainability by using the obtained prediction models.
3. A new set of structural complexity metrics for measuring understandability and modifiability from class diagrams.

1.5 Research Methodology

1. Review and study the research papers related to metrics and maintainability of object-oriented design.
2. Study XML, Rational Rose and Unisys Rose XML Tool.
3. Study Discriminant analysis, Decision tree and MLP neural network.
4. Set up and perform an experiment in order to capture understandability, modifiability and maintainability of sample software design models.
5. Construct understandability, modifiability and maintainability models using Discriminant analysis, Decision tree and MLP neural network.
6. Compare models constructed using Discriminant analysis, Decision tree and MLP neural network.
7. Analyze the result and make conclusions.
8. Develop a tool for measuring structural complexity metrics and predicting understandability, modifiability and maintainability by using the obtained prediction models.
9. Define and derive a new set of structural complexity metrics for measuring understandability and modifiability from class diagrams.
10. Write thesis.

1.6 Organization of the Thesis

The remainder of the thesis is organized into six chapters as follows.

Chapter II describes theoretical background including introduction of UML, software measurement and some statistical and classification techniques used in this work. This chapter also reviews the literature in UML metrics, maintainability of object-oriented software and aesthetic criteria of class and sequence diagrams.

Chapter III presents a controlled experiment for constructing maintainability models from structural complexity and aesthetic metrics using 3 techniques: Discriminant analysis, Decision tree and MLP neural network.

Chapter IV proposes a new set of structural complexity metrics for measuring maintainability from class diagrams. Validating the proposed metrics is also presented.

Finally, chapter V concludes research work and presents some directions for the future work.

CHAPTER II

BACKGROUND AND LITERATURE REVIEW

2.1 Background

This section reviews the theoretical background used in this thesis including UML, software measurement, MANOVA and classification techniques.

2.1.1 UML

The Unified Modeling Language (UML) is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems [9]. The UML is a very important tool for modeling object-oriented software design. Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software.

Each UML diagram is designed to let developers and customers view the software from different perspective and varying degrees of abstraction. UML diagrams commonly created in visual modeling tools include Use case diagram, Class diagram, Object diagram, Sequence diagram, Collaboration diagram, State diagram, Activity diagram, Component diagram and Deployment diagram.

2.1.1.1 Class Diagram

The purpose of a class diagram is to depict the classes within a model. In an object-oriented application, classes have attributes (member variables), operations (member functions) and relationships with other classes. The fundamental element of the class diagram is an icon that represents a class. This icon is shown in Figure 2.1.

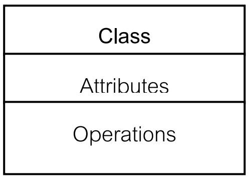


Figure 2.1: The class icon.

Class icon is simply a rectangle divided into three compartments. The top compartment contains the name of the class. The middle compartment contains a list of attributes, and the bottom compartment contains a list of operations. In many diagrams, the bottom two compartments are omitted. The goal is to show only those attributes and operations that are useful for the particular diagram.

The static relationships and their notations are as follows:

- **Association** is a relationship between classes which concerns the connection between its instances. An association is denoted by a line drawn between the participating classes.
- **Aggregation and composition** are relationships between an aggregate A and a component B, “B is a part of A”. Aggregation is also called a "has a" relationship. A weak form of aggregation is denoted with an open diamond. It denotes that the aggregate class (the class with the diamond touching it) is someway the “whole”, and the other class in the relationship is somehow the “part” of that whole. Composition is a strong form of aggregation. It is represented by the black diamond. Component cannot exist without aggregate and dies with its aggregate.
- **Generalization** is a relationship between a general concept A (superclass) and a more special concept B (subclass). The generalization relationship in UML is depicted by a triangular arrowhead. This arrowhead points to the base class. One or more lines proceed from the base of the arrowhead connecting it to the derived classes. Generalization is also called an "is a" relationship, because the subclass is a type of the super class.
- **Dependency** is a relationship that one class instantiates another or that it uses the other as an input parameter. It is represented by a dotted line with an open arrowhead.
- **Realization** is a relationship that one entity (normally an interface) defines a set of functionalities as a contract and the other entity (normally a class) "realizes" the contract by implementing the functionality defined in the contract. It is represented by a dotted line with a triangular arrowhead.

Table 2.1: Notation of class relationships.

Relationship	Notation
Association	————————
Generalization	———————▷
Aggregation	———————◇
Composition	———————◆
Dependency	- - - - - - - - - - ->
Realization	- - - - - - - - - - -▷

The UML notation for class diagram is shown below.

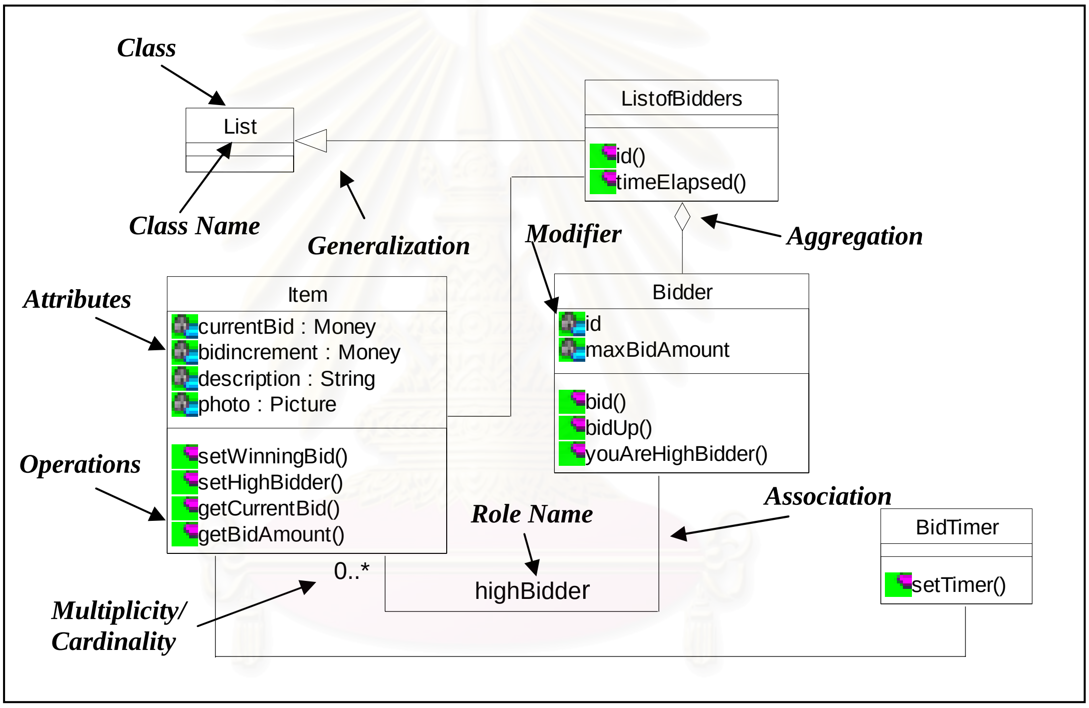


Figure 2.2: Notation for class diagram.

- Object : A specific entity or concept that has meaning in an application domain.
- Class : A definition of a set of potential objects that have the same data, behavior, and relationships.
- Attribute : A data value defined in a class and held within an object that has meaning within the application domain.
- Behavior : A service defined in a class and provided by an object.

- **Operation** : The implementation of a behavior in an object-oriented programming language.
- **Modifier** : Modifier of an operation or attribute defines the level of access that objects have to it.
- **Multiplicity/ Cardinality** : The minimum and maximum number of objects that participate in an association or aggregation. The common ones are 0..*, 0..1, 1..*, and 1..1.
- **Role Name** : It is a name describing the participation of the class in the association more exactly.

2.1.1.2 Sequence Diagram

A sequence diagram depicts the sequence of actions that occur in a software. The invocation of methods in each object, and the order in which the invocation occurs are captured in a sequence diagram. It is a very useful diagram to easily represent the dynamic behavior of a software. Typically, a sequence diagram describes the detailed implementation of a single use case (or one variation of a single use case). A sequence diagram is two-dimensional in nature. The horizontal axis shows the life of the represented object, while the vertical axis shows the sequence of the creation or invocation of these objects. The UML notation for sequence diagram is shown below.

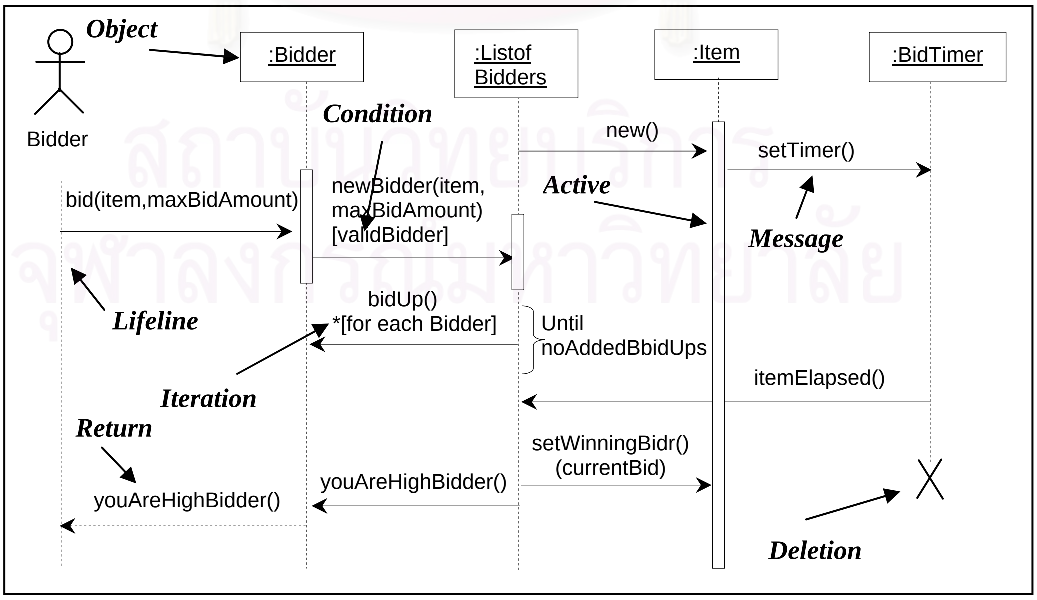


Figure 2.3: Notation for sequence diagram.

- **Object** : Each of the objects that participate in the processing represented in the sequence diagram is drawn across the top.
- **Lifeline:** A dotted line is dropped from each object in the sequence diagram. Arrows terminating on the lifeline indicate messages (commands) sent to the object. Arrows originating on the lifeline indicate messages sent from this object to another object. Time flows from top to bottom on a sequence diagram.
- **Active** : To indicate that an object is executing, i.e., it has control of the CPU, the lifeline is drawn as a thin rectangle.
- **Message** : A horizontal arrow represents a message (command) sent from one object to another.
- **Return** : When one object commands another, a value is often returned. This may be a value computed by the object as a result of the command or a return code indicating whether the object completed processing the command successfully.
- **Condition** : Square brackets are used to indicate a condition, i.e., a Boolean expression. The message is sent only if the expression is TRUE.
- **Iteration** : Square brackets proceeded by an asterisk (*) indicate iteration. The message is sent multiple times. The expression within the brackets describes the iteration rule.
- **Deletion** : An X is used to indicate the termination (deletion) of an object.

2.1.2 Software Measurement

Measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules [10]. In the assessment process prescribed by ISO-9126 [11], the goals of measurement must first be defined, then the measurement itself is specified, the means of measurement are implemented and the measurement is carried out. In the final step, the measurement results are evaluated.

Software metrics have been classified by Fenton [10] into three classes.

- **Process metrics** are used to measure characteristics of software processes such as development process, maintenance process and testing process. Typical process characteristics are effort involved, costs occurred, tasks accomplished and elapsed time.
- **Product metrics** are used to measure characteristics of software products such as programs, components, system and databases. Typical product characteristics are size, complexity and various qualities.
- **Resource metrics** are used to measure characteristics of software resources which may be hardware, software or person. Typical resource characteristics are performance, availability, reliability and productivity.

Fenton distinguishes further between internal and external characteristics.

- **Internal attributes** of a product, process or resource are those which can be measured purely in terms of the product, process or resource themselves. Internal attributes of software products are, for example, complexity, modularity, testability and reusability. They can be measured by examining source code themselves.
- **External attributes** are those which can only be measured with respect to how the product, process or resource relates to their environment. External attributes of software products are, for example, reliability, security, usability and performance. They can only be measured by testing the product in a particular environment.

2.1.3 MANOVA

MANOVA is simply an ANOVA with several dependent variables [12]. This is a test of overall relationship between groups and predictors by considering variance in the set of predictors that effects on group classification. MANOVA follows the model of ANOVA where variation in samples is partitioned into between-group variation and within-group variation. In MANOVA, however, each sample has a score on each of several independent variables. When several independent variables of each sample are measured, there is a matrix of scores (subjects by independent variables) rather than a simple set of independents within each group. Matrices of different scores are formed by subtracting from each score and appropriate mean. Then the matrix of differences is squared. When the squared differences are summed, a sum-of-squares-and-cross-products matrix, SS matrix, is formed, analogous to a sum of squares in ANOVA. Determinants of the various SS matrices are found, and ratios between them provide tests of hypotheses about the effects of the dependent variables on the linear combination of independent variables.

The between-group sums of squares and cross-products matrix can be shown as below. It is denoted by SS_{bg}.

$$SS_{bg} = \begin{pmatrix} SS_{b;1} & SP_{b;1,2} & SP_{b;1,3} & \ldots\ldots & SP_{b;1,p} \\ SP_{b;1,2} & SS_{b;2} & SPS_{b;2,3} & \ldots\ldots & SP_{b;2,p} \\ SP_{b;1,3} & SP_{b;2,3} & SS_{b;3} & \ldots\ldots & SP_{b;3,p} \\ : & & & & \\ SP_{b;1,p} & SP_{b;2,p} & SP_{b;3,p} & \ldots\ldots & SS_{b;p} \end{pmatrix}$$

where

$$SP_{b;r,s} = \sum n_j(\overline{X}_{j,r} - \overline{\overline{X}}_r)(\overline{X}_{j,s} - \overline{\overline{X}}_s) = \sum(T_{j,r}T_{j,s}/n_j) - T_rT_s/N$$

and

$$SS_{b;r} = SP_{b;r,r}\sum n_j(\overline{X}_{j,r} - \overline{\overline{X}}_r)^2 = \sum(T_{j,r}^2/n_j) - T_r^2/N$$

where $T_{j,r}$ is the sum of observations on independent variable r in group j and T_r is the sum of all observations in all groups on independent variable r.

The within-group covariance matrix can be shown as below. It is denoted by SS_{wg}.

$$SS_{wg} = \begin{pmatrix} SP_{w;1,1} & SP_{w;1,2} & SP_{w;1,3} & \ldots\ldots & SP_{w;1,p} \\ SP_{w;1,2} & SS_{w;2,2} & SPS_{w;2,3} & \ldots\ldots & SP_{w;2,p} \\ SP_{w;1,3} & SP_{w;2,3} & SS_{w;3,3} & \ldots\ldots & SP_{w;3,p} \\ : & & & & \\ SP_{w;1,p} & SP_{w;2,p} & SP_{w;3,p} & \ldots\ldots & SS_{w;p,p} \end{pmatrix}$$

where

$$SPw;r,s=\sum\sum(X_{j,r}-\overline{X}_{r})(X_{j,s}-\overline{X}_{s})=\sum\sum X_{r}X_{s}-\sum(T_{j,r}T_{j,s}/n_{j})$$

In ANOVA, ratios of variances are formed to test main effects and interactions. In MANOVA, ratios of determinants are formed to test main effects and interactions when using *Wilks' lambda*. These ratios follow the general form

$$\Lambda=\frac{\left|SS_{wg}\right|}{\left|SS_{bg}+SS_{wg}\right|}$$

An approximation to F has been derived that closely fits Λ. The following procedure for calculating approximate F is based on Wilks' lambda and the various degrees of freedom associated with it.

$$\text{Approximate } F(df_1,df_2)=\left(\frac{1-y}{y}\right)\left(\frac{df_2}{df_1}\right)$$

where df_1 and df_2 are defined below as the degrees of freedom for testing the F ratio, and y is

$$y=\Lambda^{1/s}$$

$$s=\sqrt{\frac{p^2(df_{bg})^2-4}{p^2+(df_{bg})^2-5}}$$

$$df_1=p(df_{bg})$$

$$df_2=s\left[(df_{wg})-\frac{p-df_{bg}+1}{2}\right]-\left[\frac{p(df_{bg})-2}{2}\right]$$

where p is the number of independent variables,

df_{bg} is the number of groups minus one or $k-1$,

df_{wg} is the number of groups times the quantity $n-1$, where n is the number of cases per group. Because n is often not equal for all groups, an alternative equation for df_{bg} is $N-k$, where N is the total number of cases in all groups.

If obtained F exceed critical F, we can conclude that groups of sample can be distinguished on the basis of the combination of the independent variables or predictors.

2.1.4 Classification Techniques

In this research, prediction models are constructed by applying three techniques: Discriminant analysis, Decision tree and MLP neural network.

2.1.4.1 Discriminant Analysis

Discriminant analysis is multivariate technique concerned with separating distinct groups of object (or observations) and with allocating new objects to previously defined groups [13]. With deriving a variation, the linear combination of the two or more independent variables will discriminate best between defined groups. Linear combination for discriminant analysis is also known as discriminant function.

This thesis applies a technique which uses Fisher's linear discriminant function. To assign cases into groups, a classification function is developed for each group. Data for each case are inserted into each classification function to develop a classification score for each group for the case. The case is assigned to the group for which it has the highest classification score.

In its simplest form, the basic classification equation for the $j\,\mathrm{th}$ group $(j = 1,2,..,k)$ is

$$S_j = c_{j0} + c_{j1}X_1 + c_{j2}X_2 + ... + c_{jp}X_p$$

where S_j is the classification score for group j,

X_i is the independent variable i,

c_{ji} is the classification coefficient for independent variable i of group j,

c_{j0} is a constant.

The row matrix of classification coefficients for group j $C_j = c_{j1}, c_{j2}, ..., c_{jp}$ is found by multiplying the inverse of the within-group variance-covariance matrix $\left(W^{-1}\right)$ by a column matrix of means for group j on the p independent variables (M_j= $\overline{X}_{j1}$, $\overline{X}_{j2}$,....., $\overline{X}_{jp}$). In matrix form,

$$C_j = W^{-1}M_j$$

The within-group covariance matrix (W) is produced by dividing each element in the cross-products matrix of differences within groups by the within-group degrees of freedom $(N-k)$ as the follows:

$$W = S_{wg} / (N - k)$$

where N is the total number of cases in all groups and k is the number of groups. The constant for group j , c_{jo} , is found as follows:

$$c_{jo} = \left(-\frac{1}{2}\right) C_j M_j$$

2.1.4.2 Decision Tree

Decision tree classifies objects by sorting them down the tree from the root to some leaf node, which provides the classification of the object [14]. Each node in the tree specifies a test of some attribute of the object, and each branch descending from the node corresponds to one of the possible values for this attribute. An object is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute in the given example. This process is then repeated for the subtree rooted at the new node.

The basic algorithm for decision tree is a greedy algorithm that constructs decision trees in a top-down recursive divide-and-conquer manner. The algorithm is a version of ID3, a well-known decision tree algorithm. The basic strategy is as follows.

- The tree starts as a single node representing the training samples.
- If the samples are all of the same class, then the node becomes a leaf and is labeled with that class.
- Otherwise, the algorithm uses an entropy-based measure known as information gain as a heuristic for selecting the attribute that will best separate the samples into individual classes. This attribute becomes the 'test' or 'decision' attribute at the node. In this version of the algorithm, all attribute are categorical, that is, discrete-valued. Continuous-valued attributes must be discretized.
- A branch is created for each known value of the test attribute, and the samples are partitioned accordingly.
- The algorithm uses the same process recursively to form a decision tree for the samples at each partition.

- The recursive partitioning stops only when any one of the following conditions is true:
  - All samples for a given node belong to the same class, or
  - There are no remaining attributes on which the samples may be further partitioned. In this case, majority voting is employed. This involves converting the given node into a leaf and labeling it with the class in majority among samples. Alternatively, the class distribution of the node samples may be stored.
  - There are no samples for the branch test-attribute = a_i. In this case, a leaf is created with the majority class in samples.

The *information gain* measure is used to select the test attribute at each node in the tree. Such a measure is referred to as an *attribute selection measure* or a *measure of the goodness of split*. The attribute with the highest information gain (or greatest *entropy* reduction) is chosen as the test attribute for the current node. This attribute minimizes the information needed to classify the samples in the resulting partitions and reflects the least randomness or 'impurity' in these partitions. Such an information-theoretic approach minimizes the expected number of tests needed to classify an object and guarantees that a simple (but not necessarily the simplest) tree is found.

Let S be a set consisting of s data samples. Suppose the class label attribute has m distinct values defining m distinct classes, C_i *(for i=1,…,m)*. Let s_i be the number of samples of S in class C_i. The expected information needed to classify a given sample is given by

$$I(s_1, s_2, \ldots, s_m) = -\sum_{i=1}^{m} p_i \log_2(p_i),$$

where p_i is the probability that an arbitrary sample belongs to class C_i and is estimated by s_i / s. Note that a log function to the base 2 is used since the information is encoded in bits.

Let attribute A have v distinct values, $(a_1, a_2, \ldots, a_v)$. Attribute A can be used to partition S into v subsets, $\{S_1, S_2, \ldots, S_v\}$, where S_j contains those samples in S that have value a_j of A. If A were selected as the test attribute (i.e., the best attribute for splitting), then these subsets would correspond to the branches grown from the node containing the set S. Let s_{ij} be the number of samples of class C_i in a subset S_j. The entropy, or expected information based on the partitioning into subsets by A, is given by

$$E(A)=\sum_{j=1}^{v}\frac{s_{1j}+...+s_{mj}}{s}I\left(s_{1j},...,s_{mj}\right).$$

The term $\frac{s_{1j}+...+s_{mj}}{s}$ acts as the weight of the $j\text{th}$ subset and is the number of samples in the subset (i.e., having value a_j of A) divided by the total number of samples in S. The smaller the entropy value, the greater the purity of the subset partitions. Note that for a given subset S_j,

$$I\left(s_{1j},s_{2j},...,s_{mj}\right)=-\sum_{i=1}^{m}p_{ij}\log_2\left(p_{ij}\right)$$

where $p_{ij}=\frac{s_{ij}}{\left|s_j\right|}$ and is the probability that a sample in S_j belongs to class C_i.

The encoding information that would be gained by branching on A is

$$Gain(A)=I\left(s_1,s_2,...,s_m\right)-E(A).$$

In the other words, $Gain(A)$ is the expected reduction in entropy caused by knowing the value of attribute A.

The algorithm computes the information gain of each attribute. The attribute with the highest information gain is chosen as the test attribute for the given set S. A node is created and labeled with the attribute, branches are created for each value of the attribute, and the samples are partitioned accordingly.

2.1.4.3 Multilayer Perceptron and Backpropagation Learning

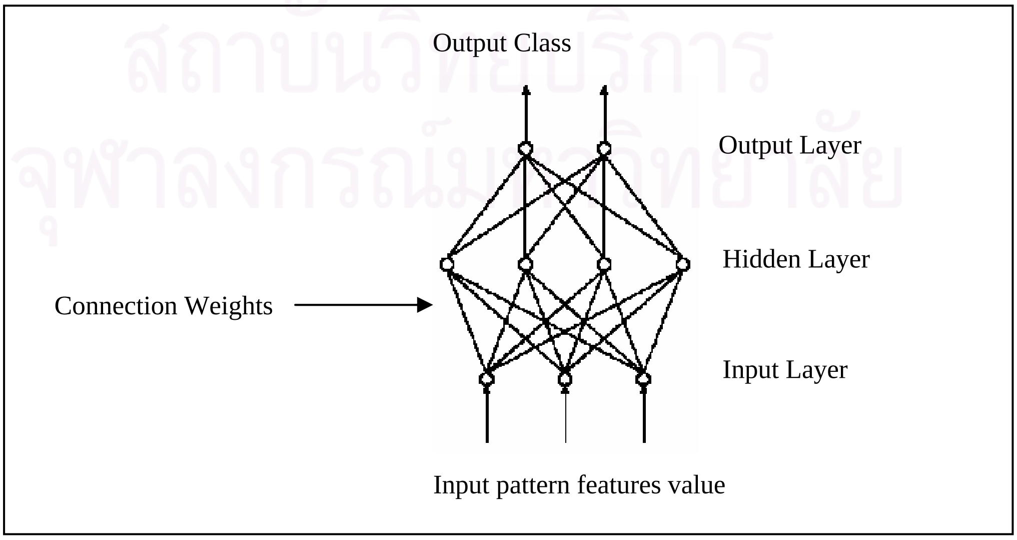


Figure 2.4: Multilayer perceptron neural network.

Multilayer perceptron (MLP) neural network is multilayer feed-forward network as shown in Figure 2.4 [15]. The network consists of a set of sensory units that constitute the input layer, one or more hidden layers of computation nodes, and output layer of computation nodes. The input signal propagates through the network in a forward direction on a layer-by-layer basis. This network uses backpropagation learning algorithm. The algorithm consists of two phases, namely, learning phase and working phase. In learning phase, a set of input patterns (training set) are presented to the input layer together with their corresponding desired output patterns.

A small random initial weight value is assigned to each connection between nodes in the input layer and the hidden layer. As each input pattern is applied, the actual output is recorded. The weights between the output layer and the previous layer (hidden layer) are recalculated using the generalized delta rule. The adjustment reduces the difference between the network actual outputs and the desired outputs for a given input pattern.

The input to each node for successive layers is the sum of the scalar products of the incoming vector components with their respective weights. Thus the input to a node j is given in the following equation.

$$input_j = \sum_i w_{ji} out_i$$

where w_{ji} is the weight connecting node i to node j and out_i is the output of node i. No calculation is performed at the input layer since it is just feeding the value of input patterns to the network. The output of a node j is, therefore,

$$out_j = f\left(input_j\right)$$

and this output is sent to all nodes in subsequent layer. This computation is continued through all layers of the network until the output layer is reached. At that point, the output vector is generated.

Based on the different error term or δ term in the output layer, the weight can be computed by the following equation.

$$w_{kj}(n+1) = w_{kj}(n) + \eta\left(\delta_k out_k\right)$$

where $w_{kj}(n+1)$ and $w_{kj}(n)$ are the weights connecting nodes k and j at iteration $(n+1)$ and n, respectively, η is a learning rate parameter. The δ terms of hidden layer nodes are computed and the weights connecting the hidden layer to the previous layer (another hidden layer or input layer) are updated accordingly.

The δ term in previous equation is often referred to as the rate of change of error with respect to the input of node k , and can be written as $\delta_k = (d_k - out_k) f(input_k)$ for nodes in the output layer, and $\delta_j = f(input_k) \sum_k \delta_k w_{kj}$ for nodes in the hidden layers, where d_k is the desired output of node k .

This calculation is repeated until

- all $w_{ji}(n+1) - w_{ji}(n)$ in the previous epoch (iteration) were so small as to be below some specified threshold, or
- the percentage of samples misclassified in the previous epoch is below some threshold, or
- a prespecified number of epochs has expired.

2.2 Literature Review

This section reviews UML metrics, some controlled experiments carried out in order to investigate maintainability of object-oriented software. Aesthetic criteria of class and sequence diagrams are also reviewed in this section.

2.2.1 UML Metrics

UML metrics were proposed by many researchers. Kim and Boldyreff proposed 27 software metrics that can be applied to UML modeling elements [6]. Metrics proposed in this paper consist of

- Metrics for Model : Model metrics are for estimating the size or the amount of information contained in a model. Model metrics are, for example, Number of the packages in a model (NPM) and Number of classes in a model (NCM).
- Metrics for Class : Class metrics concern with various characteristics of a class such as attribute, relationship and object instantiation. Class metrics are, for example, Number of the attributes in a class-unweighted (NATC1) and Number of the operations in a class-unweighted (NOPC1).

- **Metrics for Message** : This paper proposes message metrics in order to measure the degree of interactions. An example of message metrics is Number of the directly dispatched messages of a message (NDM).
- **Metrics for Use Case** : Use case captures contracts among the stakeholders about their behavior. The stakeholders are also called primary actors of a system. Use case gathers the different sequences of behavior or scenarios together. An example of use case metrics is Number of actors associated with a use case (NAU).

Genero and her colleagues introduced and analyzed a set of an existent object - oriented metrics that can be applied for assessing class diagrams complexity at the initial phases of the object oriented development life cycle [5]. They introduced and analyzed existing metrics as follows.

- **Chidamber and Kemerer's metrics** : Chidamber and Kemerer proposed a set of six object-oriented design metrics which were well-known in the field of object-oriented metrics.
- **Lorenz and Kidd's metrics** : Lorenz and Kidd proposed a group of metrics called 'design metrics', which dealt with the static characteristics of software design. They categorized their metrics into class size metrics, class inheritance metrics and class internal metrics.
- **Brito e Abreu and Melo's metrics** : Brito e Abreu and Melo proposed the MOOD (Metrics for Object Oriented Design) set of metrics. These metrics allowed the measurement of the main mechanisms of the OO paradigm, such as, encapsulation, inheritance, polymorphism and message passing.
- **Marchesi´s metrics** : Marchesi proposed a set of metrics to measure UML class diagrams at the analysis phase. These metrics consisted of metrics for single classes, metrics for packages and metrics for system.

Genero and her research team also proposed metrics for statechart diagram in [7]. Sheldon et al. suggested new metrics for measuring maintainability of a class inheritance hierarchy [8]. The proposed metrics were categorized into metric of understandability and metric of modifiability.

2.2.2 Maintainability of Object-Oriented Software

Controlled experiments carried out for investigating maintainability of object-oriented software were presented in many papers. Briand et al. presented a controlled experiment focusing on comparison of the maintainability of object-oriented and structured design document [16]. This experiment concentrated solely on the investigation of the use of quality design principal and their influence on a developer's ability to understand and modify design documents. Results strongly suggested that 'good' object-oriented design is easier to understand and modify than 'bad' object-oriented design. However, there was no strong evidence regarding the alleged higher maintainability of object-oriented design documents over structured design documents. Their next experiment showed that the system designed according to Coad and Yourdon's object-oriented design principle is significantly easier to maintain [17]. Deligiannis et al. presented a controlled experiment which investigated the impact of a single design heuristic, dealing with the 'god class' problem, on understandability and maintainability of object-oriented designs [18]. The result of this experiment accepted the hypothesis that it is easier to maintain a heuristic compliant object-oriented design than a heuristic non-compliant object-oriented design. They also proposed the new metrics to quantify the basis criteria for object-oriented design quality assessment: completeness, correctness, and consistency.

The effect of inheritance on the maintainability of object-oriented software was investigated by Daly and Brook [19]. Subjects of their empirical study were asked to modify object-oriented software with a hierarchy of 3 levels of inheritance depth and equivalent object-oriented software with no inheritance called 'flat' version. The collected timing data showed that maintaining object-oriented software with inheritance, on average, approximately 20% quicker than maintaining flat version. While Harrison and Counsell reported that it is not clear that system using inheritance will necessarily be more maintainable than those that do not [20]. The data analyzed from two out of five systems of their experiment suggested that deeper inheritance trees are attributes of systems which are harder to understand and maintain. Some of empirical studies of object-oriented metrics for maintenances and its sub-characteristic were summarized in Table 2.2.

Table 2.2: Some of empirical studies of object-oriented metrics for maintenances and its sub-characteristic.

Studies	Dependent variables	Independent variables	Technique	Object System
BS[21]	Maintenance effort	Number of clients, Fan-in, Simple Class Coupling, RFC, Fan-out, WMC, CHNL, NCIM, NOD, NOC, CDM	Spearman rank correlation	A patient collaborative care system
WD[22]	Maintenance time	DIT	Standard significant testing, Wilcoxon ran sum test	Six systems developed in C++
HC[23]	Modifiability, understandability	DIT	Chi-square analysis	Two systems, each with two versions
BB[17]	Understanding, modification	Coad and Yourdon quality design principles(Coupling, Cohesion, Clarity of design, Generalization-Specialization depth): DIT, NOC, and CBO	2x2 factorial Kruskal-Wallis test	Replication package
FN[24]	Adaptive maintenance effort	NCL, NRC, TNM, TLOC, MCC, MNA, MNM	Multilinear regression analysis	Music object-oriented distributed system coded in C++
PU[25]	The completeness and correctness of the modifications	The number of the levels in the inheritance structure	ANOVA, parametric t-test	Four models of a hotel administration, written in MERODE
GJ[26]	Understandability, analyzability, modifiability	NC, NA, NM, NAssoc, NAgg, NAggH, NDep, NGen, NGenH, MaxHAgg, MaxDIT	Fuzzy classification, regression trees, Spearman rank correlation, Kolmogrov-smirnov test, Principal Component Analysis	Twenty-eight UML class diagrams related to Bank Information Systems, nine different UML class of diagrams
BV[4]	Maintenance Time	Interaction Level(IL), Interface Size(IS), and Operation Argument Complexity(OAC)	ANOVA, correlation, single and multiple regression analysis	Quadrilateral, tractor-trailer

2.2.3 Aesthetic Criteria of Class and Sequence Diagrams

As a modeling standard, the UML does not say anything on how to produce readable programs. Especially when larger diagrams are shared, an agreement on aesthetics has to be made in order to produce the cost of communication and to minimize misunderstanding resulted from drawing the same diagram in many different ways.

Purchase and his colleagues presented an empirical study attempting to identify the most important aesthetics for class diagrams from human comprehension point of view [27]. Aesthetic criteria considered in their work were Minimize bends, Node distribution, Edge variation, Direction of flow, Orthogonality, Edge lengths and Symmetry. In [28], they carried out an experiment aiming to determine which variant of each of five notations used in class diagrams is the more suitable with respect to human performance. Five notations comprised of Inheritance direction, Inheritance arcs, Association representation, Association names and Cardinalities. They also performed preference experiment by assessing the effect of individual aesthetic in the application domain of class and collaboration diagrams [29]. The results showed that two most preference aesthetic criterion for class diagram were fewer crosses and fewer bends. Two most preference aesthetic criteria for collaboration diagrams were fewer crosses and no adjacent arrows. Eichelberger proposed some aesthetic criterion that reflect the highly sophisticated structural and semantic features of class diagrams in [30]. In this work, class diagram was described in terms of graph theory. It is obvious that classes, packages, rhombs representing n-ary association and ovals in pattern notations map to nodes. Associations, dependencies as well as inheritance relations map to edges. So, in his paper, properties which describe the graph and its embedding were given in terms of nodes and edges. Some examples of class diagram aesthetic criteria suggested in this paper are listed as follows.

- Edges should not overlap nodes.
- Edges should have not too much bends.
- Nodes on the same hierarchy level should have the same vertical or horizontal coordinate.

Gutwenger et al. suggested a new approach for visualizing class diagrams leading to a balanced mixture of the following aesthetic criteria : crossing minimization, bend minimization, uniform direction within each class hierarchy, no nesting of one class hierarchy within another, orthogonal layout, merging of multiple inheritance edges, and good edge labeling [31]. The traditional graph drawing aesthetic criteria and new aesthetic criteria applicable to sequence diagrams were proposed by Paranen et al. [32].

CHAPTER III

CONSTRUCTING MAINTAINABILITY MODELS

This chapter presents a controlled experiment carried out in order to construct maintainability models from structural complexity and aesthetic metrics. Figure 3.1 shows activity diagram of this work. The detail of each activity will be described in later sections.

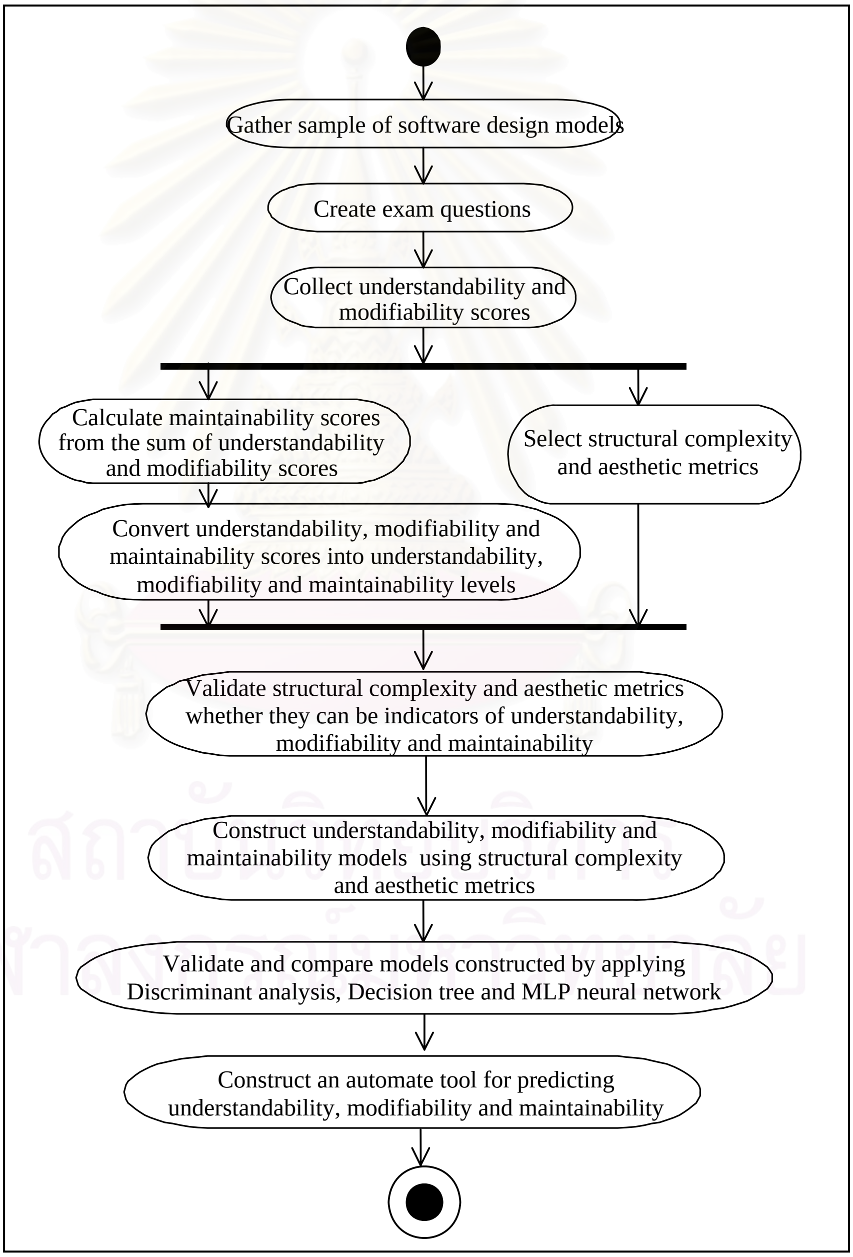

Gather sample of software design models
Create exam questions
Collect understandability and modifiability scores
Calculate maintainability scores from the sum of understandability and modifiability scores
Select structural complexity and aesthetic metrics
Convert understandability, modifiability and maintainability scores into understandability, modifiability and maintainability levels
Validate structural complexity and aesthetic metrics whether they can be indicators of understandability, modifiability and maintainability
Construct understandability, modifiability and maintainability models using structural complexity and aesthetic metrics
Validate and compare models constructed by applying Discriminant analysis, Decision tree and MLP neural network
Construct an automate tool for predicting understandability, modifiability and maintainability

Figure 3.1: Activity diagram of the research.

3.1 Metric Selection

An important step to analyze software quality using metrics is to identify a collection of metrics that reflect on software characteristics which are being analyzed. Metrics used in this work consist of structural complexity metrics and aesthetics metrics.

3.1.1 Structural Complexity Metrics

It is widely accepted that the greater complex software design model is, the greater complex finally implemented software is. Thus more effort is needed to develop and maintain it.

Table 3.1: Structural complexity metrics.

Metrics for class diagram				
Group		**Metric**		**Description**
Classes		NC	*	The total number of classes.
Attributes		ANAUW	**	The total number of attributes divided by the total number of classes.
		ANAW	**	It is the weighted version of ANAUW. Each attribute is weighted depending on its visibility, i.e. 1.0 for public, 0.5 for protected and 0.0 for private attributes.
Methods		ANMUW	**	The total number of methods divided by the total number of classes.
		ANMW	**	It is the weighted version of ANMUW. Each method is weighted depending on its visibility as same as weighting attribute in ANAW.
Relation-ships	Association	ANAssoc	**	The total number of association relationships divided by the total number of classes.
	Aggregation	ANAgg	**	The total number of association relationships divided by the total number of classes.
		NAggH	*	The number of aggregation hierarchies.
		MaxHAgg	*	It is the maximum among 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.
	Generalization	ANGen	**	The total number of generalization relationships divided by the total number of classes.
		NGenH	*	The number of generalization hierarchies.
		MaxDIT	*	It is the maximum among the DIT 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.
Metrics for sequence diagrams				
Scenarios		NOS		The total number of scenarios.
		WMBO		The total of average number of messages per instance objects in all scenarios divided by the total number of scenarios.
Messages		ANRM		The total number of return messages in all scenarios divided the total number of scenarios.
		ANDM	**	It is calculated from the total number of directly dispatched messages (NDM) of each message in all scenarios divided by the total number of scenarios. According to the UML semantics, a message can be an activator of other messages. For example, in Figure 3.2, the message a() activates the message c(), the NDM value of message a() is 1.
Conditions		ANCM		The total number of condition messages in all scenarios divided by the total number of scenarios.

* Metrics are proposed by Genero et al.[33].

** Metrics are modified from metrics proposed by Genero et al. [33] and Kim & Boldreff [6].

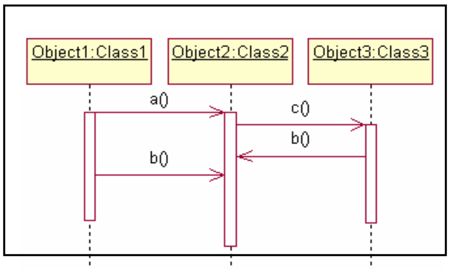


Figure 3.2: An example of sequence diagram.

Structural complexity metrics used in this work are shown in Table 3.1. Metrics for class diagram are categorized into metrics related to classes, attributes, methods and relationships. Metrics for sequence diagrams are categorized into metrics related to scenarios, messages and conditions. Existing metrics symbolized by '*' are proposed by Genero et al. [33]. Metrics symbolized by '**' are modified from the metrics proposed by Genero et al. [33] and Kim & Boldreff [6]. NOS, WMBO, ANRM and ANCM are four new proposed metrics.

Metrics symbolized by '*'; NC, NAggH, MaxHAgg, NGenH and MaxDIT; related to components in a class diagram including classes, aggregation relationship and generalization relationship, were used in an empirical study [33]. An experiment was carried out in order to validate that these metrics can be good indicators of class diagram maintainability. The analysis result obtained from technique of Fuzzy Deformable Prototype indicated that these metrics can be good indicators for classifying maintainability levels. In the same experiment, metrics related to methods and association relationship were validated. The results showed that they also could be good indicators of class diagram maintainability.

The followings are the reasons why metrics presented in Table 3.1 are chosen:

NC: This metric counts the number of classes in a class diagram. It is comparable to the traditional LOC (lines of code) or a more advance McCabe's cyclomatic complexity (MVG) metric for estimating the size of a software [6]. Thus, in object-oriented paradigm this metric can be used to compare sizes of software. The greater size, the more effort put to maintain.

ANAW and ANAUW: These metrics are the average number of attributes in each class with weight and un-weight respectively. If most of classes contain many attributes both metric values will be high. It captures the idea that a class containing more attributes makes maintenance more complex.

ANMW and ANMUW: These metrics are calculated from the average number of methods in each class with weight and un-weight respectively. If most of classes contain many methods, both metric values will be high. The number of methods in a class indicates the class size that related to the amount of collaboration being used. The more methods a class has, the more complex the class's interface. Hence, these metrics should be considered when analyzing maintainability.

ANAsso: This metric is the average number of association relationships. An association is a connection, or a link, between classes. This metric is useful for estimating the scale of relationships between classes. The higher number of associations renders more coupling between classes. So, this metric may affect maintainability.

ANAgg, NAggH and MaxHAgg: These metrics are aggregation metrics. An aggregation is a special form of association that specifies a whole-part relationship between the aggregate (whole) and component parts. Although the parts may exist independently of the whole, their existence is primarily to form the whole. The higher number of aggregations indicates more coupling between part and whole classes which makes software harder to maintain.

ANGen, NGenH and MaxDIT: These metrics are inheritance metrics. The higher the value of inheritance metrics, the greater the chance of reuse becomes. However, it can cause comprehension problem. Moreover, changing something in a superclass can affect the subclasses in a none desirable way. Hence inheritance can affect the maintainability of a software.

NOS: It is the total number of scenarios. A scenario represents a sequence of behavior of a software. More scenarios means more complexity of software behavior which affects maintainability.

WMBO and ANRM: These measures capture the property of how many messages call from and return to other classes. Messages are exchanged between objects manifesting various interactions. Since higher coupling leads to higher complexity, it will also make the class more difficult to modify and understand. So these metrics are essential when analyzing maintainability.

ANDM: It is calculated from the total number of directly dispatched messages of each message in all scenarios divided by the total number of scenarios. According to UML semantics, a message can be an activator of other messages. The more number of dispatched messages, the harder maintenance. Since, it is more difficult to trace with a lot of messages.

ANCM: It is the average number of condition messages in each scenario. A message with condition is easier to maintain than the one without condition. The more number of condition messages is, the more effort put to understand. Moreover, the conditions may be changed in the future, this leads to put more effort to modify them.

3.1.2 Aesthetic Metrics

An aesthetic criterion is a general graphical property of the layout that we would like to have. Drawing graph layout algorithms by conforming to aesthetic criteria is claimed that the resultant graph drawing is improved its readability. In this work, a set of aesthetic criteria is selected. Then a set of aesthetic metrics corresponding to the selected aesthetic criteria are defined. The aesthetic metrics consist of metrics for class diagram and metrics for sequence diagrams which are shown in Table 3.2.

In this table, the metrics corresponding to aesthetic criteria for traditional graph drawing that can be applied to class diagram are Cross, UnifEdgeLen, TotEdgeLen, MaxEdgeLen, UnifBends, TotBends, MaxBends, and Orthogonal [34,35]. Metrics namely Join, Center, Below, SameCo and Indicator are metrics corresponding to aesthetic criteria for class diagrams proposed by Purchase et al. [28,29] and Eichelberger et al. [30,37]. The term hierarchy in description of these metrics refers to the relations in class diagram including generalizations, aggregations and compositions.

Metrics for sequence diagrams are adapted from metrics proposed by Paranen et al. [32]. Figure 3.3 shows the example of drawings which conform and do not conform to some aesthetic criteria.

Table 3.2: Aesthetic metrics.

Metrics for Class Diagram		
Metric	**Description**	**Aesthetic Criterion**
Cross	The total number of edge crossings.	The total number of edge crossings should be minimized
UnifEdgeLen	The standard deviation of the edge length. (In this work, the edge length is measured in unit of centimeter.)	Edge lengths should be uniform.
TotEdgeLen	The total edge length.	The total edge length should be minimized.
MaxEdgeLen	The maximum edge length.	The maximum edge length should be minimized.
UnifBends	The standard deviation of the number of bends on the edges.	The standard deviation of the number of bends on the edges should be minimized.
TotBends	The total number of bends in the drawing.	The total number of bends should be minimized.
MaxBends	The maximum number of bends on the edges.	The maximum number of bends on the edges should be minimized.
Orthogonal	The total number of edges fixed to an orthogonal grid divided by the total number of edges.	Nodes and edges should be fixed to an orthogonal grid.
Join	The total number of joined hierarchies divided by the total number of hierarchies.	Generalizations, aggregations and compositions should be joined.
Center	The total number of hierarchies that the parent is located as the center of its children divided by the total number of hierarchies.	A parent node should be positioned as close as possible to the median position of its children.
Below	The total number of hierarchies that the parent is located above its children divided by the total number of hierarchies.	A child node should be positioned below its parent.
SameCo	The total number of hierarchies that the children nodes are located on the same vertical or horizontal coordinate divided by the total number of hierarchies.	Nodes on the same hierarchy level should have the same vertical or horizontal coordinate.
Indicator	The total number of edges representing association relationships that have clear label and directional indicator divided by the total number of edges representing association relationships.	Edge should be clearly labeled and should have directional indicator.
Metrics for Sequence Diagrams		
Metric	**Description**	**Aesthetic Criterion**
AvgCrossS	The average number of edge crossings. (The total number of edge crossings in all sequence diagrams divided by the total number of sequence diagrams.)	The total number of edge crossings should be minimized.
MaxEdgeLenS	The maximum edge length of all sequence diagrams.	The maximum edge length should be minimized.
AvgUnifEdgeLenS	The average standard deviation of edge length. (Summation of standard deviation of edge length of all sequence diagrams divided by the total number of sequence diagrams.)	Edge lengths should be uniform.
AvgSubsetSepS	The average number of distinct subsets of participants. (The total number of distinct subsets of participants of all sequence diagrams divided by the total number of sequence diagrams.)	The distinct subsets of participants should be maximized.

Many empirical studies were performed to determine which graph drawing aesthetics are important for human comprehension. An experimental study of Purchase and his colleagues revealed that increasing the number of crossings in a graph decreases the understandability of the graph [36]. The same group of researchers also presented an empirical study attempting to identify the most important aesthetics for class diagrams from human comprehension point of view [27]. Aesthetic criteria considered in their work were Edge bends, Edge lengths, Orthogonality and Direction of flow. The results suggested that these aesthetic criteria should be considered in order to produce a graph drawing that is easy to understand. The experimental result presented in [28] showed that joining inheritance arcs and placing a superclass above its subclasses are preferable for human understanding.

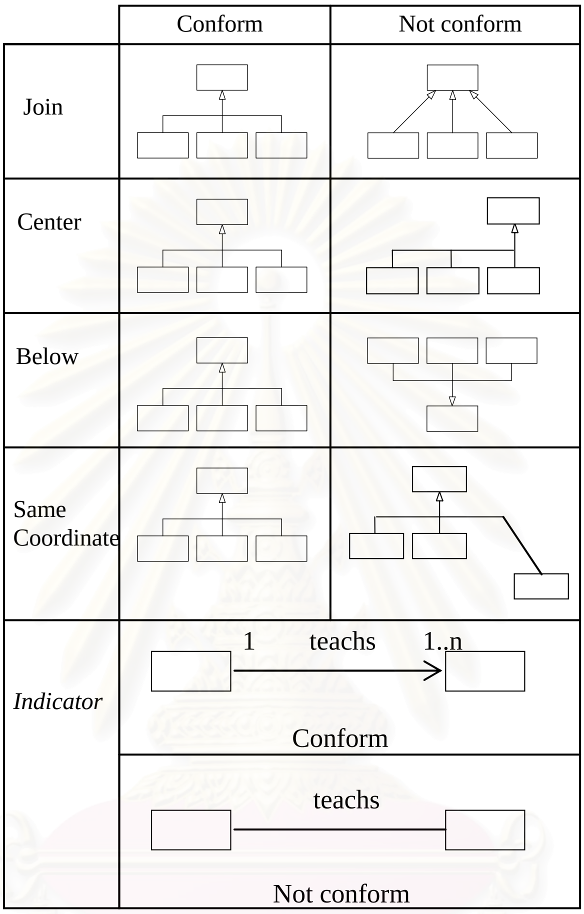


Figure 3.3: Example of the drawings which conform and do not conform to some aesthetic criteria.

A class diagram can be described in terms of graph theory. It is obvious that classes, packages and rhombs representing n-ary associations are mapped to nodes. Associations, aggregations as well as inheritance relationships are mapped to edges [37]. Aesthetic criteria of class diagram considered in this work are listed as follows.

Cross: Different edges should not overlap, this means that every edge should be visible and readable as an individual. In [36], the result of an empirical study of human understanding of graphs drawn using various aesthetic layouts shows that increasing the number of edge crossings in a graph decreases the understandability of the graph.

So, the number of edge crossings should be minimized to make edges more continuous and easier to follow.

UnifEdgeLen, TotEdgeLen, MaxEdgeLen: Edges should not be too long or too short because it is hard to make grouping and separation. Edge lengths should be uniform. Since it is difficult to follow long edges, the total edge length should be minimized and the maximum edge length should be minimized.

UnifBends, TotBends, MaxBends: The number of bends in any edge should be minimized to make edge more continuous and easier to follow. Therefore, the total number of bends in polyline edges should be minimized and the maximum number of bends on the edges should be minimized.

Orthogonal: Nodes and edges should be arranged to an orthogonal grid, i.e., maximize the number of orthogonal edges. Orthogonal drawing minimizes crossing between edges and the number of bends of the edges, and leads to neatness and readable layout.

Join: Inheritance relationships, aggregations and compositions should be joined as described in [30]. This admits a kind of orthogonal layout for hierarchical relationships. The experimental result presented in [28] showed that joined inheritance arcs are preferable to separated arcs. The two main reasons were; first, the joined inheritance notation demonstrates that the subclasses are on the same level of specialization. Second, in larger diagrams with more inheritance relationships, there is a potential for the diagram to "sprawl" and to look less "neat."

Center: Especially in hierarchy relationships, a parent node should be positioned as close as possible to the median position of its children. They should be placed as close as possible because they are closely related.

Below: A superclass should be placed above its subclasses and the inheritance arrows should be upwards because people usually are familiar with putting superior objects on top of other objects. The result of an empirical study showed that pointing the inheritance arrows upwards was preferred than pointing downwards, with the reason that it appears more natural to have a parent above its children [28]. This result revealed that most people would read from top-to-bottom, and it is important to identify the superclasses before the subclasses.

SameCoordinate: Nodes on the same hierarchy level should have the same vertical or horizontal coordinate, respectively, according to the way (top-down, left-right, e.g.) the hierarchy is drawn. This aesthetic makes the hierarchy easy to read.

Indicator: Edge should be clearly labeled and should have directional indicator. All text labels should be horizontal, rather than a mixture of horizontal and vertical, so that they can be read easily. For reasons of neatness and clarity, the label should be placed beside the edge, it should not be placed over the edge. The experimental result in [29] revealed that most subjects preferred having directional indicators associated with every labeled relationship, rather than not having the directional indicators at all. The reasons are "directional labels make edges more readable" and "clear, precise". Cardinalities should be defined explicitly for improvement of clarity and reduction of ambiguity. It is clearer to put both upper and lower bounds to avoid confusion.

This thesis considers four aesthetic metrics for a sequence diagram proposed by Poranen et al. [32] including Crossing, Maximum edge length, Uniform edge length and Subset separation. The first three metrics are aesthetic metrics of the traditional graph drawing which can be applied for a sequence diagram. For any software, it may use many sequence diagrams to visualize software behaviors. So, this work proposes the metrics for measuring all sequence diagrams described behaviors of the software by adapting metrics of Poranen et al. [32]. These metrics are measured in concept of average number (AvgCrossS, AvgUnifEdgeLenS, AvgSubsetSepS) and maximum number (MaxEdgeLenS). Aesthetic criteria of sequence diagrams considered in this work are listed as follows.

AvgCrossS: For sequence diagram, a message arrow can be viewed as an edge. It is easy to measure the number of edge crossings and edge length by noticing that each message whose length L increses the number of edge crossings by L-1 (the number of edge crossings are counted from the number of edge crossings with the lifelines.). When laying out sequence diagram, the number of edge crossings should be minimized.

MaxEdgeLenS: An aesthetic criterion for message arrows is to limit the maximum length of the arrows, or at least to decrease the number of the longer arrows.

It is corresponding to criterion for traditional graph drawing which is to minimize the maximum edge length.

AvgUnifEdgeLenS: Edge lengths should be uniform. In fact, this criterion often contradicts with the minimization of the total edge length. It might be impossible to shorten some of the edges. Then, minimizing the variance of the edge lengths would mean that the other edges should be made longer.

AvgSubsetSepS: Subset separation criterion plays an important role in visualizing a system having such subsets of participants that do not communicate to each other or do communicate very little. Suppose that there are two distinct sets of participants, and one participant, a filter, who receives and forwards all messages from/to those two sets. Now all communication between those two sets of participants go first to the filter participant which forwards messages to a participant in the second set. It is natural to place this filter participant in the middle of the diagram and then place the other two sets of participants to the left and to the right side. The goal of the subset separation property is to find out distinct subsets of participants that have as little communication as possible.

3.2 A Controlled Experiment

This section describes the detail of an experiment including of experimental aim and definitions, subjects, materials, tasks and data collection.

3.2.1 Experimental Aims and Definitions

The controlled experiment is carried out for two main objectives.

- To validate the structural complexity and aesthetic metrics whether they can be indicators of understandability, modifiability and maintainability.
- To construct understandability, modifiability and maintainability from structural complexity and aesthetic metrics.

In this work, understandability and modifiability are defined as follows.

- **Understandability** is the degree to which the software design model can provide its clear meaning to evaluator.

- **Modifiability** is the ease with which a change or changes can be made to the software design model.

3.2.2 Subjects

The experimental subjects used in this work were 60 graduate students from the Department of Computer Engineering at Chulalongkorn University, Bangkok, Thailand, who passed the classes on Software Requirements Engineering and Object-Oriented Technology. During lectures, students were taught basic software engineering principles and object-oriented development techniques. The lectures were supplemented by practical lessons where the students had the opportunity to design real-world object-oriented software using UML diagrams.

The information captured from the debriefing questionnaire based on the ordinal scale of 1 to 5 revealed that the subjects had medium experience with

- software engineering practice – median response 3 (min 2, max 4),
- design documents – median response 3 (min 2, max 4),
- modeling with UML – median response 3 (min 2, max 4) and
- software maintenance – median response 3 (min 2, max 4).

In order to control differences among students, the students were categorized into A, B+ and B by considering the grades they obtained from 2 classes mentioned above. Table 3.3 shows how to categorize the students. For example, if a student got A in Software Requirements Engineering and got B+ in Object-Oriented Technology, that student would be categorized into A. After that, they were randomly grouped into 20 teams of three students. Each team had one A, one B+ and one B students.

Table 3.3: Subject category.

Software requirements engineering	Object-Oriented Technology	Subject Category
A	A	A
A	B+	A
B+	A	A
B+	B+	B+
A	B	B+
B	A	B+
B+	B	B
B	B+	B
B	B	B

3.2.3 Experimental Materials and Tasks

Following Boehm model [10], this work focused on two sub-characteristics of maintainability: understandability and modifiability. Forty software design models, listed in Table 3.4, with different domains were used in this experiment. The documentation of each software design model included of the general software description, the class diagram, the sequence diagrams and a set of the examinations for assessing understandability and modifiability. An example of examination and examination validation are shown in Appendix B.

Table 3.4: Sample software.

No.	Software	Size (#Classes)	No.	Software	Size (#Classes)
1	Payment I	6	21	Advertisement	11
2	Seminar Registration	6	22	Tourist Agency	11
3	Discount System	6	23	Online Shop III	11
4	Online Shop I	7	24	Inventory II	12
5	Lift	7	25	Payment II	12
6	Material Management	7	26	Transportation	13
7	Course Registration	8	27	Online Book Shop	13
8	Drugstore	8	28	Export	13
9	Online Shop II	8	29	VDO Rental	14
10	Photo Gallery	8	30	Order System	14
11	Pet Shop	8	31	Hospital	15
12	Stock Exchange II	8	32	Pre-paid Mobile Phone	15
13	Banking	9	33	Library II	16
14	Stock Exchange I	9	34	VDO Shop	17
15	Online Movie Ticket Vending	9	35	Petrol Station	18
16	Car Rental	9	36	Online CD Shop	24
17	Restaurant	10	37	ATM	24
18	Multiplex Cinema	10	38	Investment	24
19	Library I	10	39	Garment	25
20	Inventory I	10	40	Calculator	36

The subjects were joined in a room. Two monitors explained to them how to carry out the tests. There were two tasks to be performed by the participants. First, each subject team was asked to complete the examinations of 2 software design models that were randomly assigned for the team. Assigning software design models to subject team is shown in Table 3.5. Each subject sat next to a subject who was examining the other documentation. This was performed to reduce plagiarism, although this was by no means a significant worry. Subjects were told verbally that there were different designs being worked upon, but were not told anything about the nature of the study, i.e., what hypotheses were being tested. During this time subjects were told not to talk among themselves, but to direct any questions they had to the two monitors.

Time out periods of the examination for assessing understandability and the examination for assessing modifiability were 30 minutes and 40 minutes respectively. These timeout periods were determined from a pilot test. There was 20-minute break between experimental tasks of 2 software design models. A pilot test was performed using four experienced subjects. The pilot test was conducted in order to find mistakes in the experimental procedure, to test that the experimental instructions are clear and to check tasks have reasonable complexity, but that they can be completed within the allotted time. No significant issues were encountered during the pilot test. The second task was to complete a debriefing questionnaire. This questionnaire captured personal information, experience, motivation, and subjective opinion of each subject. The questionnaire used in this work is shown in Appendix C.

Table 3.5: Assigning software to subject groups.

Subjects				Software No.	
Group No.	A	B+	B		
1	S1	S2	S3	14	33
2	S4	S5	S6	21	35
3	S7	S8	S9	39	31
4	S10	S11	S12	4	32
5	S13	S14	S15	5	36
6	S16	S17	S18	8	29
7	S19	S20	S21	17	26
8	S22	S23	S24	7	37
9	S25	S26	S27	13	18
10	S28	S29	S30	22	27
11	S31	S32	S33	9	23
12	S34	S35	S36	11	28
13	S37	S38	S39	12	30
14	S40	S41	S42	1	34
15	S43	S44	S45	6	19
16	S46	S47	S48	16	25
17	S49	S50	S51	2	24
18	S52	S53	S54	10	38
19	S55	S56	S57	3	40
20	S58	S59	S60	15	20

3.2.4 Data Collection

In many studies [4,17,38], maintainability had been operationalized in terms of understandability time (time required to understand) and modifiability time (time required to make changes). In this work, time for performing experimental tasks was restricted. So, maintainability was considered in terms of accuracy instead. For each software design model, data collected from the experiment can be listed as follows:

- **Understandability score** is quantified from the mean of 3 subjects' score of the examination for assessing understandability.
- **Modifiability score** is quantified from the mean of 3 subjects' score of the examination for assessing modifiability.
- **Maintainability score** is calculated from the sum of understandability score and modifiability score.
- **Understandability level** is captured by converting understandability score into 0, 1 or 2 which indicates understandability levels: difficult, medium, and easy respectively. Each understandability score can be converted using the following condition:

```
If Understandability score < Average value of understandability scores –
    W * Standard deviation value of understandability scores
Then Understandability level = 0
Else If  Understandability score > Average value of understandability scores +
     W * Standard deviation value of understandability scores
   Then Understandability level = 2
   Else Understandability level = 1
```

- **Modifiability level** is captured by converting modifiability score using the same approach of converting understandability score into understandability level.
- **Maintainability level** is captured by converting maintainability score using the same approach of converting understandability score into understandability level.

Kolmogorov-Smirnov is a statistical technique used to decide if a sample comes from a population with a specific distribution. The results of Kolmogorov-Smirnov test on understandability, modifiability and maintainability scores showed that understandability scores, modifiability scores and maintainability scores had normal distribution. Therefore, this approach could be considered valid. The approach can be summarized as shown in Figure 3.4.

W is a constant number. Its value is adjusted according to data distribution. In this experiment, for finding understandability and maintainability levels, value of W is 0.75. For finding modifiability level, value of W is 0.5.

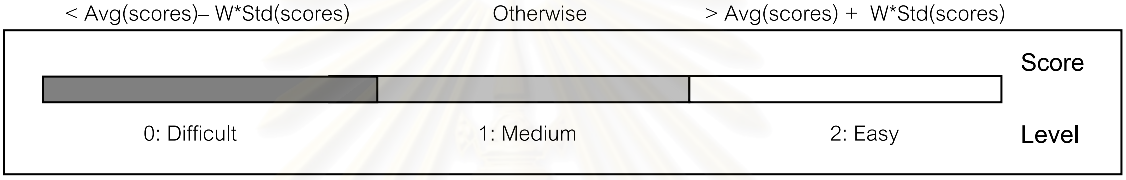


Figure 3.4: Converting understandability, modifiability and maintainability scores into understandability, modifiability and maintainability levels.

For each software design model, structural complexity and aesthetic metrics mentioned earlier in section 3.1 were measured. One discarded metric was Cross since its values obtained from sample class diagrams were not different (more than 80% were zero value). Therefore, it was useless for classifying understandability, modifiability and maintainability levels. Table 3.6 - Table 3.11 show all data collected from 40 sample software design models.

Table 3.6: Structural complexity metric values of sample software.

Software No.	NC	ANAUW	ANAW	ANMUW	ANMW	ANAssoc	ANAgg	NAggH	MaxHAgg	ANGen	NGenH	MaxDIT	NOS	WMBO	ANRM	ANDM	ANCM
1	6.000	3.000	0.167	6.333	5.167	0.167	0.167	1.000	1.000	0.500	1.000	2.000	1.000	2.000	3.000	1.000	1.000
2	6.000	2.143	0.000	2.429	2.286	0.571	0.143	1.000	1.000	0.000	0.000	0.000	2.000	.889	7.000	2.000	0.500
3	6.000	2.833	0.000	1.500	1.500	0.500	0.167	1.000	1.000	0.000	0.000	0.000	1.000	1.400	3.000	2.000	0.000
4	7.000	2.857	0.214	1.571	1.571	0.429	0.000	0.000	.000	0.286	1.000	1.000	2.000	1.325	2.500	2.500	0.000
5	7.000	1.625	0.063	1.500	1.500	0.375	0.375	2.000	1.000	0.250	1.000	1.000	2.000	1.800	0.000	4.000	0.000
6	7.000	2.857	0.000	2.143	2.143	3.429	0.143	1.000	1.000	0.286	1.000	2.000	3.000	1.667	1.000	1.000	0.333
7	8.000	2.500	0.125	4.625	4.625	0.750	0.250	2.000	1.000	0.000	0.000	0.000	3.000	1.600	1.333	2.000	1.333
8	8.000	3.000	0.000	2.000	2.000	0.750	0.250	2.000	1.000	0.000	0.000	0.000	3.000	1.111	0.000	1.333	0.667
9	8.000	1.875	0.186	3.000	3.000	0.625	0.250	1.000	1.000	0.250	1.000	1.000	1.000	2.167	2.000	7.000	1.000
10	8.000	3.125	1.125	3.500	3.500	0.500	0.000	0.000	.000	0.625	1.000	3.000	2.000	1.450	1.000	1.500	0.000
11	8.000	4.250	0.375	3.750	3.750	0.500	0.375	2.000	1.000	0.250	1.000	1.000	4.000	1.938	2.250	2.000	0.000
12	8.000	2.125	0.000	3.875	3.875	1.000	0.000	0.000	.000	0.000	0.000	0.000	3.000	2.022	2.333	2.667	3.000
13	9.000	2.222	0.333	3.667	3.667	0.556	0.222	2.000	1.000	0.333	2.000	1.000	5.000	1.250	0.200	0.800	1.400
14	9.000	0.889	0.333	3.222	3.222	0.333	0.333	3.000	1.000	0.556	2.000	1.000	3.000	1.111	3.333	1.333	1.667
15	9.000	1.778	0.000	5.333	5.333	0.111	0.667	1.000	2.000	0.000	0.000	0.000	3.000	4.944	1.333	2.000	0.667
16	9.000	2.889	0.222	2.000	2.000	0.667	0.111	1.000	1.000	0.222	2.000	1.000	3.000	1.250	0.667	1.333	0.667
17	10.000	2.000	0.000	3.100	3.100	0.800	0.100	1.000	1.000	0.000	0.000	0.000	3.000	1.250	1.000	2.333	0.667
18	10.000	1.200	0.100	2.900	2.900	0.400	0.200	1.000	2.000	0.200	1.000	1.000	4.000	1.075	1.000	2.750	1.000
19	10.000	1.800	0.300	3.000	3.000	0.400	0.000	0.000	.000	0.500	2.000	1.000	2.000	1.500	1.000	3.000	1.000
20	10.000	2.000	0.000	3.000	2.950	0.800	0.000	0.000	.000	0.300	1.000	1.000	3.000	3.111	1.667	1.000	1.000

Table 3.7: Structural complexity metric values of sample software (continued).

Software No.	NC	ANAUW	ANAW	ANMUW	ANMW	ANAssoc	ANAgg	NAggH	MaxHAgg	ANGen	NGenH	MaxDIT	NOS	WMBO	ANRM	ANDM	ANCM
21	11.000	2.727	0.409	4.364	4.364	0.364	0.364	1.000	2.000	0.364	2.000	1.000	2.000	1.875	0.000	2.500	0.000
22	11.000	4.545	0.000	1.364	1.364	0.909	0.272	3.000	1.000	0.000	0.000	0.000	2.000	1.571	0.500	2.000	1.000
23	11.000	2.636	0.000	3.091	3.091	1.000	0.455	4.000	1.000	0.000	0.000	0.000	2.000	1.875	0.500	2.500	0.500
24	12.000	1.438	0.219	2.813	2.813	0.188	0.188	2.000	2.000	0.188	2.000	1.000	4.000	2.458	0.000	2.250	0.250
25	12.000	1.583	0.000	3.667	3.667	0.250	0.250	1.000	2.000	0.000	0.000	0.000	3.000	1.806	0.667	0.667	0.333
26	13.000	2.000	0.077	2.462	2.462	0.846	0.154	2.000	1.000	0.308	1.000	1.000	3.000	2.111	0.000	3.333	1.333
27	13.000	1.846	0.385	2.767	2.462	0.385	0.462	2.000	2.000	0.385	2.000	1.000	3.000	1.833	2.000	2.000	0.000
28	13.000	1.769	0.154	1.615	1.615	0.923	0.077	1.000	1.000	0.538	2.000	1.000	4.000	0.875	3.000	1.750	0.500
29	14.000	2.143	0.250	1.429	1.357	0.286	0.286	2.000	2.000	0.429	2.000	1.000	2.000	1.750	2.000	3.500	0.000
30	14.000	0.947	0.000	2.947	2.947	0.368	0.211	4.000	1.000	0.000	0.000	0.000	5.000	1.933	0.400	1.800	0.000
31	15.000	2.067	0.333	1.000	1.000	0.600	0.333	3.000	1.000	0.467	2.000	2.000	3.000	1.417	0.000	1.000	0.333
32	15.000	2.833	0.305	1.556	1.556	0.444	0.056	1.000	1.000	0.389	2.000	2.000	4.000	1.558	1.250	2.750	0.750
33	16.000	1.500	0.125	1.500	1.500	0.250	0.000	0.000	0.000	0.625	3.000	1.000	4.000	1.000	1.250	0.750	0.750
34	17.000	3.353	0.559	2.824	2.824	0.353	0.176	2.000	1.000	0.471	2.000	2.000	3.000	1.733	1.667	3.000	1.000
35	18.000	1.611	0.000	1.167	1.167	0.833	0.166	2.000	1.000	0.000	0.000	0.000	2.000	1.000	0.000	2.000	0.000
36	24.000	0.917	0.104	1.000	1.000	0.583	0.208	3.000	1.000	0.333	3.000	1.000	2.000	0.833	1.000	1.500	0.000
37	24.000	0.792	0.042	1.250	1.250	0.917	0.458	2.000	2.000	0.167	1.000	1.000	4.000	1.363	1.500	1.250	0.500
38	24.000	2.125	0.042	2.792	2.792	0.417	0.250	4.000	1.000	0.083	1.000	1.000	3.000	2.156	0.000	2.000	0.333
39	25.000	1.560	0.220	1.400	1.400	0.480	0.280	4.000	1.000	0.360	3.000	1.000	2.000	1.000	1.000	3.000	1.000
40	36.000	1.556	0.361	3.611	3.542	0.444	0.000	0.000	0.000	0.667	4.000	2.000	2.000	1.807	0.000	10.500	0.000

Table 3.8: Aesthetic metric values of sample software.

Software No.	Unif EdgeLen	Tot EdgeLen	Max EdgeLen	Unif Bends	Tot Bends	Max Bends	Ortho-gonal	Join	Center	Below	SameCo	Indicator	Avg CrossS	Max EdgeLens	AvgUnif EdgeLenS	Avg SubsetSepS
1	0.635	12.500	3.000	1.200	4.000	2.000	1.000	1.000	1.000	0.750	1.000	0.000	3.000	2.000	0.220	2.000
2	4.673	16.300	7.000	0.000	0.000	0.000	0.400	1.000	1.000	1.000	1.000	1.000	15.000	5.000	1.090	1.000
3	0.387	9.600	3.300	0.000	0.000	0.000	0.000	1.000	1.000	1.000	1.000	1.000	6.000	3.000	0.710	2.000
4	0.906	17.800	3.700	1.143	6.000	2.000	1.000	1.000	1.000	1.000	1.000	0.750	4.500	3.000	0.380	2.000
5	5.898	32.900	9.300	0.786	6.000	2.000	1.000	0.667	1.000	0.800	0.667	1.000	2.000	2.000	0.190	3.000
6	6.018	69.800	9.700	0.555	9.000	2.000	0.231	1.000	1.000	1.000	1.000	4.500	1.667	2.000	0.280	1.333
7	0.468	14.700	2.800	0.000	0.000	0.000	0.625	1.000	1.000	1.000	1.000	1.000	3.000	4.000	0.470	2.000
8	6.174	24.800	8.900	0.000	0.000	0.000	0.625	1.000	1.000	0.500	1.000	1.000	0.667	2.000	0.110	2.333
9	10.041	41.300	10.900	0.778	5.000	2.000	0.778	0.500	1.000	1.000	0.500	0.600	11.000	4.000	0.780	2.000
10	1.678	25.700	5.100	1.143	8.000	2.000	0.875	1.000	1.000	1.000	1.000	0.667	6.000	4.000	0.710	1.500
11	1.133	20.700	4.600	0.952	4.000	2.000	1.000	1.000	1.000	1.000	1.000	1.000	1.750	3.000	0.290	1.750
12	6.440	40.300	9.100	0.722	5.000	2.000	0.700	1.000	1.000	1.000	1.000	0.286	9.000	4.000	0.590	1.333
13	7.645	33.500	10.900	0.711	4.000	2.000	1.000	1.000	1.000	0.600	1.000	1.000	0.400	2.000	0.090	1.600
14	6.703	42.200	10.100	0.000	0.000	0.000	0.364	0.600	0.800	0.750	0.800	1.000	2.667	3.000	0.410	2.333
15	2.155	22.500	6.100	0.143	1.000	1.000	0.571	0.333	0.667	0.833	0.667	1.000	6.000	6.000	0.820	1.000
16	2.091	22.800	5.100	0.857	4.000	2.000	0.875	1.000	1.000	1.000	1.000	0.167	2.000	2.000	0.260	1.667
17	1.847	22.700	5.000	0.000	0.000	0.000	0.778	1.000	1.000	1.000	1.000	1.000	1.667	3.000	0.350	2.667
18	4.438	26.500	6.900	0.444	2.000	0.000	0.667	1.000	1.000	1.000	1.000	1.000	0.750	2.000	0.140	2.750
19	0.572	25.800	3.900	1.111	10.000	2.000	1.000	1.000	1.000	1.000	1.000	0.000	1.500	2.000	0.100	3.500
20	0.920	29.900	4.600	0.873	6.000	2.000	0.818	1.000	1.000	1.000	1.000	1.000	0.333	2.000	0.070	2.333

Table 3.9: Aesthetic metric values of sample software (continued).

Software No.	Unif EdgeLen	Tot EdgeLen	Max EdgeLen	Unif Bends	Tot Bends	Max Bends	Ortho-gonal	Join	Center	Below	SameCo	Indicator	Avg CrossS	Max EdgeLens	AvgUnif EdgeLenS	Avg SubsetSepS
21	2.570	30.800	5.500	1.091	12.000	2.000	0.833	0.750	1.000	0.750	0.500	1.000	2.000	2.000	0.230	2.000
22	9.336	41.700	11.900	0.769	6.000	2.000	0.615	1.000	1.000	0.000	1.000	0.600	5.500	7.000	2.210	1.500
23	4.835	63.200	10.300	1.140	16.000	3.000	0.316	0.750	1.000	0.600	0.750	0.000	7.000	4.000	1.400	1.500
24	1.470	27.300	4.900	0.778	4.000	2.000	0.778	1.000	1.000	0.667	1.000	0.000	5.500	2.000	0.260	2.750
25	0.370	10.900	2.500	0.000	0.000	0.000	0.667	0.500	1.000	0.667	1.000	1.000	6.000	4.000	0.580	1.667
26	1.447	45.600	5.500	0.059	1.000	1.000	0.471	0.667	0.667	0.500	0.667	0.818	3.333	4.000	0.420	1.667
27	1.561	36.000	5.700	0.117	2.000	1.000	0.313	0.400	1.000	0.182	0.800	0.000	3.667	5.000	0.620	2.000
28	12.576	92.100	12.600	0.889	22.000	2.000	0.944	1.000	1.000	1.000	1.000	1.000	4.500	5.000	0.850	1.750
29	1.366	33.600	4.400	0.879	8.000	2.000	0.857	0.800	1.000	0.700	0.800	0.750	10.000	4.000	0.860	1.500
30	1.047	27.100	4.400	0.000	0.000	0.000	0.818	1.000	1.000	0.750	1.000	1.000	3.600	3.000	0.510	2.000
31	4.740	60.300	10.800	0.912	12.000	2.000	0.684	0.800	1.000	0.600	0.800	0.875	4.000	3.000	0.830	2.000
32	6.159	59.900	9.300	0.996	15.000	2.000	0.500	1.000	0.750	1.000	0.750	1.000	6.500	5.000	1.570	2.250
33	1.868	44.500	5.500	0.917	10.000	2.000	0.625	0.667	1.000	0.600	1.000	0.800	2.750	3.000	0.400	1.750
34	2.358	52.800	6.300	1.053	18.000	2.000	0.895	1.000	0.857	0.833	1.000	1.000	3.333	3.000	0.420	2.333
35	0.874	33.100	3.600	0.000	0.000	0.000	0.778	0.500	1.000	0.000	1.000	0.125	1.000	2.000	0.200	3.000
36	1.054	45.300	6.000	0.560	8.000	2.000	0.600	0.667	2.000	0.846	2.000	0.917	5.000	4.000	1.045	2.000
37	2.122	88.000	7.200	0.480	10.000	2.000	0.459	0.750	0.750	0.333	0.750	0.571	4.000	3.000	0.360	1.750
38	1.263	55.300	4.800	0.623	8.000	2.000	0.682	0.800	1.000	0.500	0.800	1.000	4.000	3.000	0.550	2.000
39	1.903	63.200	5.300	0.893	23.000	2.000	0.679	0.857	0.857	0.688	0.714	0.583	4.500	4.000	0.640	3.000
40	1.402	74.100	5.600	1.026	38.000	2.000	0.667	0.857	0.857	0.917	0.857	0.786	53.000	11.000	7.540	4.500

Table 3.10: Understandability, Modifiability and Maintainability Scores and Levels of sample software.

Software No.	Understandability Score	Modifiability Score	Maintainability Score	Understandability Level	Modifiability Level	Maintainability Level
1	15.33	15.33	30.67	2	2	2
2	13.00	8.67	21.67	1	0	0
3	14.00	14.00	28.00	1	2	2
4	17.33	10.67	28.00	2	1	2
5	12.67	11.00	23.67	1	1	1
6	13.50	12.00	25.50	1	1	1
7	14.00	15.67	29.67	1	2	2
8	13.00	16.00	29.00	1	2	2
9	13.33	11.00	24.33	1	1	1
10	11.33	12.33	23.67	0	1	1
11	15.00	14.67	29.67	2	2	2
12	14.33	13.67	28.00	2	2	2
13	14.00	12.33	26.33	1	1	1
14	12.67	13.33	26.00	1	1	1
15	16.00	10.33	26.33	2	1	1
16	16.00	14.33	30.33	2	2	2
17	16.33	14.00	30.33	2	2	2
18	14.00	10.33	24.33	1	1	1
19	11.33	12.00	23.33	0	1	1
20	13.00	11.00	24.00	1	1	1
21	8.33	9.00	17.33	0	0	0
22	13.67	12.00	25.67	1	1	1
23	15.00	13.83	28.83	2	2	2
24	8.00	8.67	16.67	0	0	0
25	15.33	12.67	28.00	2	1	2
26	15.33	11.00	26.33	2	1	1
27	13.33	12.33	25.67	1	1	1
28	13.00	12.33	25.33	1	1	1
29	11.33	12.00	23.33	0	1	1
30	13.00	12.33	25.33	1	1	1
31	13.33	6.67	20.00	1	0	0
32	13.00	10.67	23.67	1	1	1
33	11.33	9.00	20.33	0	0	0
34	11.33	11.00	22.33	0	1	1
35	12.67	11.67	24.33	1	1	1

Table 3.11: Understandability, Modifiability and Maintainability Scores and Levels of sample software (continued).

Software No.	Understandability Score	Modifiability Score	Maintainability Score	Understandability Level	Modifiability Level	Maintainability Level
36	9.00	8.67	17.67	0	0	0
37	10.33	11.00	21.33	0	1	0
38	11.33	11.67	23.00	0	1	1
39	12.67	7.33	20.00	1	0	0
40	9.00	8.33	17.33	0	0	0

3.3 Experimental Analysis and Results

This section presents metric validation and construction of maintainability models applying Discriminant analysis, Decision tree and MLP neural network.

3.3.1 Metric Validation

The main goal of the experiment is to analyze class and sequence diagrams for the purpose of validating the possibility of structural complexity and aesthetic metrics being used as good indicators of class and sequence diagrams maintainability. Therefore, the following hypotheses are proposed:

$H1_0$: Structural complexity metrics cannot be indicators for classifying understandability level.

$H1_A$: Structural complexity metrics can be indicators for classifying understandability level.

$H2_0$: Structural complexity metrics cannot be indicators for classifying modifiability level.

$H2_A$: Structural complexity metrics can be indicators for classifying modifiability level.

$H3_0$: Structural complexity metrics cannot be indicators for classifying maintainability level.

$H3_A$: Structural complexity metrics can be indicators for classifying maintainability level.

$H4_0$: Aesthetic metrics cannot be indicators for classifying understandability level.

$H4_A$: Aesthetic complexity design metrics can be indicators for classifying understandability level.

$H5_0$: Aesthetic metrics cannot be indicators for classifying modifiability level.

$H5_A$: Aesthetic metrics can be indicators for classifying modifiability level.

$H6_0$: Aesthetic metrics cannot be indicators for classifying maintainability level.

$H6_A$: Aesthetic metrics can be indicators for classifying maintainability level.

$H7_0$: Structural complexity and aesthetic metrics cannot be indicators for classifying understandability level.

$H7_A$: Structural complexity metrics and aesthetic can be indicators for classifying understandability level.

$H8_0$: Structural complexity and aesthetic metrics cannot be indicators for classifying modifiability level.

$H8_A$: Structural complexity and aesthetic metrics can be indicators for classifying modifiability level.

$H9_0$: Structural complexity and aesthetic metrics cannot be indicators for classifying maintainability level.

$H9_A$: Structural complexity and aesthetic metrics can be indicators for classifying maintainability level.

In order to test these hypotheses, Multivariate analysis of variance (MANOVA) is applied. MANOVA is simply an ANOVA with several dependent variables. This is a test of overall relationship between groups and predictors by considering variance in the set of predictors that effects on group classification. The results of MANOVA test are shown in Table 3.12 - Table 3.14. Because all obtained P-value are less than significant level

(0.001), all null hypotheses are rejected. This result can be concluded that three groups of understandability, modifiability and maintainability levels can be distinguished on the basis of structural complexity metrics, aesthetic metrics and combination of structural complexity and aesthetic metrics.

Table 3.12: MANOVA test of structural complexity metrics.

Hypothesis	Source of Variance	Wilks' Lambda	df1	df2	Multivariate F	P-value
H1	Understandability	0.006427	34	42	14.1734	<0.001
H2	Modifiability	0.005586	34	42	15.2927	<0.001
H3	Maintainability	0.005232	34	42	15.8427	<0.001

Table 3.13: MANOVA test of aesthetic metrics.

Hypothesis	Source of Variance	Wilks' Lambda	df1	df2	Multivariate F	P-value
H4	Understandability	0.008599	32	44	13.4529	<0.001
H5	Modifiability	0.006765	32	44	15.3424	<0.001
H6	Maintainability	0.006473	32	44	15.7153	<0.001

Table 3.14: MANOVA test of structural complexity and aesthetic metrics.

Hypothesis	Source of Variance	Wilks' Lambda	df1	df2	Multivariate F	P-value
H7	Understandability	0.009932	46	30	5.8919	<0.001
H8	Modifiability	0.006647	46	30	7.3471	<0.001
H9	Maintainability	0.005762	46	30	7.9395	<0.001

3.3.2 Constructing Understandability, Modifiability and Maintainability Models

This section presents how to construct understandability, modifiability and maintainability models applying 3 techniques: Discriminant analysis, Decision tree and MLP neural network.

In order to find the best prediction models for understandability, modifiability and maintainability, eighteen prediction models for each technique are constructed. So, the number of models constructed in this work is 54 models.

Prediction models constructed by applying Discriminant analysis consist of the following models.

DiscUnd1: Understandability model constructed from all structural complexity metrics.

DiscMod1: Modifiability model constructed from all structural complexity metrics.

DiscMain1: Maintainability model constructed from all structural complexity metrics.

DiscUnd2: Understandability model constructed from structural complexity metrics from which correlated metrics are discarded.

DiscMod2: Modifiability model constructed from structural complexity metrics from which correlated metrics are discarded.

DiscMain2: Maintainability model constructed from structural complexity metrics from which correlated metrics are discarded.

DiscUnd3: Understandability model constructed from all aesthetic metrics.

DiscMod3: Modifiability model constructed from all aesthetic metrics.

DiscMain3: Maintainability model constructed from all aesthetic metrics.

DiscUnd4: Understandability model constructed from aesthetic metrics from which correlated metrics are discarded.

DiscMod4: Modifiability model constructed from aesthetic metrics from which correlated metrics are discarded.

DiscMain4: Maintainability model constructed from aesthetic metrics from which correlated metrics are discarded.

DiscUnd5: Understandability model constructed from all structural complexity and aesthetic metrics.

DiscMod5: Modifiability model constructed from all structural complexity and aesthetic metrics.

DiscMain5: Maintainability model constructed from all structural complexity and aesthetic metrics.

DiscUnd6: Understandability model constructed from structural complexity and aesthetic metrics from which correlated metrics are discarded.

DiscMod6: Modifiability model constructed from structural complexity and aesthetic metrics from which correlated metrics are discarded.

DiscMain6: Maintainability model constructed from structural complexity and aesthetic metrics from which correlated metrics are discarded.

Prediction models constructed by applying Decision tree consist of DecUnd1 – DecMain6 which using same approaches of DiscUnd1 – DiscMain6. Then, 18 prediction models: MLPUnd1 – MLPMain6 are constructed using same approaches of DiscUnd1 – DiscMain6 but they are constructed by applying technique of MLP neural network.

3.3.2.1 Correlation Analysis

As mentioned in section 3.3.2, prediction models can be categorized into 2 groups.

1) Prediction models constructed from all structural complexity metrics or from all aesthetic metrics or from all structural complexity and aesthetic metrics which are listed in Table 3.1 and Table 3.2.
2) Prediction models constructed from structural complexity metrics from which correlated metrics are discarded or from aesthetic metrics from which correlated metrics are discarded or from structural complexity and aesthetic metrics from which correlated metrics are discarded.

For constructing the second group of prediction models, correlation analysis of metrics are applied. Correlation between each pair of metrics is considered in order to discard metrics that provide redundant information (i.e. the metric measures similar property as other metrics). This can be automated by applying Pearson's correlation test with significant level at 0.01. For each couple of highly correlated metrics, only one of them will be selected. Linear regression with one independent variable is performed for each metric. Then, adjusted R square value is used to determine the best choice. Adjusted R square value of independent variable indicates that it can explain the variance of dependent variable well or not. The metric with higher adjusted R square value (not consider sign) will be chosen. Results of correlation analysis are shown in Table 3.15 – Table 3.22.

Table 3.15: Pearson's correlation of structural complexity metrics.

Correlated Metrics	Pearson's Correlation	Adjusted R Square
ANAW & ANGen	0.678	0.01
ANAW & MaxDIT	0.729	0.01
ANMUW & ANMW	0.989	0.01
NAggH & MaxHAgg	0.766	0.01
NGenH & MaxDIT	0.862	0.01

Table 3.16: Adjusted R square of correlated structural complexity metrics where dependent variable is understandability level. Discarded metrics are ANAW, ANMW, NAggH, MaxDIT.

Correlated Metrics	Adjusted R Square
ANAW & ANGen	0.068 & 0.123
ANAW & MaxDIT	0.068 & 0.080
ANMUW & ANMW	0.008& 0.004
NAggH & MaxHAgg	-0.019 & -0.020
NGenH & MaxDIT	0.286 & 0.080

Table 3.17: Adjusted R square of correlated structural complexity metrics where dependent variable is modifiability level. Discarded metrics are ANAW, ANMW, NAggH, MaxDIT.

Correlated Metrics	Adjusted Square
ANAW & ANGen	-0.011 & 0.089
ANAW & MaxDIT	-0.011 & 0.068
ANMUW & ANMW	0.101 & 0.090
NAggH & MaxHAgg	-0.025 & -0.028
NGenH & MaxDIT	0.278 & 0.068

Table 3.18. Adjusted R square of correlated structural complexity metrics where dependent variable is maintainability level. Discarded metrics are ANAW, ANMW, NAggH, MaxDIT.

Correlated Metrics	Adjusted R Square
ANAW & ANGen	0.034 & 0.149
ANAW & MaxDIT	0.034 & 0.107
ANMUW & ANMW	0.073 & 0.063
NAggH & MaxHAgg	-0.022 & -0.024
NGenH & MaxDIT	0.386 & 0.107

Table 3.19: Pearson's correlation of aesthetic metrics.

Correlated Metrics	Pearson's Correlation	Significance Level (2-tailed)
UnifEdgeLen & MaxEdgeLen	0.934	0.01
UnifBends & MaxBends	0.891	0.01
AvgCrossS & MaxEdgeLenS	0.812	0.01
AvgUnifEdgeLenS & MaxEdgeLenS	0.874	0.01

Table 3.20: Adjusted R square of correlated aesthetic metrics where dependent variable is understandability level. Discarded metrics are UnifEdgeLen, UnifBends, MaxEdgeLenS.

Correlated Metrics	Adjusted R Square
UnifEdgeLen & MaxEdgeLen	-0.020 & -0.026
UnifBends & MaxBends	0.030 & 0.040
AvgCrossS & MaxEdgeLenS	0.061 & -0.017
AvgUnifEdgeLenS & MaxEdgeLenS	0.047 & -0.017

Table 3.21: Adjusted R square of correlated aesthetic metrics where dependent variable is modifiability level. Discarded metrics are UnifEdgeLen, UnifBends, MaxEdgeLenS.

Correlated Metrics	Adjusted R Square
UnifEdgeLen & MaxEdgeLen	-0.016 & -0.017
UnifBends & MaxBends	0.037& 0.050
AvgCrossS & MaxEdgeLenS	0.065 & 0.032
AvgUnifEdgeLenS & MaxEdgeLenS	0.052 & 0.032

Table 3.22: Adjusted R square of correlated aesthetic metrics where dependent variable is maintainability level. Discarded metrics are UnifEdgeLen, UnifBends, MaxEdgeLenS.

Correlated Metrics	Adjusted R Square
UnifEdgeLen & MaxEdgeLen	-0.021 &-0.023
UnifBends & MaxBends	0.053 & 0.069
AvgCrossS & MaxEdgeLenS	0.093 & 0.013
AvgUnifEdgeLenS & MaxEdgeLenS	0.075 & 0.013

3.3.2.2 Discriminant Analysis

The understandability, modifiability and maintainability obtained by applying Discriminant analysis are shown as Table 3.23 – Table 3.40. In order to classify a new software design model into one of three understandability levels utilizing the obtained understandability model presented in Table 3.23, the metrics presented in each function will be measured from class and sequence diagrams. Function D_Und1, M_Und1 and E_Und1 will be calculated. Then the software design model will be allocated to the group that provides the highest value among 3 functions. For example, if E_Und1 value is more than D_Und1 and M_Und1 values, understandability of the software design model will be categorized into group 3 which is easy level. Other models can be utilized in the same way.

Table 3.23: An understandability model: DiscUnd1.

Difficult level's function: **D_Und1** =1.286×NC +7.626×ANAUW -1.953×ANAW -14.64 ×ANMUW +17.705×ANMW +3.916×ANAssoc-34.652×ANAgg+2.737×NAggH+10.796×MaxHAgg +52.974×ANGen-3.6×NGenH-5.627×MaxDIT+2.222×NOS+5.539×WMBO +2.854×ANRM + 0.275×ANDM+0.011× ANCM -41.146
Medium level's function: **M_Und1** =0.837×NC +7.311×ANAUW -4.58×ANAW -7.526 ×ANMUW +9.387×ANMW +5.151×ANAssoc-25.543×ANAgg+3.334×NAggH+8.857×MaxHAgg +49.447×ANGen-3.988×NGenH-5.719×MaxDIT+3.694×NOS+4.814×WMBO +3.005×ANRM + 1.243×ANDM+0.939× ANCM -38.945
Easy level's function: **E_Und1** =0.933×NC +8.222×ANAUW -7.844×ANAW -6.882 ×ANMUW +9.731×ANMW +4.052×ANAssoc-23.399×ANAgg+2.542×NAggH+8.315×MaxHAgg +46.902×ANGen-3.81×NGenH-5.485×MaxDIT+3.39×NOS+5.035×WMBO +2.76×ANRM + 0.85×ANDM+1.016× ANCM -38.265

Table 3.24: An understandability model: DiscUnd2.

Difficult level's function: **D_Und2** =1.176×NC +6.628×ANAUW+1.27×ANMUW+2.652×ANAssoc -8.927×ANAgg+6.655×MaxHAgg+19.603×ANGen-1.792×NGenH+3.662×NOS +3.876×WMBO+2.4×ANRM + 0.666×ANDM+1.356× ANCM -35.439
Medium level's function: **M_Und2** =0.883×NC +6.317×ANAUW+0.703×ANMUW+3.723×ANAssoc -0.004×ANAgg+5.444×MaxHAgg+16.499×ANGen-2.25×NGenH+4.361×NOS +2.946×WMBO+2.683×ANRM + 1.171×ANDM+2.292× ANCM -34.679
Easy level's function: **E_Und2** =0.933×NC +6.885×ANAUW+1.582×ANMUW+3.096×ANAssoc -2.773×ANAgg+5.222×MaxHAgg+13.048×ANGen-1.954×NGenH+4.007×NOS +3.753×WMBO+2.581×ANRM + 0.866×ANDM+2.334× ANCM -33.71

Table 3.25: An understandability model: DiscUnd3.

Difficult level's function: **D_Und3** = -13.966×UnifEdgeLen +0.958×TotEdgeLen +14.545×MaxLen +75.821 ×UnifBends -3.752×TotBends -23.729×MaxBends +9.166×Orthogonal +90.151×Join +127.356×Center -30.013×Below -65.209×SameCo +17.276×Indicator +5.072×ACrossS +34.099×MaxEdgeLenS -75.41×AUnifEdgeLenS+38.493×ASubSetSeptS -196.65
Medium level's function: **M_Und3** =-13.015×UnifEdgeLen +0.863×TotEdgeLen +14.699×MaxLen +71.714 ×UnifBends -3.643×TotBends -24.513×MaxBends +5.536×Orthogonal +93.949×Join +129.478×Center -29.798×Below -70.557×SameCo +17.981×Indicator +4.641×ACrossS +34.455×MaxEdgeLenS -73.932×AUnifEdgeLenS+40.159×ASubSetSeptS -195.253
Easy level's function: **E_Und3** =-13.378×UnifEdgeLen +0.871×TotEdgeLen +14.178×MaxLen +66.437 ×UnifBends -3.72×TotBends -20.691×MaxBends +11.921×Orthogonal +84.817×Join +114.802×Center -21.617×Below -61.394×SameCo +15.354×Indicator +4.355×ACrossS +32.547×MaxEdgeLenS -68.231×AUnifEdgeLenS+35.643×ASubSetSeptS -175.241

Table 3.26: An understandability model: DiscUnd4.

Difficult level's function: **D_Und4** = 0.566×TotEdgeLen -0.096×MaxLen -1.538×TotBends +1.379×MaxBends +10.882×Orthogonal +31.092×Join +54.877×Center +1.846×Below -22.151×SameCo +1.412×Indicator +0.872×ACrossS -2.709×AUnifEdgeLenS+9.463×ASubSetSeptS -55.513
Medium level's function: **M_Und4** = 0.504×TotEdgeLen +0.977×MaxLen -1.503×TotBends -1.454×MaxBends +8.831×Orthogonal +33.913×Join +57.379×Center +2.475×Below -27.71×SameCo +1.968 ×Indicator +0.512×ACrossS -0.955×AUnifEdgeLenS+10.241×ASubSetSeptS -55.5
Easy level's function: **E_Und4** = 0.545×TotEdgeLen +0.088×MaxLen -1.762×TotBends +1.294×MaxBends +12.411×Orthogonal +27.2×Join +45.916×Center +8.095×Below -19.74×SameCo +0.417 ×Indicator +0.409×ACrossS +1.028×AUnifEdgeLenS+7.92×ASubSetSeptS -46.366

Table 3.27: An understandability model: DiscUnd5.

Difficult level's function: **D_Und5** =28.665×NC +166.883×ANAUW +120.5×ANAW +198.404 ×ANMUW-82.167 ×ANMW -151.319×ANAssoc-1001.203×ANAgg -5.834×NAggH+191.819×MaxHAgg +765.242 ×ANGen-9.137×NGenH-422.046×MaxDIT-104.145×NOS-81.747×WMBO-99.606×ANRM - 37.411×ANDM-163.904× ANCM-121.976×UnifEdgeLen +13.035×TotEdgeLen+148.692×MaxLen +589.797×UnifBends -54.364×TotBends +3.274×MaxBends -66.651×Orthogonal -71.259×Join -734.778×Center +466.853×Below +559.955×SameCo +104.586×Indicator +58.029×ACrossS +133.067×MaxEdgeLenS-554.49×AUnifEdgeLenS+151.791×ASubSetSeptS-1023.342
Medium level's function: **M_Und5** =26.465×NC +136.626×ANAUW +47.449×ANAW +148.76 ×ANMUW-67.18 ×ANMW -71.741×ANAssoc-726.5×ANAgg -0.742×NAggH+148.004×MaxHAgg+715.851 ×ANGen -27.007×NGenH-315.466×MaxDIT-66.704×NOS-53.686×WMBO-73.18×ANRM - 34.295×ANDM-123.532× ANCM-88.921×UnifEdgeLen +8.017×TotEdgeLen+116.301×MaxLen +446.496×UnifBends -40.421×TotBends -2.776×MaxBends -74.07×Orthogonal -41.191×Join -481.421×Center +387.264×Below +352.14×SameCo +60.593×Indicator +45.954×ACrossS +110.546×MaxEdgeLenS-448.623×AUnifEdgeLenS+136.897×ASubSetSeptS-968.124
Easy level's function: **E_Und5** = 27.722×NC +173.558×ANAUW +90.636×ANAW +237.82 ×ANMUW-125.564 ×ANMW -127.903×ANAssoc-1015.575×ANAgg -0.68×NAggH+190.115×MaxHAgg+725.956 ×ANGen-7.11×NGenH-416.565×MaxDIT-98.692×NOS-87.345×WMBO-100.473×ANRM - 39.427×ANDM-161.196× ANCM-124.521×UnifEdgeLen +12.552×TotEdgeLen+152.048×MaxLen +584.416×UnifBends -53.452×TotBends-1.138×MaxBends -38.053×Orthogonal -150.866×Join -777.813×Center +522.0×Below +562.693×SameCo +89.711×Indicator +56.987×ACrossS +135.231×MaxEdgeLenS-545.687×AUnifEdgeLenS+149.302×ASubSetSeptS-964.687

Table 3.28: An understandability model: DiscUnd6.

Difficult level’s function: **D_Und6** = 8.509×NC +31.982×ANAUW -16.898 ×ANMUW +38.808×ANAssoc +34.451×ANAgg +23.43×MaxHAgg +277.816×ANGen -46.387×NGenH+24.881×NOS+38.545×WMBO +10.463×ANRM-9.143×ANDM+22.483× ANCM -1.418×TotEdgeLen+6.638×MaxLen-0.067×TotBends -13.723×MaxBends-33.6 ×Orthogonal +61.708×Join+305.959×Center +18.882×Below -173.195×SameCo -16.656×Indicator +5.917×ACrossS -40.814×AUnifEdgeLenS+46.72×ASubSetSeptS -299.51
Medium level’s function: **M_Und6** = 8.751×NC +30.577×ANAUW-18.646 ×ANMUW +49.963×ANAssoc +39.266×ANAgg +21.645×MaxHAgg +278.040×ANGen-45.885×NGenH+26.458×NOS +41.5×WMBO+13.392×ANRM-9.37×ANDM+22.931× ANCM-1.772×TotEdgeLen +8.234×MaxLen+0. 406×TotBends -19.706×MaxBends-29.636 ×Orthogonal +64.793×Join+321.403×Center +17.864×Below-196.469×SameCo-19.585×Indicator +4.656×ACrossS -33.398×AUnifEdgeLenS+48.055×ASubSetSeptS-298.65
Easy level’s function: **E_Und6** = 7.794×NC +32.889×ANAUW-16.3 ×ANMUW +43.461×ANAssoc +40.142×ANAgg +22.44×MaxHAgg +250.484×ANGen-43.535×NGenH+24.18×NOS +35.474×WMBO+10.47×ANRM-10.36×ANDM+23.712× ANCM-1.276×TotEdgeLen +5.865×MaxLen-0.376×TotBends-13.199×MaxBends-20.713 ×Orthogonal +38.764×Join+282.232×Center +33.707×Below-164.092×SameCo-20.148×Indicator +5.226×ACrossS -32.615×AUnifEdgeLenS+45.764×ASubSetSeptS-264.343

Table 3.29: A modifiability model: DiscMod1.

Difficult level’s function: **D_Mod1** =0.936×NC +6.417×ANAUW -6.512×ANAW -13.391 ×ANMUW +15.359×ANMW +4.584×ANAssoc-30.105×ANAgg+3.528×NAggH+9.824×MaxHAgg +49.941×ANGen-2.509×NGenH-4.886×MaxDIT+2.983×NOS+5.865×WMBO +3.446×ANRM + 0.872×ANDM+0.034× ANCM -38.307
Medium level’s function: **M_Mod1** =0.915×NC +6.706×ANAUW -4.018×ANAW -11.837 ×ANMUW +12.444×ANMW +5.307×ANAssoc-32.341×ANAgg+3.695×NAggH+11.507×MaxHAgg +71.485×ANGen-8.45×NGenH-5.794×MaxDIT+4.199×NOS+7.225×WMBO +2.912×ANRM + 1.479×ANDM+1.56× ANCM -38.126
Easy level’s function: **E_Mod1** =0.771×NC +8.412×ANAUW -4.158×ANAW -2.064 ×ANMUW +4.125×ANMW +5.296×ANAssoc-20.626×ANAgg+2.914×NAggH+7.796×MaxHAgg +48.234×ANGen-5.254×NGenH-6.356×MaxDIT+4.254×NOS+3.991×WMBO +2546×ANRM + 1.459×ANDM+1.781× ANCM -37.654

Table 3.30: A modifiability model: DiscMod2.

Difficult level’s function **D_Mod2** =0.931×NC +5.532×ANAUW+0.474×ANMUW+3.659×ANAssoc -1.594×ANAgg+5.579×MaxHAgg+13.127×ANGen-0.223×NGenH+4.035×NOS +3.63×WMBO+2.954×ANRM + 0.981×ANDM+1.497× ANCM -33.578
Medium level’s function: **M_Mod2** =0.926×NC +5.889×ANAUW-0.608×ANMUW+3.879×ANAssoc -2.547×ANAgg+7.263×MaxHAgg+34.367×ANGen-6.212×NGenH+5.084×NOS +4.621×WMBO+2.474×ANRM + 1.492×ANDM+3.003× ANCM -33.504
Easy level’s function: **E_Mod2** =0.8×NC +7.342×ANAUW+1.404×ANMUW+3.738×ANAssoc +2.731×ANAgg+4.655×MaxHAgg+14.421×ANGen-3.363×NGenH+4.522×NOS +1.847×WMBO+2.459×ANRM + 1.281×ANDM+3.058× ANCM -32.828

Table 3.31: A modifiability model: DiscMod3.

Difficult level’s function: **D_Mod3** = -16.514×UnifEdgeLen +0.787×TotEdgeLen +17.637×MaxLen +80.527 ×UnifBends -3.236×TotBends -25.927×MaxBends +18.89×Orthogonal +67.151×Join +116.325×Center -30.324×Below -54.141×SameCo +19.317×Indicator +5.746×ACrossS +36.236×MaxEdgeLenS -82.177×AUnifEdgeLenS+35.994×ASubSetSeptS -198.841
Medium level’s function: **M_Mod3** = -15.089×UnifEdgeLen +0.946×TotEdgeLen +15.915×MaxLen +82.853 ×UnifBends -4.014×TotBends -24.72×MaxBends +22.559×Orthogonal +62.255×Join +101.355×Center -26.496×Below -49.053×SameCo +20.02×Indicator +5.142×ACrossS +36.444×MaxEdgeLenS -77.607×AUnifEdgeLenS+37.029×ASubSetSeptS -188.184
Easy level’s function: **E_Mod3** = -14.734×UnifEdgeLen +0.8×TotEdgeLen +15.535×MaxLen +67.992 ×UnifBends -3.52×TotBends -20.851×MaxBends +18.654×Orthogonal +70.676×Join +105.091×Center -19.777×Below -53.195×SameCo +15.798×Indicator +4.628×ACrossS +33.195×MaxEdgeLenS -70.249×AUnifEdgeLenS+33.328×ASubSetSeptS -172.95

Table 3.32: A modifiability model: DiscMod4.

Difficult level’s function: **D_Mod4** = 0.415×TotEdgeLen +0.209×MaxLen -1.114×TotBends +0.297×MaxBends +6.775×Orthogonal +30.481×Join +63.164×Center +3.844×Below -25.455×SameCo -0.132×Indicator +1.091×ACrossS -3.555×AUnifEdgeLenS+7.394×ASubSetSeptS -56.381
Medium level’s function: **M_Mod4** = 0.527×TotEdgeLen +0.07×MaxLen -1.69×TotBends +1.363×MaxBends +13.124×Orthogonal +24.952×Join +46.278×Center +8.199×Below -19.295×SameCo +0.402×Indicator +0.509×ACrossS +0.631×AUnifEdgeLenS+7.472×ASubSetSeptS -46.955
Easy level’s function: **E_Mod4** = 0.476×TotEdgeLen -0.008×MaxLen -1.577×TotBends +1.248×MaxBends +7.699×Orthogonal +31.106×Join +51.318×Center +9.607×Below -21.616×SameCo -1.211×Indicator +0.484×ACrossS +0.682×AUnifEdgeLenS+6.806×ASubSetSeptS -46.832

Table 3.33: A modifiability model: DiscMod5.

Difficult level’s function: **D_Mod5** = 11.185×NC +31.694×ANAUW -40.408×ANAW +15.273 ×ANMUW-16.367 ×ANMW +68.89×ANAssoc-7.508×ANAgg -3.385×NAggH+29.323×MaxHAgg +309.818×ANGen -30.092×NGenH-36.828×MaxDIT+8.236×NOS+15.187×WMBO +3.894×ANRM - 1.689×ANDM+0.493× ANCM-13.395×UnifEdgeLen -1.692×TotEdgeLen +22.71×MaxLen -16.394×UnifBends -1.895×TotBends -10.497×MaxBends +16.203×Orthogonal +73.031×Join +152.272×Center +117.071×Below -129.3×SameCo -43.527×Indicator -1.045×ACrossS -240.208
Medium level’s function: **M_Mod5** = 12.993×NC +32.339×ANAUW -35.826×ANAW +26.089 ×ANMUW-29.117 ×ANMW +78.004×ANAssoc+7.668×ANAgg -2.917×NAggH+31.583×MaxHAgg +376.347×ANGen -45.025×NGenH-42.177×MaxDIT+10.858×NOS+13.42×WMBO +2.068×ANRM +1.308×ANDM-0.491× ANCM-9.954×UnifEdgeLen -2.179×TotEdgeLen +20.645×MaxLen -25.558×UnifBends -2.154×TotBends -0.199×MaxBends +5.862×Orthogonal +67.489×Join +111.721×Center +138.375×Below -103.181×SameCo -47.109×Indicator -1.810×ACrossS -237.167
Easy level’s function: **E_Mod5** = 11.121×NC +53.048×ANAUW +10.32×ANAW +131.102 ×ANMUW-138.881 ×ANMW +76.334×ANAssoc+85.145×ANAgg -5.656×NAggH+18.487×MaxHAgg +319.414×ANGen -31.547×NGenH-74.012×MaxDIT+9.892×NOS+3.491×WMBO -11.640×ANRM -2.510×ANDM+6.841× ANCM-13.183×UnifEdgeLen -0.336×TotEdgeLen +19.309×MaxLen -12.345×UnifBends -5.226×TotBends -12.506×MaxBends +4.525×Orthogonal +67.265×Join +119.255×Center +230.358×Below -110.516×SameCo -56.388×Indicator -0.072×ACrossS -234.157

Table 3.34: A modifiability model: DiscMod6.

Difficult level's function: **D_Mod6** = 4.558×NC +44.64×ANAUW -12.658 ×ANMUW +36.519×ANAssoc +84.392×ANAgg +14.676×MaxHAgg +160.953×ANGen -27.629×NGenH+20.07×NOS+26.06×WMBO +4.358×ANRM-17.656×ANDM+29.333× ANCM+0.063×TotEdgeLen+0.632×MaxLen -1.697×TotBends-14.862×MaxBends-8.52 ×Orthogonal +13.688×Join+274.625×Center +68.509×Below-149.978×SameCo-28.865×Indicator +7.87×ACrossS -35.531×AUnifEdgeLenS+47.506×ASubSetSeptS -262.591
Medium level's function: **M_Mod6** = 4.8×NC +43.886×ANAUW -12.905 ×ANMUW +37.729×ANAssoc +78.186×ANAgg +17.823×MaxHAgg +184.161×ANGen -35.698×NGenH+21.059×NOS+24.932×WMBO +5.051×ANRM-14.612×ANDM+29.481× ANCM+0.063×TotEdgeLen+0.548×MaxLen -2.119×TotBends-11.217×MaxBends-4.677 ×Orthogonal +4.203×Join+250.427×Center +72.169×Below-133.22×SameCo-28.156×Indicator +6.541×ACrossS -29.219×AUnifEdgeLenS+47.698×ASubSetSeptS -256.186
Easy level's function: **E_Mod6** = 2.269×NC +55.912×ANAUW -11.21 ×ANMUW +32.305×ANAssoc +126.701×ANAgg +10.219×MaxHAgg +137.396×ANGen-26.621×NGenH+19.44×NOS+20.185×WMBO +0.503×ANRM-19.566×ANDM+35.365× ANCM+1.231×TotEdgeLen-3.443×MaxLen -3.205×TotBends-17.48×MaxBends-5.271 ×Orthogonal +9.044×Join+279.994×Center +93.09×Below-140.87×SameCo-35.707×Indicator +8.889×ACrossS -35.262×AUnifEdgeLenS+53.535×ASubSetSeptS -241.537

Table 3.35: A maintainability model: DiscMain1.

Difficult level's function: **D_Main1** =1.013×NC +5.546×ANAUW -0.595×ANAW -11.506 ×ANMUW +12.098×ANMW +6.303×ANAssoc-29.149×ANAgg+3.969×NAggH+10.753×MaxHAgg +57.695×ANGen-3.788×NGenH-5.46×MaxDIT+3.66×NOS+7.595×WMBO +3.577×ANRM + 1.241×ANDM+1.213× ANCM -45.413
Medium level's function: **M_Main1** =0.769×NC +6.091×ANAUW +2.742×ANAW -10.675 ×ANMUW +10.264×ANMW +7.383×ANAssoc-42.592×ANAgg+5.481×NAggH+12.83×MaxHAgg +82.179×ANGen-8.622×NGenH-7.772×MaxDIT+4.524×NOS+10.134×WMBO +2.949×ANRM +2.08×ANDM+2.232× ANCM -44.466
Easy level's function: **E_Main1** =0.73×NC +7.787×ANAUW -3.797×ANAW -6.797 ×ANMUW +8.386×ANMW +5.439×ANAssoc-31.11×ANAgg+3.88×NAggH+9.639×MaxHAgg +60.072×ANGen-6.206×NGenH-6.752×MaxDIT+4.065×NOS+5.965×WMBO +2.658×ANRM +1.575×ANDM+1.439× ANCM -37.02

Table 3.36: A maintainability model: DiscMain2.

Difficult level's function **D_Main2** =1.086×NC +4.959×ANAUW -0.218×ANMUW+4.454×ANAssoc +1.927×ANAgg+6.263×MaxHAgg+19.706×ANGen-0.757×NGenH+4.347×NOS +4.297×WMBO+3.2×ANRM + 1.036×ANDM+2.036× ANCM -35.464
Medium level's function: **M_Main2** =0.915×NC +5.575×ANAUW -0.896×ANMUW+4.391×ANAssoc -0.546×ANAgg+7.157×MaxHAgg +34.317×ANGen -4.807×NGenH+5.061×NOS +5.235×WMBO+2.541×ANRM + 1.599×ANDM+3.211× ANCM -35.387
Easy level's function: **E_Main2**=0.816×NC +6.681×ANAUW +0.88×ANMUW+3.568×ANAssoc -0.413×ANAgg+5.278×MaxHAgg +16.703×ANGen -2.912×NGenH+4.432×NOS +2.801×WMBO+2.526×ANRM + 1.246×ANDM+2.506× ANCM -30.679

Table 3.37: A maintainability model: DiscMain3.

Difficult level's function: **D_Main3** = -17.458×UnifEdgeLen +1.025×TotEdgeLen +18.495×MaxLen +78.769 ×UnifBends -3.743×TotBends -24.977×MaxBends +18.767×Orthogonal +75.598×Join +119.119×Center -29.647×Below -60.318×SameCo +17.971×Indicator +5.845×ACrossS +36.618×MaxEdgeLenS -83.08×AUnifEdgeLenS+37.491×ASubSetSeptS -206.491
Medium level's function: **M_Main3** =-15.951×UnifEdgeLen +1.055×TotEdgeLen +17.021×MaxLen +78.398 ×UnifBends -4.216×TotBends -23.114×MaxBends +21.388×Orthogonal +71.173×Join +107.553×Center -25.463×Below -56.67×SameCo +18.81×Indicator +5.138×ACrossS +36.786×MaxEdgeLenS -78.122×AUnifEdgeLenS+38.74×ASubSetSeptS -197.663
Easy level's function: **E_Main3** =-15.455×UnifEdgeLen +0.949×TotEdgeLen +16.303×MaxLen +69.888 ×UnifBends -3.859×TotBends -21.115×MaxBends +19.163×Orthogonal +74.183×Join +107.345×Center -21.116×Below -56.606×SameCo +16.028×Indicator +4.819×ACrossS +34.244×MaxEdgeLenS -72.769×AUnifEdgeLenS+35.229×ASubSetSeptS -179.291

Table 3.38: A maintainability model: DiscMain4.

Difficult level's function: **D_Main4** = 0.547×TotEdgeLen -0.041×MaxLen -1.504×TotBends +1.405×MaxBends +9.896×Orthogonal +25.849×Join +49.546×Center +4.916×Below -18.757×SameCo +0.087×Indicator +0.892×ACrossS -2.332×AUnifEdgeLenS+7.441×ASubSetSeptS -50.774
Medium level's function: **M_Main4** = 0.57×TotEdgeLen +0.113×MaxLen -1.872×TotBends +1.83×MaxBends +14.863×Orthogonal +22.771×Join +40.867×Center +9.179×Below -17.303×SameCo +0.697×Indicator +0.329×ACrossS +1.823×AUnifEdgeLenS+7.857×ASubSetSeptS -47.165
Easy level's function: **E_Main4** = 0.526×TotEdgeLen -0.118×MaxLen -1.804×TotBends +1.551×MaxBends +11.645×Orthogonal +27.694×Join +44.047×Center +10.858×Below -18.717×SameCo -0.659×Indicator +0.308×ACrossS +2.074×AUnifEdgeLenS+6.717×ASubSetSeptS -46.709

Table 3.39: A maintainability model: DiscMain5.

Difficult level's function **D_Main5** =21.999×NC +9.409×ANAUW -337.633×ANAW -209.383× ANMUW+250.244 ×ANMW +114.890×ANAssoc -47.616×ANAgg -4.729×NAggH+26.855×MaxHAgg +486.436×ANGen-17.264×NGenH+31.865×MaxDIT-6.643×NOS+18.310×WMBO +7.418×ANRM+3.919×ANDM-71.346× ANCM-22.392×UnifEdgeLen -5.158×TotEdgeLen +55.315×MaxLen +98.768×UnifBends -1.571×TotBends -58.207×MaxBends -53.670 ×Orthogonal +266.690×Join+92.735×Center -42.126×Below -103.882×SameCo -69.079 ×Indicator +8.137×ACrossS +52.070×MaxEdgeLenS-175.310×AUnifEdgeLenS-417.689
Medium level's function: **M_Main5** = 23.624×NC +6.824×ANAUW -300.362×ANAW-197.772 ×ANMUW+232.154 ×ANMW +126.813×ANAssoc-75.271×ANAgg -0.923×NAggH+34.966×MaxHAgg +569.702×ANGen-39.013×NGenH+25.194×MaxDIT-0.484×NOS+25.604×WMBO +10.902×ANRM+8.858×ANDM-65.508× ANCM-16.413×UnifEdgeLen-6.152×TotEdgeLen +50.591×MaxLen +62.489×UnifBends -0.475×TotBends -37.859×MaxBends-57.327 ×Orthogonal +237.782×Join+70.912×Center -25.845×Below -90.892×SameCo -69.520 ×Indicator +4.502×ACrossS +44.409×MaxEdgeLenS-146.909×AUnifEdgeLenS-387.102
Easy level's function: **E_Main5** = 19.417×NC +27.125×ANAUW-273.173×ANAW-122.521 ×ANMUW+152.081 ×ANMW +104.491×ANAssoc-17.669×ANAgg -3.027×NAggH+23.832×MaxHAgg +465.745×ANGen-24.265×NGenH+3.289×MaxDIT-1.895×NOS+10.106×WMBO -2.584×ANRM+1.457×ANDM-55.750× ANCM-19.270×UnifEdgeLen-4.016×TotEdgeLen +46.296×MaxLen +91.115×UnifBends-2.476×TotBends-52.539×MaxBends-49.062 ×Orthogonal +190.167×Join+63.376×Center +38.545×Below -70.107×SameCo -67.107 ×Indicator +9.021×ACrossS +41.953×MaxEdgeLenS-156.174×AUnifEdgeLenS-342.923

Table 3.40: A maintainability model: DiscMain6.

Difficult level's function: **D_Main6** = 9.942×NC +24.464×ANAUW -14.603 ×ANMUW +55.289×ANAssoc +20.179×ANAgg +25.593×MaxHAgg +259.156×ANGen-42.469×NGenH+22.729×NOS+38.076×WMBO +12.444×ANRM-7.55×ANDM+17.413× ANCM-1.997×TotEdgeLen+8.773×MaxLen -0.228×TotBends-7.686×MaxBends-15.02 ×Orthogonal +31.467×Join+230.229×Center +35.345×Below-143.128×SameCo-23.218×Indicator +3.596×ACrossS -27.3×AUnifEdgeLenS+36.159×ASubSetSeptS -266.059
Medium level's function: **M_Main6** = 10.426×NC +23.538×ANAUW -14.385 ×ANMUW +59.105×ANAssoc +5.443×ANAgg +27.302×MaxHAgg +282.659×ANGen-47.683×NGenH+23.694×NOS+40.553×WMBO +13.418×ANRM-3.571×ANDM+16.04× ANCM-2.201×TotEdgeLen+9.753×MaxLen -0.203×TotBends-7.802×MaxBends-13.873 ×Orthogonal +27.558×Join+216.64×Center +33.859×Below-132.892×SameCo-23.541×Indicator +1.978×ACrossS -21.759×AUnifEdgeLenS+34.256×ASubSetSeptS -265.501
Easy level's function: **E_Main6** = 6.694×NC +35.042×ANAUW -14.038 ×ANMUW +42.118×ANAssoc +41.682×ANAgg +22.757×MaxHAgg +215.83×ANGen-40.804×NGenH+22.118×NOS+29.278×WMBO +8.382×ANRM-11.18×ANDM+24.813× ANCM-0.84×TotEdgeLen+3.719×MaxLen -1.178×TotBends-7.769×MaxBends-5.303 ×Orthogonal +3.107×Join+233.770×Center +55.251×Below-131.067×SameCo-22.889×Indicator +4.93×ACrossS-25.762×AUnifEdgeLenS+42.96×ASubSetSeptS -248.146

3.3.2.3 Decision Tree

The understandability, modifiability and maintainability prediction models obtained by applying Decision tree are shown in Figure 3.5 - Figure 3.13. In order to classify a new software design model utilizing the decision tree, the new software design model is classified by starting at the root node of the desired tree, testing the attribute specified by this node (in this work attribute is metric), then moving down the tree branch corresponding to the value of the attribute. This process is then repeated for the subtree root at the new node until leaf node is reached. The leaf node will provide the predicted class (in this work, class is understandability level or modifiability level or maintainability level). The decision tree has a process of selecting metrics that are useful for classifying classes. So, after constructing the prediction model, only useful metrics are appeared in the prediction model.

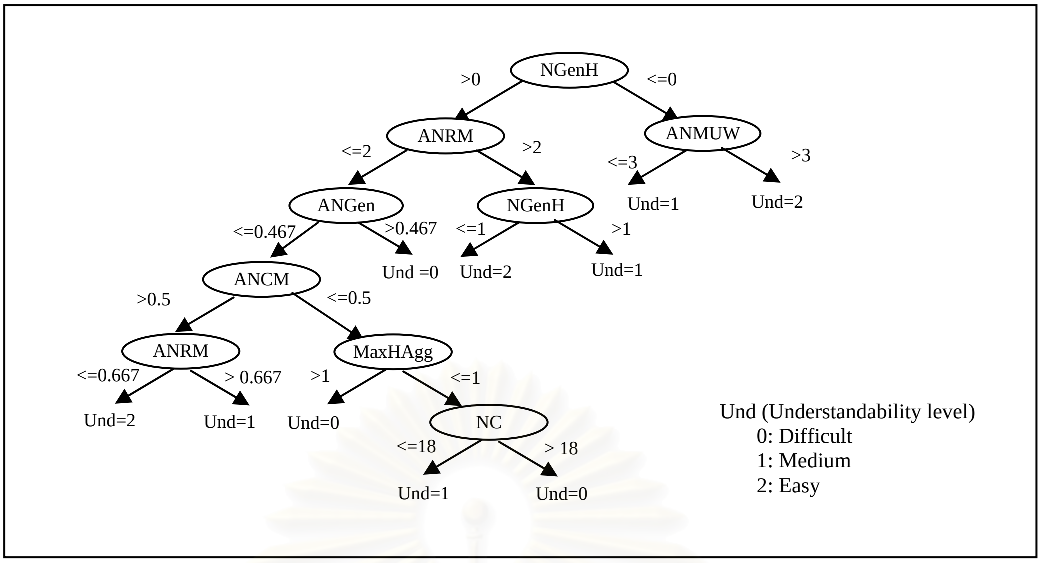


Figure 3.5: Understandability model: DecUnd1, DecUnd2.

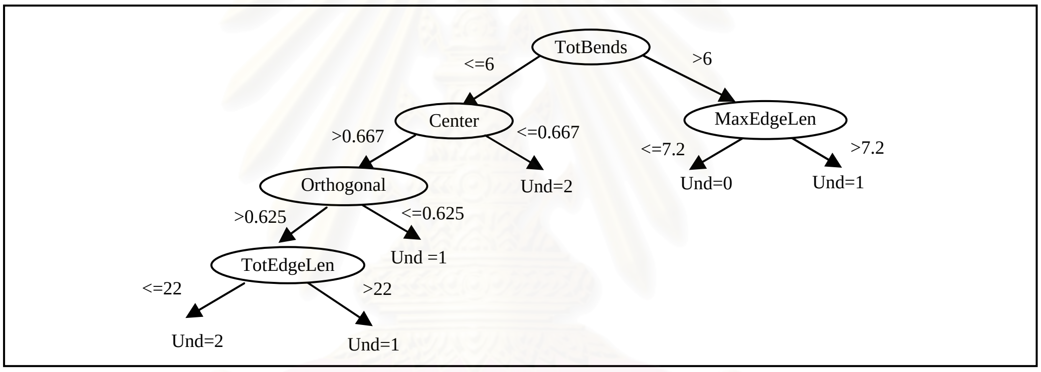


Figure 3.6: Understandability model: DecUnd3, DecUnd4.

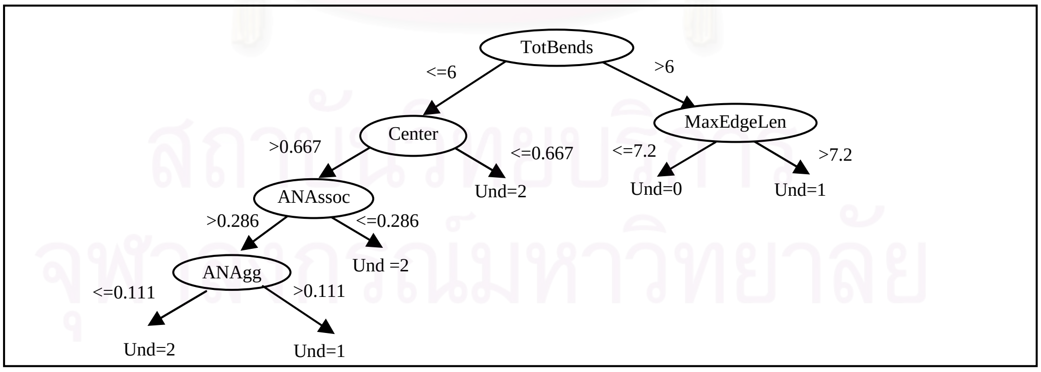


Figure 3.7: Understandability model: DecUnd5, DecUnd6.

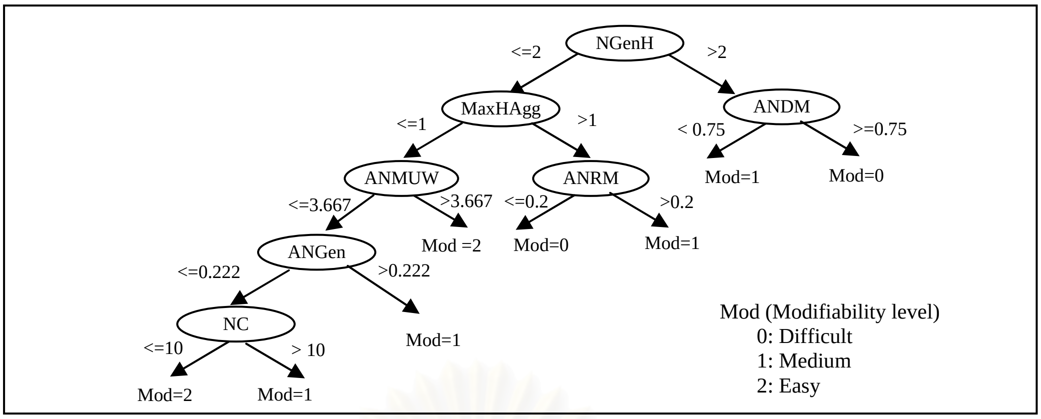


Figure 3.8: Modifiability model: DecMod1, DecMod2.

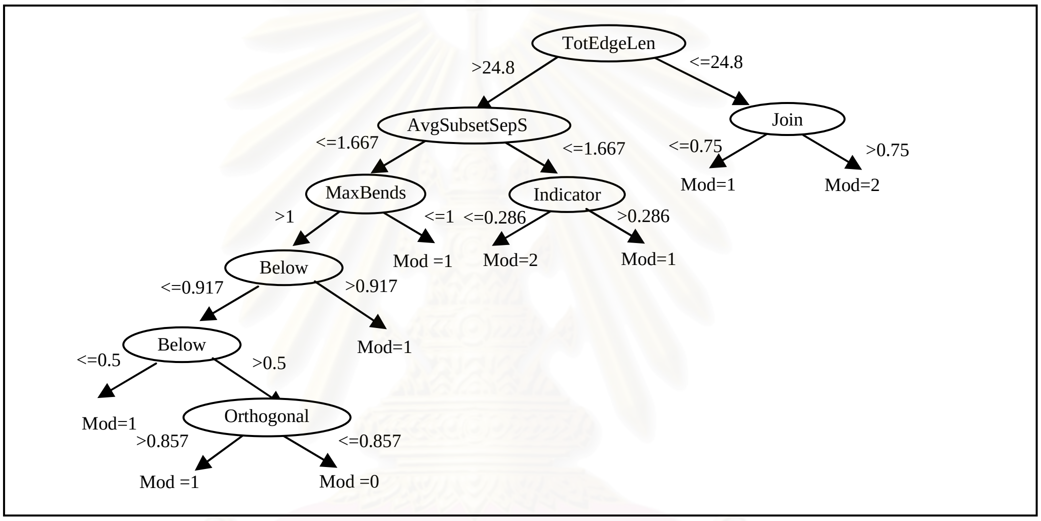


Figure 3.9: Modifiability model: DecMod3, DecMod4.

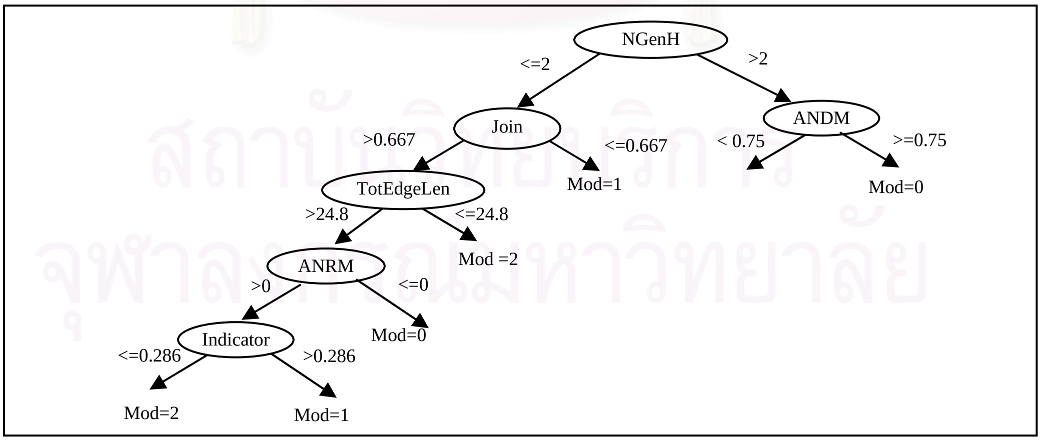


Figure 3.10: Modifiability model: DecMod5, DecMod6.

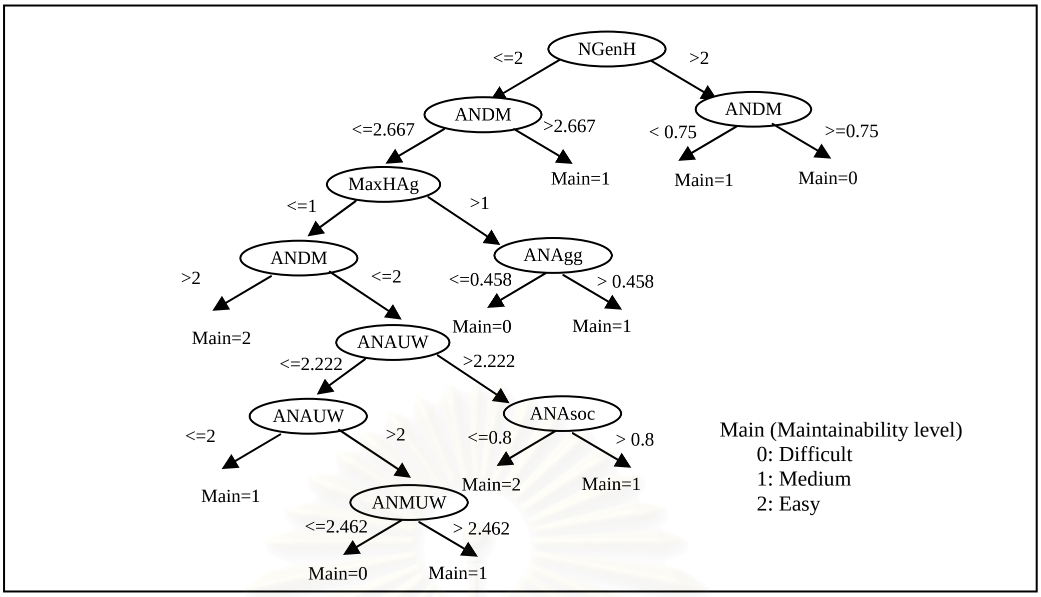


Figure 3.11: Maintainability model: DecMain1, DecMain2.

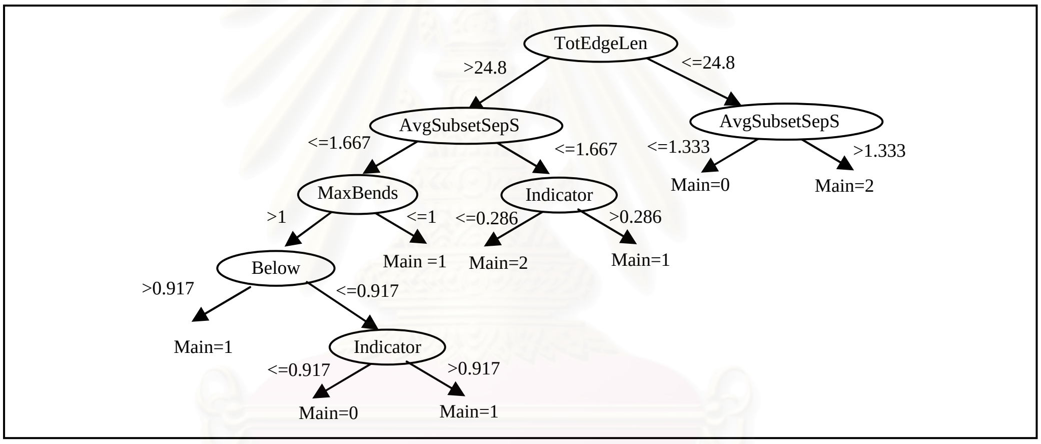


Figure 3.12: Maintainability model: DecMain3, DecMain4.

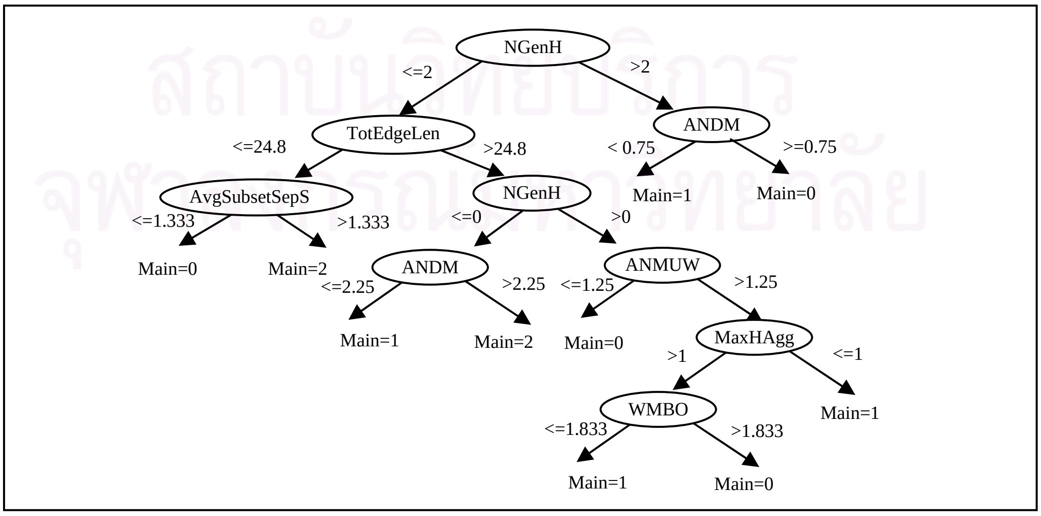


Figure 3.13: Maintainability model: DecMain5, DecMain6.

3.3.2.4 Multilayer Perceptron Neural Network

Eighteen prediction models are constructed applying MLP neural network as already mentioned in section 3.3.2. MLP neural network does not has a process of selecting metrics during constructing the prediction model. Therefore, the obtained prediction model still contains all metrics used to construct it. These metrics are used as input values in the input layer. Each prediction model also contains a set of connection weights and other parameters of neural network. The prediction model in a form of neural network can be viewed like a black box. In order to classify understandability level, modifiability level or maintainability level of a new software design model, all metrics used in each prediction model are measured from the software design model. They are used to be input values of the prediction model. Then the prediction model provides the output as the predicted level of understandability, modifiability or maintainability.

3.3.3 Comparison between Prediction Models Obtained by Applying Discriminant Analysis, Decision Tree and MLP Neural Network

There is a number of different approaches which can be used for estimating the accuracy of a prediction model, for example, using a hold-out sample, bootstrapping, or leave-one-out cross-validation. It has been recommended that in studies where sample sizes are less than 100, as in this case, a leave-one-out approach provides reliable estimates of accuracy [39]. Therefore this approach is applied for accuracy estimation. Leave-one-out cross validation is simply n-fold cross validation, where n is a number of instances in the dataset. Each sample in turn is left out, and the learning scheme is trained on all the remaining instances. It is judged by its correctness on the remaining instance, success or failure. The results of all n judgments, one for each member of the dataset, are averaged, and that average represents the final error estimate.

Table 3.41 – Table 3.43 summarize each prediction model in terms of the metrics used to construct each model and the model accuracy.

Table 3.41: Result summation of understandability models.

Model	Metric						Number of metrics	Fit				Validate			
	Structural complexity metrics		Aesthetic metrics		Structural complexity and aesthetic metrics			Accuracy rate (%)	Miss rate (%)			Accuracy rate (%)	Miss rate (%)		
	All metrics	Discard correlated metrics	All metrics	Discard correlated metrics	All metrics	Discard correlated metrics			Type A	Type B	Sum		Type A	Type B	Sum
DiscUnd1	X						17	85.0	15.0	0.0	15.0	70.0	22.5	7.5	30.0
DecUnd1	X						7	95.0	5.0	0.0	5.0	72.5	22.5	5.0	27.5
MLPUnd1	X						17	100.0	0.0	0.0	0.0	82.5	15.0	2.5	17.5
DiscUnd2		X					13	85.0	15.0	0.0	15.0	67.5	25.0	7.5	32.5
DecUnd2		X					7	95.0	5.0	0.0	5.0	72.5	22.5	5.0	27.5
MLPUnd2		X					13	97.5	2.5	0.0	2.5	75.0	22.5	2.5	25.0
DiscUnd3			X				16	92.5	5.0	2.5	7.5	70.0	22.5	7.5	30.0
DecUnd3			X				5	95.0	5.0	0.0	5.0	75.0	17.5	7.5	25.0
MLPUnd3			X				16	97.5	2.5	0.0	2.5	80.0	17.5	2.5	20.0
DiscUnd4				X			13	90.0	7.5	2.5	10.0	72.5	22.5	5.0	27.5
DecUnd4				X			5	95.0	5.0	0.0	5.0	75.0	17.5	7.5	25.0
MLPUnd4				X			13	97.5	2.5	0.0	2.5	72.5	27.5	0.0	27.5
DiscUnd5					X		33	97.5	2.5	0.0	2.5	72.5	22.5	5.0	27.5
DecUnd5					X		5	90.0	7.5	2.5	10.0	70.0	27.5	2.5	30.0
MLPUnd5					X		33	100.0	0.0	0.0	0.0	82.5	17.5	0.0	17.5
DiscUnd6						X	27	95.0	5.0	0.0	5.0	72.5	27.5	0.0	27.5
DecUnd6						X	5	90.0	7.5	2.5	10.0	75.0	17.5	7.5	25.0
MLPUnd6						X	27	100.0	0.0	0.0	0.0	80.0	17.5	2.5	20.0

Table 3.42: Result summation of modifiability models.

Model	Metric						Number of metrics	Fit				Validate			
	Structural complexity metrics		Aesthetic metrics		Structural complexity and aesthetic metrics			Accuracy rate (%)	Miss rate (%)			Accuracy rate (%)	Miss rate(%)		
	All metrics	Discard correlated metrics	All metrics	Discard correlated metrics	All metrics	Discard correlated metrics			Type A	Type B	Sum		Type A	Type B	Sum
DiscMod1	X						17	90.0	7.5	2.5	10.0	67.5	25.0	7.5	32.5
DecMod1	X						7	95.0	5.0	0.0	5.0	80.0	20.0	0.0	20.0
MLPMod1	X						17	97.5	2.5	0.0	2.5	80.0	20.0	0.0	20.0
DiscMod2		X					13	92.5	5.0	2.5	7.5	75.0	17.5	7.5	25.0
DecMod2		X					7	95.0	5.0	0.0	5.0	80.0	20.0	0.0	20.0
MLPMod2		X					13	**100.0**	**0.0**	**0.0**	**0.0**	**82.5**	**17.5**	**0.0**	**17.5**
DiscMod3			X				16	87.5	12.5	0.0	12.5	70.0	25.0	5.0	30.0
DecMod3			X				7	97.5	2.5	0.0	2.5	72.5	22.5	5.0	27.5
MLPMod3			X				16	100.0	0.0	0.0	0.0	80.0	15.0	5.0	20.0
DiscMod4				X			13	87.5	10.0	2.5	12.5	67.5	25.0	7.5	32.5
DecMod4				X			7	97.5	2.5	0.0	2.5	72.5	22.5	5.0	27.5
MLPMod4				X			13	100.0	0.0	0.0	0.0	80.0	15.0	5.0	20.0
DiscMod5					X		30	97.5	2.5	0.0	2.5	72.5	25.0	2.5	27.5
DecMod5					X		6	92.5	5.0	2.5	7.5	70.0	25.0	5.0	30.0
MLPMod5					X		33	97.5	0.0	2.5	2.5	80.0	20.0	0.0	20.0
DiscMod6						X	27	97.5	2.5	0.0	2.5	77.5	15.0	7.5	22.5
DecMod6						X	6	92.5	5.0	2.5	7.5	72.5	20.0	7.5	27.5
MLPMod6						X	27	97.5	2.5	0.0	2.5	80.0	17.5	2.5	20.0

Table 3.43: Result summation of maintainability models.

Model	Metric						Number of metrics	Fit				Validate			
	Structural complexity metrics		Aesthetic metrics		Structural complexity and aesthetic metrics			Accuracy rate (%)	Miss rate (%)			Accuracy rate (%)	Miss rate (%)		
	All metrics	Discard correlated metrics	All metrics	Discard correlated metrics	All metrics	Discard correlated metrics			Type A	Type B	Sum		Type A	Type B	Sum
DiscMain1	X						17	87.5	10.0	2.5	12.5	70.0	22.5	7.5	30.0
DecMain1	X						7	97.5	2.5	0.0	2.5	77.5	15.0	7.5	22.5
MLPMain1	X						17	97.5	0.0	2.5	2.5	82.5	15.0	2.5	17.5
DiscMain2		X					13	87.5	10.0	2.5	12.5	72.5	20.0	7.5	27.5
DecMain2		X					7	97.5	2.5	0.0	2.5	75.0	20.0	5.0	25.0
MLPMain2		X					13	**100.0**	**0.0**	**0.0**	**0.0**	**85.0**	**12.5**	**2.5**	**15.0**
DiscMain3			X				16	85.0	12.5	2.5	15.0	70.0	25.0	5.0	30.0
DecMain3			X				5	97.5	2.5	0.0	2.5	77.5	17.5	5.0	22.5
MLPMain3			X				16	97.5	0.0	2.5	2.5	80.0	15.0	5.0	20.0
DiscMain4				X			13	85.0	12.5	2.5	15.0	72.5	20.0	7.5	27.5
DecMain4				X			5	97.5	2.5	0.0	2.5	80.0	15.0	5.0	20.0
MLPMain4				X			13	97.5	2.5	0.0	2.5	82.5	17.5	0.0	17.5
DiscMain5					X		32	97.5	2.5	0.0	2.5	77.5	17.5	5.0	22.5
DecMain5					X		7	97.5	2.5	0.0	2.5	80.0	12.5	7.5	20.0
MLPMain5					X		33	97.5	0.0	2.5	2.5	80.0	20.0	0.0	20.0
DiscMain6						X	27	95.0	5.0	0.0	5.0	75.0	20.0	5.0	25.0
DecMain6						X	7	97.5	2.5	0.0	2.5	75.0	17.5	7.5	25.0
MLPMain6						X	27	100.0	0.0	0.0	0.0	82.5	15.0	2.5	17.5

Table 3.41 - Table 3.43 show the result summation of all prediction models. Model accuracy is presented in Fit column and Validate column. The result in Fit column is captured from using the prediction models classify 40 software design models which are used to construct the prediction models. The result in Validate column is captured from leave-one-out cross-validation.

Percent of Type A error can be computed from the total number of cases that meet the following conditions divided by the total number of cases.

- Case in group of 0 is classified into group of 1.
- Case in group of 1 is classified into group of 0.
- Case in group of 1 is classified into group of 2.
- Case in group of 2 is classified into group of 1.

Percent of Type B error can be computed from the total number of cases that meet the following conditions divided by the total number of cases.

- Case in group of 0 is classified into group of 2.
- Case in group of 2 is classified into group of 0.

It is obvious that Type B error is more fatal than Type A error. For example a case in group of 0 is classified into group of 2 means a software design model which is difficult to understand/modify/maintain is predicted that it is easy to understand/modify /maintain.

Prediction models obtained by applying Discriminant analysis, Decision tree and MLP neural network can be compared as follows.

- Prediction model in a form of decision tree can obviously indicate that which metrics can be good indicators of each quality. A metric placed in upper node shows that it is better than other metrics placed in lower nodes for being indicator. Prediction models presented in Figure 3.5 - Figure 3.13 show that the best indicators for understandability are NGenH and TotBends. While NGenH and TotEdgeLen are the best indicators for modifiability and maintainability. For prediction models in a form of discriminant functions presented in Table 3.23 - Table 3.40, we cannot conclude that which metrics can be the best indicator of each quality. The classification function coefficient of each metric cannot indicate that which metric is better because of difference of metric unit. For finding better indicators, we have to normalize value of each metric and reconstruct prediction models. Then, we can find which metric is the best indicator by considering the most classification function coefficient (not consider sign). Neural network does not provide any information which can indicate that which metrics are better indicators for each quality.

- Decision tree and Discriminant analysis have a process of selecting metrics which are significant for classifying level of each quality. While MLP neural network does not has this process during constructing prediction models. Therefore, MLP neural network uses all metrics for constructing prediction model without discarding any metrics. The experimental results show that the number of metrics used in each prediction model obtained from Decision tree is less than the number of metrics used in the prediction model constructed from the same metric set applying Disciminant analysis and MLP neural network. The number of metrics used in each prediction model is listed in Table 3.41 – Table 3.43.
- Consider model accuracy presented in Table 3.41 – Table 3.43, the average accuracy in Fit column and Validate column of prediction models obtained from MLP neural network is higher than that of prediction models obtained from Discriminant analysis and Decision tree.
- Concerns with the usage of prediction model, prediction model in a form of decision tree can be used easily and manually. Prediction model in a form of discriminant functions is not too difficult for calculating. While prediction model in a form of neural network is very difficult for manually use because its algorithm is very complex. However, this problem can be solved by implementing a tool for utilizing the prediction model automatically.

3.3.4 Conclusion and Discussion

The obtained prediction models can be concluded and discussed as follows:

- For the understandability prediction models obtained from Decision tree presented in Figure 3.5 – Figure 3.7, we can find that DecUnd1 is similar to

DecUnd2, DecUnd3 is similar to DecUnd4, and DecUnd5 is similar to DecUnd6. As already mentioned in section 3.3.2, DecUnd1 is constructed from all structural complexity metrics. DecUnd2 is constructed from the same metric set discarded correlated metrics. After finishing constructing both prediction models, metrics appeared in decision trees of DecUnd1 and DecUnd2 are similar. This result implies that correlated metrics are not significant for classifying level of understandability. The similarity of DecUnd3 and DecUnd4, DecUnd5 and DecUnd6, DecMod1 and DecMod2, DecMod3 and DecMod4, DecMod5 and DecMod6, DecMain1 and DecMain2, DecMain3 and DecMain4, DecMain5 and DecMain6 can be described in the same way.

- Prediction models obtained from Decision tree show that the best indicators for understandability are NGenH and TotBends. While NGenH and TotEdgeLen are the best indicators for modifiability and maintainability. NGenH is a structural complexity metric. It is the number of generalization hierarchies which is one of inheritance metrics. The experimental result in [40] showed that NGenH is the best indicator for predicting modifiability correctness for the maintenance process. Another experimental result of the same researcher group confirmed that NGenH is the good indicator for predicting modifiability time [38]. Inheritance increases reuseability and improves similarity of implementation. On the other hand, it also increases the complexity of a software and the coupling between classes leading to the increasing of effort put for maintenance. This result is supported in the following studies. An experimental investigation found that making changes to a C++ program with inheritance consumes more effort than a program without inheritance [41]. Another controlled experiment was conducted to establish the effects of varying levels of inheritance on understandability and modifiability [23]. Result of the experiment indicated that a software without inheritance was easier to modify than a corresponding software containing three or five levels of inheritance. It was also easier to understand a software without inheritance than a corresponding version containing three levels of inheritance. TotBends (the total number of bends) and TotEdgeLen (the total edge length) are aesthetic metrics. The result corresponds to their aesthetic criterion that is the total number of bends and the total edge length should be minimized. It is supported by experimental results presented in [27,36]. Both experimental results showed that decreasing the number of edge bends and the total number of edge length in a graph increases the understandability of the graph.

- Fit column of Table 3.41 – Table 3.43 show that the model accuracy of every prediction model obtained from MLP neural network is greater than or equal to that of prediction models obtained from Discriminant analysis and Decision tree. The accuracy of each prediction model obtained from MLP neural network is 97.5 or 100. The accuracy value of 97.5 means the 39 out of 40 samples are correctly classified. In other words, only 1 sample is incorrectly classified. The reason may be that for constructing a prediction model using MLP Neural Network, we can change some parameters in neural network, for instance learning rate, the number of epochs and the number of nodes in hidden layer. For the same set of samples, we can run neural network many times with different parameters to find the best model which has the highest accuracy.
- In case of automated utilizing the prediction models, the best prediction models for understandability, modifiability and maintainability should be MLPUnd1, MLPMod2 and MLPMain2 respectively. The reason is that these prediction models use structural complexity metrics which can be measured by an automated tool. While aesthetic metrics is very difficult for automated measuring. Another reason is that these prediction models provide the most model accuracy.

3.4 Threats to Validity

Following several empirical studies [17,18,33], this section discusses the various issues that threaten the validity of the experiment.

3.4.1 Threats to Internal Validity

The internal validity is the degree of confidence in a cause-effect relationship between factors of interest and the observed results.

- **Differences among subjects.** Each software design model was evaluated by group of 3 subjects. Although the ability to understand and to modify the software design with UML was not exactly equivalent among groups. Differences among groups were reduced by assigning one A, one B+ and one B students to each group.
- **Knowledge of the universe of discourse among software design models.** Software design models were designed from different universe of discourse, but they were simple enough to be easily understood by the subjects. So, knowledge of the domain did not affect internal validity.
- **Accuracy of subject responses.** Subjects had medium experience in modeling the software design with UML. Their responses to examination were considered valid.
- **Learning effects.** Learning effect was little relevant because each subject performed experimental task of only 2 software design models.
- **Fatigue effects.** Each subject performed experimental tasks of 2 software design models with 20-minute break between them. The fatigue was little relevant.
- **Persistence effects.** Subjects had never performed a similar experiment. So, persistence effect was avoided.
- **Other factors.** Plagiarism and influence between subjects were controlled. Two subjects who sat adjacently performed different examinations. The test was controlled by 2 monitors. Subjects were asked to avoid talking to each other.

3.4.2 Threats to External Validity

External validity is the degree to which the research results can be generalized to the population under study and to other research settings.

- **Materials and tasks used.** Examination questions tried to capture understandability and modifiability of software design models. All questions were approved by experts. Class and sequence diagrams used in this work represented real software, but the software were small and simple. The software which had maximum number of classes contained only 36 classes. This is a limitation of the study since it is not easy to find software design models of real world software.
- **Experimental Subject.** To solve the problem of lacking expert participation, students were used as experimental subjects. We are aware that more experiments with experts should be carried out in order to be able to generalize the results. Nevertheless, this experiment did not require high level of industrial experience. Students are usually accepted as valid subjects [17,18,33].

CHAPTER IV

A NEW PROPOSED SET OF STRUCTURAL COMPLEXITY METRICS FOR MAINTAINABILITY

Chapter III proposed constructing the maintainability prediction models by using 3 classification techniques. Independent variables or predictors are structural complexity metrics and/or aesthetic metrics. Dependent variables are understandability, modifiability and maintainability levels which are measured subjectively from the experimental subjects. Therefore, this chapter proposes a new set of structural complexity metrics to measured maintainability objectively in early phase of object-oriented software life cycle.

Objective measurement is the repetition of a unit amount that stays constant and unchanging (within the allowable error) across the persons measured, across different brands of instruments, and across instrument users [42].

There are two important things for software maintenance: in-depth understanding the structure and behavior of the software, and the ability to make changes easily. So, this chapter introduces a new set of structural complexity metrics for understandability and modifiability as constituting the metrics for maintainability. Programming is sometimes called the art. In this sense, programming may be viewed as a technique completed with heuristics such as with the art [43]. It follows from this that software metrics should consider heuristic properties. Accordingly, it is not reasonable to measure the software product and process, which is actually a labor-intensive industry, only by mathematical and logical metrics without considering the human aspects. Therefore, the measurement of understandability and modifiability should be considered in a heuristic way.

4.1 Metric Definition

Sheldon et al. proposed the metrics for maintainability of class inheritance hierarchies in [43]. Their work focused on maintainability related to inheritance only. We adapt their idea and extend their work by considering maintainability related to other relationships. This section begins with defining the new proposed metrics. Section 4.2 describes how to compute these metrics.

4.1.1 Metrics for Understandability

Metrics for understandability consist of 8 metrics for a class and 1 metric for a software.

Metrics for understandability of a class

- Und_{Gen} (X) is the degree of understandability of class X related to generalization.
- Und_{Agg}(X) is the degree of understandability of class X related to aggregation.
- Und_{Com}(X) is the degree of understandability of class X related to composition.
- Und_{CAssoc}(X) is the degree of understandability of class X related to common association.
- Und_{AssocC} (X) is the degree of understandability of class X related to association class.
- Und_{Dep}(X) is the degree of understandability of class X related to dependency.
- Und_{Real}(X) is the degree of understandability of class X related to realization.
- Und_{Class}(X) is the degree of understandability of class X.

Metric for understandability of a software

- $AvgUnd_{Sys}$(S) is the average degree of understandability of software S.

4.1.2 Metrics for Modifiability

Metrics for modifiability consist of 8 metrics for a class and 1 metric for a software.

Metrics for Modifiability of a Class

- Mod_{Gen} (X) is the degree of modifiability of class X related to generalization.
- Mod_{Agg}(X) is the degree of modifiability of class X related to aggregation.
- Mod_{Com}(X) is the degree of modifiability of class X related to composition.

- $\mathrm{Mod}_{CAssoc}(X)$ is the degree of modifiability of class X related to common association.
- Mod_{AssocC} (X) is the degree of modifiability of class X related to association class.
- $\mathrm{Mod}_{Dep}(X)$ is the degree of modifiability of class X related to dependency.
- $\mathrm{Mod}_{Real}(X)$ is the degree of modifiability of class X related to realization.
- $\mathrm{Mod}\ _{Class}(X)$ is the degree of modifiability of class X.

Metric for Modifiability of a Software

- $\mathrm{AvgMod}_{Sys}(S)$ is the average degree of modifiability of software S.

4.1.3 Terms and Functions

In order to compute metrics introduced in section 4.1.1 and 4.1.2, terms and functions used to compute them are defined as follows.

Head and Tail Classes

For each relationship, let arrowhead of relationship line indicates the position of each class. If relationship line points from class A to class B, then class B is called a *head class* of class A and class A is called a *tail class* of class B.

Consider a hierarchy of i relationship related to class X, where i can be Gen(Generalization), CAssoc(Common association), AssocC(Association class), Agg(Aggregation), Com(Composition), Dep(Dependency) or Real (Realization).

- ImHead_i (X) is immediate head classes of class X related to class X by i relationship.
- ImTail_i (X) is immediate tail classes of class X related to class X by i relationship.
- $\mathrm{AllHead}_i$ (X) is all head classes of class X related to class X by i relationship.
- $\mathrm{AllTail}_i$ (X) is all tail classes of class X related to class X by i relationship.

For better understanding, please see examples of the usage of these metrics.

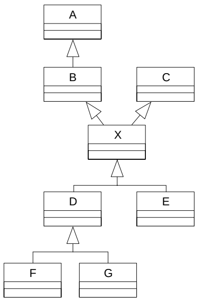


Figure 4.1: A hierarchy of generalization related to class X.

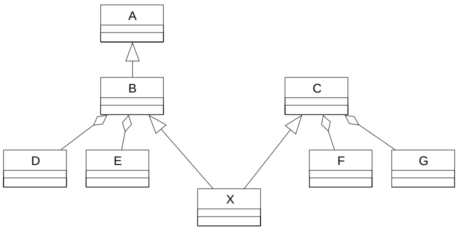


Figure 4.2: A hierarchy of generalization and aggregation related to class X.

For Figure 4.1, $ImHead_{Gen}(X)$ = B,C. $ImTail_{Gen}(X)$ = D,E. $AllHead_{Gen}(X)$ = A,B,C. and $AllTail_{Gen}(X)$ = D,E,F,G.

For Figure 4.2,

$$\begin{aligned} ImTail_{Agg}((AllHead_{Gen}(X))) &= ImTail_{Agg}(A,B,C) \\ &= ImTail_{Agg}(A) + ImTail_{Agg}(B) + ImTail_{Agg}(C) \\ &= \{\} + D,E \ + F,G \\ &= D,E,F,G \end{aligned}$$

Class Complexity

C(X) is the complexity of class X. It is simply defined as follows [43].

C(X) = number of methods of class X + number of attributes of class X.

In case of many classes in parenthesis, the value of the complexity is sum of the complexity of all classes. For example

$$C(X,Y,Z) = C(X) + C(Y) + C(Z).$$

Dependency Weight Value of Relationships

W_i is dependency weight value of i relationship. The dependency between classes is the main cause of the amount of the complexity on understanding and modifying the relationships between the classes. Different kinds of relationship influence the dependency between classes in different degrees. So, dependency weight value of each relationship should be defined in order to indicate its dependency degree. This thesis considers 7 kinds of relationship consisting of dependency, common association, association class, aggregation, composition, generalization and realization. Table 4.1

shows dependency weight value of relationships proposed by Kang et al. [44]. Relationships in this table are sorted from weak to strong dependency degree.

Dependency is the most common relationship. A dependency shows that there is dependency between two classes without any explanations and restrictions, so W_{Dep} should be the minimum. Common association denotes the relationship between instances of classes; it cannot be weaker than dependency relationship. Association class adds restrictions to association, it may be more complex and the dependency between classes with this relationship may be stronger than between classes with common association. Aggregation is a specific association, and composition is a specific aggregation. For example, A is a composite of B; when A is destroyed, B should be destroyed or given to another object. Aggregation does not has this restriction, but it is more restrict than dependency relationship, as one object cannot aggregate itself directly or indirectly. In generalization, subclasses inherited all characteristics of the parent classes, and composition classes can only access the public elements of the nested classes. When parent classes are concrete, subclass can add new elements and override inherited operations. When parent classes are abstract, subclasses should implement the virtual operations of the parent classes or they cannot have any instances. Considering realization, when realizing a class (usually interface), an implementation class must realize all the operations of the interface. So realization has the highest weight value.

Table 4.1: Dependency weight value of relationships.

No.	Relationship	Weight
1	Dependency	W_{Dep}
2	Common association	W_{CAssoc}
3	Association class	W_{AssocC}
4	Aggregation	W_{Agg}
5	Composition	W_{Com}
6	Generalization	W_{Gen}
7	Realization	W_{Real}

In [44], all weight values are summarized in the following form:

$$W_{Dep} <= W_{CAssoc} <= W_{AssocC} <= W_{Com} \quad (1)$$

$$W_{Dep} < W_{Agg} < W_{Com} < W_{Gen} < W_{Real} \quad (2)$$

Following [44], each relationship is given a weight value satisfying Equations (1) and (2): W_{Dep} =1, W_{CAssoc} = 2, W_{AssocC} = 3, W_{Agg} = 4,W_{Com} = 5, W_{Gen} = 6, W_{Real} = 7.

4.2. Metric Derivation

This section describes derivation of all metrics defined in sections 4.1.1 and 4.1.2.

4.2.1. Metrics for Understandability

4.2.1.1 Understandability of a Class

Metrics for understandability of a class proposed with an idea that if we want to understand class X, we should not only read class X, but also read classes related to class X. Therefore, in order to find the degree of understandability of class X, we should consider efforts put for both understanding class X and understanding classes related to class X.

In this work, the effort put for understanding class X is represented by complexity of class X or C(X) because if class X has high complexity, we should put much effort to understand it. The effort put for understanding classes related to class X will be represented by 7 metrics: Und_{Gen}, Und_{Agg}, Und_{Com}, Und_{CAssoc}, Und_{AssocC}, Und_{Dep} and Und_{Real}. They are calculated according to the kind of relationships that class X and the others are related. The degree of understandability of class X is summation of the efforts put for understanding class X and classes related to class X as follows.

$$Und_{Class}(X) = C(X) + Und_{Gen}(X) + Und_{Agg}(X) + Und_{Com}(X) + Und_{CAssoc}(X) + Und_{AssocC}(X) + Und_{Dep}(X) + Und_{Real}(X)$$

In this work, relationships between 2 classes are classified into 2 kinds: direct and indirect relationships. If there is a relationship line connecting between class A and class B, then class A and class B have direct relationship. If class A is a descendant class of class B and there is a relationship line connecting between class A and class C, then class B and class C have indirect relationship. In other words, indirect relationship is relationship through inheritance.

Understandability Related to Generalization

Consider classes related to class X by generalization. We should understand its ancestor classes because class X inherits characteristics of its ancestor classes. These classes can be represented by AllHead$_{Gen}$(X) as an example shown in dotted oval area of Figure 4.3. Und$_{Gen}$ can be defined in expression of multiplication between dependency weight value of generalization and complexity of classes related to generalization as follows.

$$\text{Und}_{Gen}(X) = W_{Gen}\, C(\text{AllHead}_{Gen}(X))$$

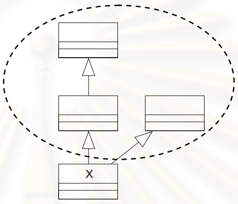


Figure 4.3: Generalization.

Understandability Related to Aggregation

Aggregation is a stronger form of association. It is used to show a logical containment relationship. Consider classes related to class X by aggregation. We should understand classes which are parts of class X. These classes can be represented by ImTail$_{Agg}$(X) as an example shown in dotted oval area of Figure 4.4. Furthermore, we should understand all classes related to class X by indirect aggregation as an example shown in dotted rectangle area of Figure 4.4. Two classes are the parts of a descendant class of class X. Therefore, they are the parts of class X as well. These classes can be represented by ImTail$_{Agg}$(AllHead$_{Gen}$ (X)). Und$_{Agg}$(X) is defined as follows.

$$\text{Und}_{Agg}(X) = W_{Agg}\ [\ C(\text{ImTail}_{Agg}(X)) + C(\text{ImTail}_{Agg}(\text{AllHead}_{Gen}\ (X))\ ]$$

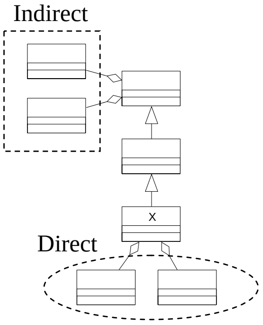


Figure 4.4: Aggregation.

Understandability Related to Composition

Composition is specific aggregation. Classes related to class X by composition can be considered similar to classes related to class X by aggregation with different dependency weight value. Examples of these classes are shown in Figure 4.5. $Und_{Com}(X)$ is defined as follows.

$$Und_{Com}(X) = W_{Com}\ [\ C(ImTail_{Com}(X)) + C(ImTail_{Com}(AllHead_{Gen}(X))\ ]$$

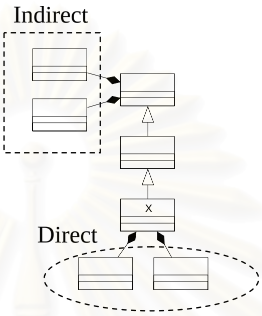


Figure 4.5: Composition.

Understandability Related to Common Association

Conceptually, an association between two classes signifies that some sort of structural relationship exists between the classes. Associations may be unidirectional or bidirectional. A unidirectional association implies that an object of the class which the arrow is originating from may invoke methods on the class towards which the arrow is pointing to. This manifests itself as an instance variable on the class that may invoke methods. A bidirectional association simply means that either object in the association may invoke methods on the other.

Consider classes related to class X by unidirectional common association. We should understand structure of classes invoked by class X (i.e., classes that has the arrowhead side of the relationship). These classes are represented by $ImHead_{CAssoc}(X)$ as an example shown in dotted oval are of Figure 4.6. We should also understand all classes related to class X by indirect common association as an example shown in dotted rectangle area of Figure 4.6. These classes can be represented by $ImHead_{CAssoc}(AllHead_{Gen}(X))$. $Und_{CAssoc}(X)$ is defined as follows.

$$Und_{CAssoc}(X) = W_{CAssoc}\ [\ C(ImHead_{CAssoc}(X)) + C(ImHead_{CAssoc}(AllHead_{Gen}(X)))\ ]$$

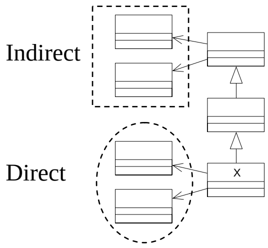


Figure 4.6: Common association.

For bidirectional common association, we transform the relationship to unidirectional common association as an example shown in Figure 4.7 before measuring understandability using the same formula.

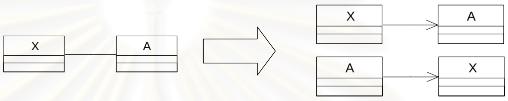
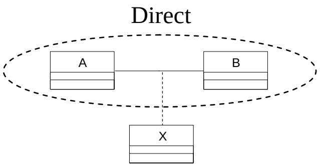


Figure 4.7: Transforming bidirectional association to unidirectional association.

Understandability Related to Association Class

Information relevant to the association roles cannot always reside with the classes involved in the association. In this situation, an association class may be used to model the relationship. In order to measuring understandability, we will transform association class to unidirectional association form. Then we will consider each pair of relationship. Figure 4.8 shows an example of classes related to class X by direct association class and Figure 10 shows an example of classes related to class X by indirect association class. In order to measure understandability related to direct and indirect association class of class X, Figure 4.8 will be transformed to Figure 4.9, and Figure 4.10 will be transformed to Figure 4.11. Then, $\text{Und}_{\text{AssocC}}$ will be measured from Consider boxes of Figure 4.9 and Figure 4.11. For example in Consider box of Figure 4.9, if we want to understand class X, we should also read class A and class B. $\text{Und}_{\text{AssocC}}$ (X) is defined in similar way of defining $\text{Und}_{\text{CAssoc}}$ as follows.

$$\text{Und}_{\text{AssocC}}\,(X) = W_{\text{AssocC}}\,[\,C(\text{ImHead}_{\text{AssocC}}\,(X)) + C(\text{ImHead}_{\text{AssocC}}\,(\text{AllHead}_{\text{Gen}}(X)))\,]$$



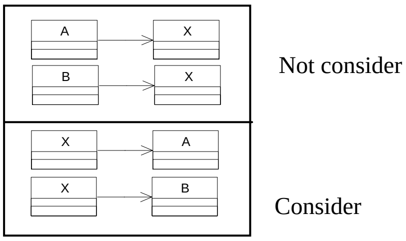


Figure 4.8: Direct association class.

Figure 4.9: Transformed direct association class.

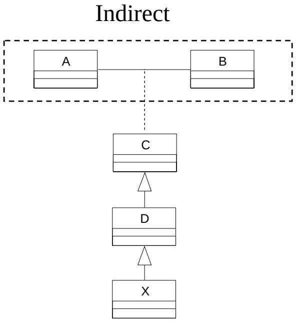


Figure 4.10: Indirect association class.

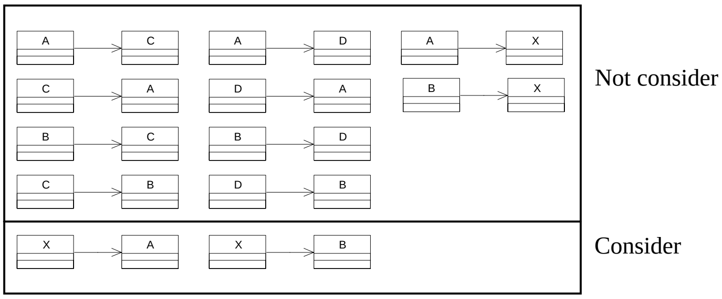


Figure 4.11: Transformed indirect association class.

Understandability Related to Dependency

Anytime a class uses another class in some fashion, a dependency exists between the two. The relationship is that of the user depending on the class that it is using. A dependency exists if a class has a local variable based on another class, a reference to an object directly, or a reference to an object indirectly, for example via some operation parameters, or uses a class's static operation.

Consider classes related to class X by dependency. If class X directly depended on any classes, we should also understand these classes as an example shown in dotted oval area of Figure 4.12. These classes can be represented by $ImHead_{Dep}(X)$. Moreover, we should understand all classes related to class X by indirect dependency as an example shown in dotted rectangle of Figure 4.12. These classes can be represented by $ImHead_{Dep}(AllHead_{Gen}(X))$. $Und_{Dep}(X)$ is defined as follows.

$$Und_{Dep}(X) = W_{Dep} [C(ImHead_{Dep}(X)) + C(ImHead_{Dep} (AllHead_{Gen}(X)))]$$

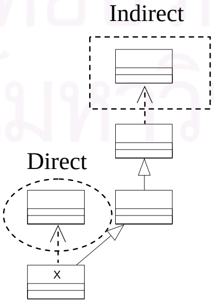


Figure 4.12: Dependency.

Understandability Related to Realization

Consider classes related to class X by realization. We should understand interface classes which are defined a set of functionalities as a contract and class X realizes that contract by implementing the functionality. The classes related to class X by direct realization are shown in dotted oval area of Figure 4.13. These classes can be represented by $\mathrm{ImHead}_{Real}(X)$. We should also understand classes related to class X by indirect realization as an example shown in dotted rectangle of Figure 4.13. These classes can be represented by $\mathrm{ImHead}_{Real}(\mathrm{AllHead}_{Gen}\ (X))$. $\mathrm{Und}_{Real}(X)$ is defined as follows.

$$\mathrm{Und}_{Real}(X) = W_{Real}\ [\ C(\mathrm{ImHead}_{Real}(X)) + C(\mathrm{ImHead}_{Real}\ (\mathrm{AllHead}_{Gen}(X)))\ ]$$

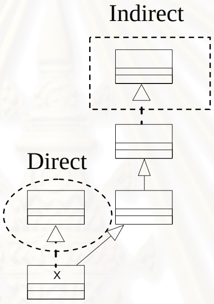


Figure 4.13: Realization.

4.2.1.2 Understandability of a Software

Understandability of a software will be calculated from understandability of all classes in the software. Generally, large software is more complex than small one. Accordingly, it is more reasonable to compare understandability of a software to understandability of another software of the same size. For comparing understandability of software with different size, we should introduce the concept of averages. The average degree of understandability of a software is defined as follows.

$$AvgUnd_{Sys}(S) = \left(\frac{\sum_{i=1}^{n} Und_{Class}(X_i)}{n} \right)$$

Where X_i is a class of software S; i = 1,2,…,n.

n is the total number of classes of software S.

4.2.2. Metrics for Modifiability

4.2.2.1 Modifiability of a Class

Metrics for modifiability of a class proposed with an idea that if we want to modify class X, class X will be modified. Moreover, if class X affects other classes, these classes will be modified too. In the best case, only class X will need to be modified. In the worst case, class X and all classes affected from class X must be modified. In the average case, class X will be modified and half of classes affected from class X should be modified. In this work, modifiability will be measured in the average case.

In order to measure modifiability of class X, we will consider the efforts put for modifying class X and modifying classes affected from class X.

In this work, the effort put for modifying class X is represented by complexity of class X or C(X) because if class X has high complexity, we should put much effort to modify it. The effort put for modifying classes related to class X will be represented by 7 metrics: Mod_{Gen}, Mod_{Agg}, Mod_{Com}, Mod_{CAssoc}, Mod_{AssocC}, Mod_{Dep} and Mod_{Real}. They are calculated according to the kind of relationships that class X and the others are related.

The degree of modifiability of class X is summation of the efforts put for modifying class X and modifying classes related to class X as follows.

$$Mod_{Class}(X) = C(X) + Mod_{Gen}(X) + Mod_{Agg}(X) + Mod_{Com}(X) + Mod_{CAssoc}(X) + Mod_{AssocC}(X) + Mod_{Dep}(X) + Mod_{Real}(X)$$

Modifiability Related to Generalization

Consider classes related to class X by generalization. If we modified class X, we may need to modify descendant classes of class X. These classes can be represented by $AllTail_{Gen}(X)$ as an example shown in Figure 4.14. In the average case, half of them will be modified. Mod_{Gen} can be defined in expression of multiplication between dependency weight value of generalization and complexity of classes related to generalization considering in the average case as follows.

$$Mod_{Gen}(X) = W_{Gen} C(AllTail_{Gen}(X)) / 2$$

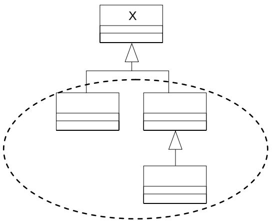


Figure 4.14: Generalization.

Modifiability Related to Aggregation

Consider classes related to class X by aggregation. If we modify class X, classes which compose of class X may be modified. These classes can be represented by $ImHead_{Agg}$ (X) as an example shown in dotted oval area of Figure 4.15. Classes in dotted rectangle area of Figure 4.15 are classes related to class X by indirect aggregation. If class X is changed, they may be affected. These classes can be represented by $AllTail_{Gen}$ ($ImHead_{Agg}$ (X)). Considering in the average case, Mod_{Agg}(X) is defined as follows.

$$Mod_{Agg}(X) = W_{Agg} [C(ImHead_{Agg}(X)) + C(AllTail_{Gen} (ImHead_{Agg} (X)))] /2$$

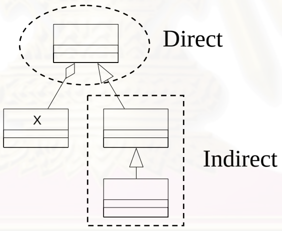


Figure 4.15: Aggregation.

Modifiability Related to Composition

Classes affected from class X by composition can be considered similar to classes affected from class X by aggregation. Considering in the average case, Mod_{Com}(X) is defined as follows.

$$Mod_{Com}(X) = W_{Com} [C(ImHead_{Com}(X)) + C(AllTail_{Gen} (ImHead_{Com} (X)))] /2$$

Modifiability Related to Common Association

Consider classes related to class X by unidirectional common association. If we modify class X, classes which directly invoke methods of class X may be affected. These classes can be represented by $ImTail_{CAssoc}$(X) as an example shown in dotted oval area of Figure 4.16. Furthermore, classes related to class X by indirect common association may be affected as an example shown in dotted rectangle area of Figure 4.16. These classes can be represented by $\text{AllTail}_{\text{Gen}}$ ($\text{ImTail}_{\text{CAssoc}}$ (X)). Considering in the average case, $\text{Mod}_{\text{CAssoc}}$ (X) is defined as follows.

$$\text{Mod}_{\text{CAssoc}}(X) = W_{\text{CAssoc}} \, [\, C(\text{ImTail}_{\text{CAssoc}}(X)) + C(\text{AllTail}_{\text{Gen}}(\text{ImTail}_{\text{CAssoc}}(X))) \,] /2$$

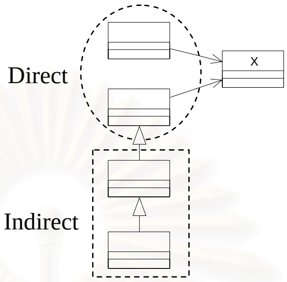


Figure 4.16: Common association.

For bidirectional common association, we will transform the relationship to unidirectional common association before measuring modifiability with the same approach.

Modifiability Related to Association Class

For classes related to class X by association class, we will transform association class to unidirectional association form. Then we will consider each pair of relationships. For examples, in order to measure modifiability related to direct and indirect association class of class X, Figure 4.17 will be transformed to Figure 4.18, and Figure 4.19 will be transformed to Figure 4.20. Then $\text{Mod}_{\text{AssocC}}$ will be measured from Consider boxes of Figure 4.18 and Figure 4.20. For example in Consider box of Figure 4.18, if we want to modify class X, we may also modify class A and B. $\text{Mod}_{\text{AssocC}}$ (X) is defined in similar way of defining $\text{Mod}_{\text{AssocC}}$ as follows.

$$\text{Mod}_{\text{AssocC}}(X) = W_{\text{AssocC}} \, [\, C(\text{ImTail}_{\text{AssocC}}(X)) + C(\text{AllTail}_{\text{Gen}}(\text{ImTail}_{\text{AssocC}}(X))) \,] /2$$

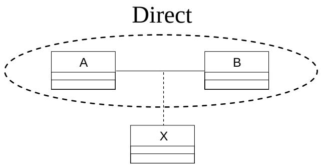


Figure 4.17: Direct association class.

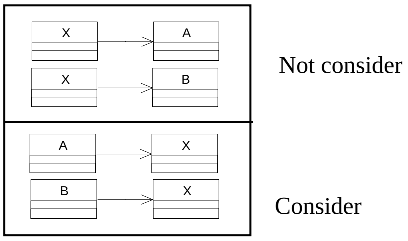


Figure 4.18: Transformed direct association class.

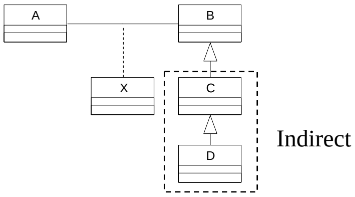


Figure 4.19: Indirect association class.

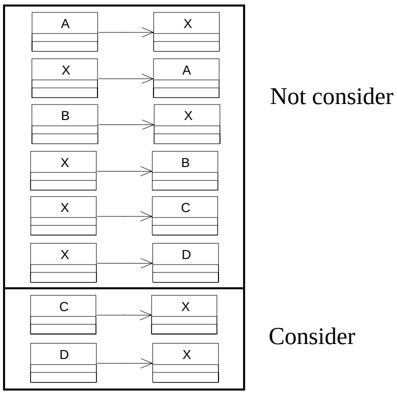


Figure 4.20: Transformed indirect association class.

Modifiability Related to Dependency

Consider classes related to class X by dependency. If we modify class X, classes depending on class X may be modified. These classes can be represented by $ImTail_{Dep}(X)$ as an example shown in dotted oval area of Figure 4.21. Furthermore, classes related to class X by indirect dependency may be modified as an example shown in dotted rectangle area of Figure 4.21. These classes can be represented by $AllTail_{Gen}$ $(ImTail_{Dep}$ $(X))$. Considering in the average case, $Mod_{Dep}(X)$ is defined as follows.

$$Mod_{Dep}(X) = W_{Dep} [C(ImTail_{Dep}(X)) + C(AllTail_{Gen}(ImTail_{Dep}(X)))] /2$$

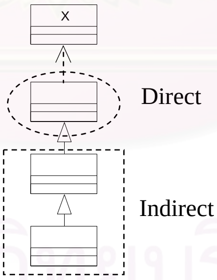


Figure 4.21: Dependency.

Modifiability Related to Realization

Consider classes related to class X by realization. If we modify class X, implementation classes of class X must be modified. These classes can be represented by $ImTail_{Real}(X)$ as an example shown in dotted oval area of Figure 4.22. Furthermore, classes related to class X by indirect realization may be modified as an example shown in dotted rectangle area of Figure 4.22. These classes can be represented by $AllTail_{Gen}$ $(ImTail_{Real}$ $(X))$. Considering in the average case, $Mod_{Real}(X)$ is defined as follows.

$$Mod_{Real}(X) = W_{Real} [C(ImTail_{Real}(X) (X)) + C(AllTail_{Gen} (ImTail_{Real} (X)))] /2$$

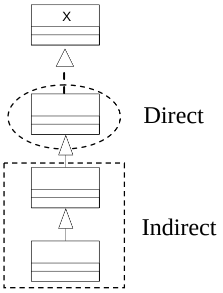


Figure 4.22: Realization.

4.2.2.2 Modifiability of a Software

Modifiability of a software will be calculated from modifiability of all classes in the software. It is defined with concept of average as same as understandability. The average degree of modifiability of a software is defined as follows.

$$AvgMod_{Sys}(S) = \left(\frac{\sum_{i=1}^{n} Mod_{Class}(X_i)}{n} \right)$$

Where X_i is a class of software S; i = 1,2,…,n.

n is the total number of classes of software S.

4.3 Metric Validation

The previous section introduced metrics for understandability and modifiability, which still needs empirical validation in order to validate their usability. Generally, a metric will be invalid in practice while be valid in theoretical argument, and vice versa. Whether a metric is valid depends on whether it is consistent with human beings' intuition or not. This section describes validating the proposed metrics.

In section 3.2, we presented an experiment captured degree of understandability and modifiability of 40 software in human beings' view. We collected the degree of understandability in terms of Understandability score and the degree of modifiability in terms of Modifiability score. In order to validate the proposed metrics, the degree of understandability and modifiability of the same 40 software were measured using the proposed metrics: $AvgUnd_{Sys}$ and $AvgUnd_{Sys}$. To find out the correlation between the degree of understandability and modifiability measured by the proposed metrics and human beings' intuition, 2 hypotheses were formulated:

$H1_0$: There is no correlation between $AvgUnd_{Sys}$ and Understandability score.

$H1_A$: There is correlation between $AvgUnd_{Sys}$ and Understandability score.

$H2_0$: There is no correlation between $AvgMod_{Sys}$ and Modifiability score.

$H2_A$: There is correlation between $AvgMod_{Sys}$ and Modifiability score.

In order to test both hypotheses, the Pearson's correlation test was applied. The correlation between two variables reflects the degree to which the variables are related. Correlation value ranges from -1 to +1. Value of 1 means that there is a perfect positive relationship between both variables. Value of -1 indicates a perfect negative relationship and value of 0 indicates no relationship. The result of correlation analysis is shown in Table 4.2 and Table 4.3.

The result shown in Table 4.2 and Table 4.3 can be concluded that $H1_0$ and $H2_0$ are rejected. The correlation values of -0.651 and –0.685 indicate that

- $AvgUnd_{Sys}$ and Understandability score have negative correlation, and
- $AvgMod_{Sys}$ and Modifiability score have negative correlation.

So, this result can be concluded that new proposed metrics are correlated with human beings' intuition at significant level 0.01.

Table 4.2: Correlation relationships between $AvgUnd_{Sys}$ and Understandability score.

		$AvgUnd_{Sys}$	UndScore
$AvgUnd_{Sys}$	Pearson Correlation	1	-.651(**)
	Sig. (2-tailed)	.	.000
	N	40	40
UndScore	Pearson Correlation	-.651(**)	1
	Sig. (2-tailed)	.000	.
	N	40	40

** Correlation is significant at the 0.01 level (2-tailed).

Table 4.3: Correlation relationships between AvgMod$_{Sys}$ and Modifiability score.

		AvgMod$_{Sys}$	ModScore
AvgMod$_{Sys}$	Pearson Correlation	1	-.685(**)
	Sig. (2-tailed)	.	.000
	N	40	40
ModScore	Pearson Correlation	-.685(**)	1
	Sig. (2-tailed)	.000	.
	N	40	40

** Correlation is significant at the 0.01 level (2-tailed).

Understandability and modifiability captured by two new proposed metrics and by human beings' intuition have negative correlation because of the following reason. AvgUnd$_{Sys}$ and AvgMod$_{Sys}$ measure the degree of understandability and modifiability of a software considering the efforts put for understanding and for modifying the software. Understandability and Modifiability scores capture the degree of understandability and modifiability of a software using examinations. If the software is easy to understand and modify, AvgUnd$_{Sys}$ and AvgMod$_{Sys}$ values will be low but Understandability and Modifiability scores will be high. In contrast, if the software is difficult to understand and modify, AvgUnd$_{Sys}$ and AvgMod$_{Sys}$ values will be high but Understandability and Modifiability scores will be low.

4.4 Metric Threshold

To estimate thresholds or value ranges of AvgUnd$_{Sys}$ that lie in 3 understandability levels: easy, medium and difficult, we calculate lower confidence limit (L) and upper confidence limit (U) values of mean (μ)of population of AvgUnd$_{Sys}$ for each understandability levels. Range of mean of population can be expressed as the follows [45].

$$L < \mu < U$$

Where L and U can be computed from the following formula.

$$\bar{x} - Z_{1-\alpha/2}\frac{\sigma}{\sqrt{n}} < \mu < \bar{x} + Z_{1-\alpha/2}\frac{\sigma}{\sqrt{n}}$$

where $\bar{x}$ is the mean of samples,

$Z_{1-\alpha/2}$ is the probability value in Z table, and

$\frac{\sigma}{\sqrt{n}}$ is the standard deviation.

At significant level(σ) 0.05, we compute lower and upper confidence limit values: L2 and U2, L1 and U1, and L0 and U0, for easy, medium and difficult levels of understandability respectively as shown in Figure 4.23.

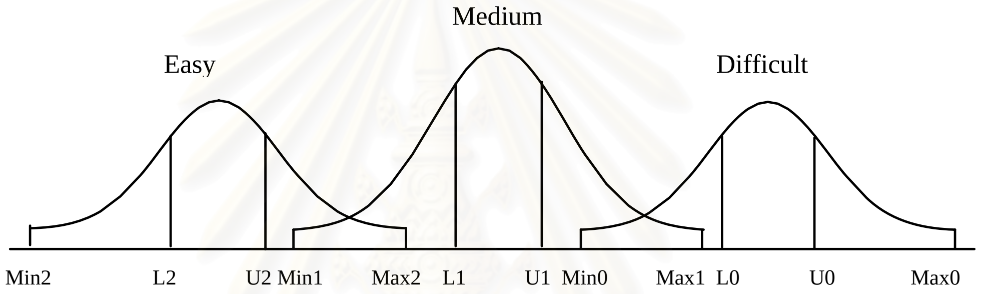


Figure 4.23: Lower and upper confidence limit values.

From this figure, we can imply that if a software has $\text{AvgUnd}_{\text{Sys}}$ < U2, it should be classified to easy level of understandability. If a software has $\text{AvgUnd}_{\text{Sys}}$ > L0, it should be classified to difficult level of understandability. Estimating value ranges of $\text{AvgMod}_{\text{Sys}}$ can be performed in the same way. The preliminary result obtained from our experimental data is shown in Table 4.4

Table 4.4: Value ranges of $\text{AvgUnd}_{\text{Sys}}$ and $\text{AvgMod}_{\text{Sys}}$.

Metric	Level	Value range
$\text{AvgUnd}_{\text{Sys}}$ (understandability)	Easy	<21
	Medium	21 - 33
	Difficult	>33
$\text{AvgMod}_{\text{Sys}}$ (modifiability)	Easy	<13
	Medium	13-18
	Difficult	>18

These thresholds are used to classify understandability and modifiability levels of 40 sample software design models. The result shows that the accuracy of AvgUnd$_{Sys}$ is 95% and the accuracy of AvgMod$_{Sys}$ is 97.5%. Although the accuracy of AvgUnd$_{Sys}$ and AvgMod$_{Sys}$ is less than 100% which is the accuracy of MLPUnd2 and MLPMod1, the best prediction models for understandability and modifiability presented in Chapter III, MLPUnd1 and MLPMod2 use 17 metrics and 13 metrics for classifying understandability and modifiability. For future work, the experiment should be repeated with more number of sample software design models in order to improve the threshold accuracy of AvgUnd$_{Sys}$ and AvgMod$_{Sys}$.

CHAPTER V

CONCLUSION AND FUTURE WORK

This chapter concludes the research work and presents some directions for the future work.

5.1 Conclusion

Software quality has become essential to good software development. One quality which should be concerned in early phase is maintainability. Predicting maintainability in early phase will help software designers to alter the design of the software for better performance which will lead to the ease of implementation and reduction of maintenance cost.

This thesis selects two metric sets: structural complexity and aesthetic metrics. These metrics are expected that they can be predictors of maintainability and its two sub-characteristics: understandability and modifiability. MANOVA test are performed to validate the expectation. The result shows that the structural complexity metrics and the aesthetic metrics can be good indicators of understandability, modifiability and maintainability.

Understandability, modifiability and maintainability prediction models are constructed from structural complexity and aesthetic metrics applying 3 techniques called Discriminant analysis, Decision tree and MLP neural network. The experimental result shows that the model accuracy of every prediction model obtained from MLP neural network is higher than or equal to that of the prediction models obtained from Discriminant analysis and Decision tree. The trade-off is that the number of metrics used in every prediction model obtained from MLP neural network is greater than or equal to that of the prediction models obtained from Discriminant analysis and Decision tree. However, if all metrics used in prediction model can be measured automatically, the problem of using greater number of metrics can be solved.

In case of automated utilizing the prediction models, the experimental result can be conclude that best prediction models for understandability, modifiability and maintainability are MLPUnd1, MLPMod2 and MLPMod3 respectively.

An automated tool for measuring structural complexity metrics from UML class and sequence diagrams is also constructed and this tool can predict understandability, modifiability and maintainability utilizing the prediction models obtained from the experiment.

This thesis also proposes two set of structural complexity metrics to assess understandability and modifiability objectively. These metrics are developed considering the number and the kind of relationships among classes. To validate the new proposed metrics, correlation analysis between understandability and modifiability measured by the proposed metrics and measured by human beings' intuition are performed. The result from the experiment can be concluded that the new proposed metrics are significantly correlated with human beings' intuition at significant level 0.01. The new proposed metrics could be used as early maintainability indicators.

5.2 Future Work

1. The experiment uses only forty simple software design models. This is a limitation of the research, since it is difficult to find software design models in real world software. The experiment should be repeated with more number of sample software design models in order to increase reliability of experimental result.
2. Size of software design models should be increased. By increasing size of software design models, we will have examples that are closer to reality. In addition, if we are working with professionals, we can make better use of their potential capability and conclude that the results are more general.
3. This work considers only 2 types of UML diagrams: class and sequence diagrams. It may be considered as the preliminary approach for constructing maintainability models from UML diagrams. For future work, other type of UML diagrams may be considered.

4. Understandability, modifiability and maintainability levels can be tried on more than 3 levels.
5. The understandability, modifiability and maintainability models obtained from the experiment can predict understandability, modifiability and maintainability levels of a software design model, but they cannot suggest how to improve understandability, modifiability and maintainability. Research on improving these qualities should be further considered.
6. Concerns with the new proposed objective structural complexity metrics presented in Chapter IV, the $AvgUnd_{Sys}$ and $AvgMod_{Sys}$ metrics are developed in order to measure the maintainability of class diagram during the design phase of the software life cycle. One function used for computing $AvgUnd_{Sys}$ and $AvgMod_{Sys}$ is the complexity of a class. The metrics compute the complexity of a class roughly from the summation of the number of attributes and the number of methods in the class. These metrics can be extended to comprehend the complexity of a class. For example, visibility of attributes and methods (i.e. public, protected and private) should be considered.
7. As mentioned in heading 4.1.3, we can order dependency degree of each relationship but we don’t know exact weight values. Ideally, these weight values should be assigned by experts or captured from empirical study with enough supported data. In this work, each relationship is given a weight value: W_{Dep} =1, W_{CAssoc} = 2, W_{AssocC} = 3, W_{Agg} = 4, W_{Com} = 5, W_{Gen} = 6, and W_{Real} = 7. Different weight values should be used in the future experiment to find proper dependency weight values of each relationship.
8. An automated tool for measuring the proposed objective structural complexity metrics for understandability and modifiability should be constructed.

REFERENCES

(1) Brantley, C. L. and Osajima, Y. R. Continuing Development of Centrally Developed and Maintained Software Systems. IEEE Computer Society, 45,1(1975):285-288.

(2) Riggs, R. Computer System Maintenance. Datamation, November 15, 1969:227-235.

(3) Sterns, S. Experience with Centralized Maintenance of a Large Application System. IEEE Computer Society, 45,3(1975):114-120.

(4) Bandi, R., Vaishnavi, V. and Turk, D. Predicting Maintenance Performance Using Object-Oriented Design Complexity Metrics. IEEE Transactions on Software Engineering, 29,1(2003):77-87.

(5) Genero, M., Piattini, M. and Calero, C. Early Measures for UML Class Diagrams. L' Object, 6,4(2000):489-515.

(6) Kim, H. and Boldyreff, C. Developing Software Metrics Applicable to UML Models. Proceedings of the 6th ECOOP Workshop on Quantitative Approaches in Objected-Oriented Software Engineering, June 11, 2002.

(7) Genero, M., Miranda, D. and Piattini, M. Defining and Validating Metrics for UML Statechart Diagrams. Proceedings of the 6th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering, 11 June 2002.

(8) Sheldon, F. T., Jerath, K. and Chung, H. Metrics for Maintainability of Class Inheritance Hierarchies. Journal of Software Maintenance and Evolution: Research and Practice, 14,1(2002):147-160.

(9) Booch, G., Rumbuagh, J. and Jacobson, I. The Unified Modeling Language User Guide. Addison Wesley Longman,1999.

(10) Fenton, N. E. and Pfleeger, S. L. Software Metrics: A Rigorous and Practical Approach. PWS Publishing, 1997.

(11) ISO/IEC: Standard 9126-Software Product Evaluation-Quality Characteristics and Guidelines for Their Use. Geneva, 1995.

(12) Harris, R. J. A Primer of Multivariate Statistics. Third Edition, Lawrence Erlbaum Associates, 2001.

(13) Tabachnick, B. G. and Fidell, L. S. Using Multivariate Statistics. Allyn & Bacon, 2001.

(14) Mitchell, T. M. Machine Learning. The McGraw-Hill Companies, Inc., 1997.

(15) Haykin, S. Neural Networks: A Comprehensive Foundation. Prentic-Hall, Inc.1990.

(16) Briand, L. C., Bunse, C., Daly, J. W. and Differding C. An Experimental Comparison of the Maintainability of Object-Oriented and Structured Design Document. Proceedings of International Conference on Software Maintenance, 1977:130-138.

(17) Briand, L. C., Bunse, C. and Daly, J. W., A Controlled Experiment for Evaluating Quality Guidelines on the Maintainability of Object-Oriented Designs. IEEE Transactions on Software Engineering, 27,6(2001):513-530.

(18) Deligiannis, I., Stamelos, I., Angelis, L., Roumeliotis, M. and Shepperd, M., A Controlled Experiment Investigation of an Object-Orited Design Heuristic for Maintainability. ESERG Technical Reports, ESERG: Empircal Software Engineering Research Group at Bournemouth University, May 22, 2002.

(19) Daly, J. and Brook, A. The Effect of Inheritance on the Maintainability of Object-Oriented Software: An Empirical Study. Proceedings of the International Conference on Software Maintenance, 1995.

(20) Harrison, R. and Counsell, S. The Role of Inheritance in Maintainability of Object-Oriented Systems. Proceedings of the 11th European Software Control and Metrics Conference, Munich, Germany, April 2000.

(21) Binkley, A. and Schach, S. Inheritance-Based Metrics for Predicting Maintenance Effort: an Empirical Study. Technical Report TR97-05, Computer Science Department, Vanderbilt University, 1997.

(22) Wood, M., Daly, J. and Miller, J. Multi-Method Research: an Empirical Investigation of Object-Oriented Technology. Journal of System and Software, 48,1(1999):13-26.

(23) Harrison, R., Counsell, S. and Nithi, R. Experimental Assessment of the Effect of Inheritance on the Maintainability of Object-Oriented Systems. Journal of Systems and Software, 44,2(2000):173-176.

(24) Fioravanto, F. and Nesi, P. Estimation and Prediction Metrics for Adaptive Maintenance Effort of Object-Oriented Systems. IEEE Transactions on software Engineering, 27,12(2001):1062-1084.

(25) Prechelt, L., Unger, B., Phillippsen, M. and Tichy, W. A Controlled Experiment on Inheritance Depth as a Cost Factor for Code Maintenance. Journal of Systems and Software, 65,2(2003):115-126.

(26) Genero, M., Jiménez, L. and Piattini, M. A Controlled Experiment for Validating Class Diagram Structural Complexity Metrics. Proceedings of the 8th International Conference on Object-Oriented Information Systems, September 2002:372-383.

(27) Purchase, H. C., Mcgill, M., Colpoys, L. and Carrington, D. Graph Drawing Aesthetics and the Comprehension of UML Class Diagrams: an Empirical Study. Proceedings of the Australian Symposium on Information Visualisation, Australian Computer Society, 2001.

(28) Purchase, H. C., Colpoys, L. and Mcgill, M., UML Class Diagram Syntax: An Empirical Study of Comprehension. Proceedings of the Australian Symposium on Information Visualisation, Sydney, Australia, 2001.

(29) Purchase, H. C., Allder, J. and Carrington, D. Graph Layout Aesthetic in UML Diagrams: User Preferences. Journal of Graph Algorithms and Applications, 6,3(2002):255-279.

(30) Eichelberger, H. Aesthetics of Class Diagrams. Proceedings of the 1st IEEE Interation Workshop on Visualizing Software for Understanding and Analysis, Paris, France, 2002:23-31.

(31) Gutwenger, C., Junger, M., Klein, K., Kupke, J., Leipert, S. and Mutzel, P. A New Approach for Visualizing UML Class Diagrams. Proceedings of the 2003 ACM Symposium on Software Visualization, ACM Press, 2003:179-188.

(32) Poranen, T., Makinen, E. and Nummenmaa, J. How to Draw a Sequence Diagram. Proceedings of the 8th Symposium on Programming Languages and Software Tools, 2003.

(33) Genero, M., Olivas, J., Piattini, M., and Romero, F. Using Metrics to predict OO Information Systems Maintainability. Proceedings of the 13th International Conference on Advanced Information Systems Engineering (CAiSE 2001), Interlaken, Switzerland, June 4-8, 2001.

(34) Battista, G. D., Eades, P., Tamassia, R. and Tollis, I.G. Graph Drawing. Prentice Hall, 1999.

(35) Purchase, H. C., Cohen, R. F. and James, M. Validating Graph Drawing Aesthetics. Lecture Notes in Computer Science, 1027,1(1996):435-446.

(36) Purchase, H. C., Cohen, R. F. and James, M. An Experimental Study of the Basis for Graph Drawing Algorithms. Journal of Experimental Algorithmics (JEA), 2,4(1997):111-119.

(37) Eichelberger, H. Nice class diagrams admit good design? Proceedings of the 2003 ACM Symposium on Software Visualization, ACM Press, 2003:159–168.

(38) Genero, M., Paittini, M. and Manso, E. Finding 'Early' Indicators of UML Class Diagrams Understandability and Modifiability. Proceedings of the 2004 International Symposium on Empirical Software Engineering, 2004.

(39) Weiss, S. and Kulikowski, C. Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems, Morgan Kaufmann Publishers, 1991.

(40) Genero, M., Piattini, M., Manso, E. and Cantone, G. Building UML Class Diagram Maintainability Prediction Models Based on Early Metrics. Proceedings of the 9th International Software Metrics Symposium, 2003.

(41) Cartwright, M. An Empirical View of Inheritance. Information and Software Technology, 1998:795-799.

(42) Program Committee of the Institute for Objective Measurement. Definition of Objective Measurement. Available from: http://www.rasch.org/define.htm [2007, 1 January]

(43) Sheldon, F. T., Jerath, K. and Chung, H. Metrics for Maintainability of Class Inheritance Hierarchies. Journal of Software Maintenance and Evolution: Research and Practice, 14,1(2002):147-160.

(44) Kang, D., Xu, B., Lu, J. and Chu, W. C. A Complexity Measure for Ontology Based on UML. Proceedings of the 10th IEEE International Workshop on Future Trends of Distrubuted Computing Systems, Suzhou, Chaina, May 2004:222-228.

(45) กัลยา วานิชย์บัญชา. การวิเคราะห์สถิติ: สถิติสำหรับการบริหารและวิจัย. โรงพิมพ์แห่งจุฬาลงกรณ์มหาวิทยาลัย, 2546.

(46) Rovinelli, R. J. and Hambleton, R. K. On the Use of Content Specialists in the Assessment of Criterion-Referenced test Item Validity. Dutch Journal of Educational Research, 2,1(1977):49-60.

(47) พิชิต ฤทธิ์จรูญ. หลักการวัดและประเมินผลการศึกษา. กรุงเทพฯ: เฮ้าส์ ออฟ เคอร์มีสท์, 2548.

APPENDICES

APPENDIX A

PUBLICATIONS

A.1 International Journal

1) Kiewkanya, M. and Muenchaisri, P. Measuring Maintainability in Early Phase Using Aesthetic Metrics. WSEAS Transaction on Computers, 4,2(2005): 227-232.

A.2 International Conferences

1) Kiewkanya, M. and Muenchaisri, P. Measuring Maintainability in Early Phase Using Aesthetic Metrics. Proceedings of the 6th WSEAS International Conference on Software Engineering, Parallel & Distributed Systems (SEPADS 2005), Salzburg, Austria, February 13-15, 2005.
2) Kiewkanya, M. and Muenchaisri, P. Predicting Modifiability of UML Class and Sequence Diagrams. Proceedings of the 2nd Workshop on Software Quality in conjunction with the 26th International Conference on Software Engineering (ICSE 2004), Edinburgh, Scotland, UK, May 24-28, 2004.
3) Jindasawat, N., Kiewkanya, M. and Muenchaisri, P. Investigating Correlation between the Object-Oriented Design Maintainability and Two Sub-Characterictics: Understandability and Modifiability. Proceedings of the 13th ISCA International Conference on Intelligent and Adaptive Systems and Software Engineering (IASSE 2004), Nice, France, July 1-3, 2004.
4) Kiewkanya, M., Jindasawat, N. and Muenchaisri, P. A Methodology for Constructing Maintainability Model of Object-Oriented Design. Proceedings of the 4th International Conference on Quality Software (QSIC 2004), Braunschweig, Germany, September 8-10, 2004.
5) Kiewkanya, M., Jindasawat, N., Prompoon, N. and Muenchaisri, P. Constructing Understanding Model using Design Metrics. Proceedings of the 2003 International Conference on Software Engineering and Knowledge

Engineering (SEKE 2003), Hotel Sofitel San Francisco Bay, CA, USA, July 1-3, 2003.

6) Kiewkanya, M. and Muenchaisri, P. Measuring Internal Reuse of Object-Oriented Software Using Metrics Applicable to UML Class and Sequence Diagrams. Proceedings of the ISCA 12th International Conference on Intelligent and Adaptive Systems and Software Engineering (IASSE-2003), Canterbury hotel, San Francisco, CA, USA, July 9-11, 2003.

APPENDIX B

EXAMINATIONS

B.1 Validating the Examinations

The development of measuring instruments is a process which includes both a) the development of the item and subscale components and b) the qualitative and quantitative assessments of the item and subscale parameters. In order to appropriately use and interpret data obtained from a measuring instrument, there must be operational definitions of the constructs being measured and information on the reliability and validity of the scores. This information assists users in placing appropriate meaning to the results obtained and interpreting the scores within the confines of the assessment parameters identified.

Content validity is the degree of confidence that the items are measuring what they are intended to measure. Evidence of content validity can be obtained from an evaluation, conducted by independent experts, of the effectiveness of items in measuring one or more objectives. An efficient measure for numerically assessing content experts' evaluations of items is *the index of item-objective congruence (IOC)* [46].

An evaluation using the IOC is a process where content experts, at least 3 experts [47], rate individual items on the degree to which they do or do not measure specific objectives listed by the test developer. More specifically, a content expert will evaluate each item by giving the item a rating of 1 (for clearly measuring), -1 (clearly not measuring), or 0 (degree to which it measures the content area is unclear) for each objectives. The experts are not told which constructs the individual items are intended to measure, thus they remain independent and unbiased evaluators.

After the experts complete an evaluation of the items, the ratings are combined to provide indices of item-objective congruence measures for each item on each objective. The range of the index score for an item is -1 to 1 where a value of 1 indicates that all experts agree that the item is clearly measuring only the objective that it is hypothesized to measure and is clearly not measuring any other objective. A value of -1 would indicate that the experts believe the item is measuring all objectives that it was not defined to measure and is not measuring the hypothesized objective. The item will be accepted if its index score is greater than or equal 0.5.

The IOC is computed using the following equation.

$$IOC = \frac{\sum R}{N}$$

where IOC is consistency between item & the objective,

R is sum of scores from experts,

N is the number of experts.

B.2 Applying the IOC to the Examinations

In order to validate the examinations used in this work applying the IOC, we prepare validation documents including of document for validating the template questions shown in Table B.1 – Table B.8 and general description, class diagram and sequence diagrams of a sample software named ‘Online CD shop system’ shown in Table B.10 and Figure B.1 – Figure B.3. In this work, we cannot validate examination questions of all software design models (40 software) because of a lot of examination questions. However, examination questions of each software design model are constructed from the same template questions. So, we decide to validate the template questions instead. The validation documents are sent to three experts in modeling with UML. The IOC values of all template questions obtained from all evaluators are shown in Table B.9. The result shows that all template questions are accepted.

Table B.1: Validation form of template questions for assessing understandability of class diagram.

คำถามวัดความสามารถในการทำความเข้าใจแผนภาพคลาส								
ส่วนประกอบ	วัตถุประสงค์	หมายเลขคำถามต้นแบบ	คำถามต้นแบบ	ตัวอย่างคำถาม จากระบบการขายซีดีออนไลน์	คำตอบ	ความสอดคล้องของรูปแบบคำถามกับวัตถุประสงค์		
						สอดคล้อง	ไม่แน่ใจ	ไม่สอดคล้อง
Classes	เพื่อวัดความเข้าใจโดยรวมเกี่ยวกับคลาส	UC1	คลาส C สามารถให้บริการ อะไรได้บ้าง	คลาส Order สามารถให้บริการอะไรได้บ้าง	สร้างและแสดงผลรายการสั่งซื้อ			
Attributes	เพื่อวัดความเข้าใจเกี่ยวกับแอตทริบิวต์ ได้แก่ ข้อมูลที่จัดเก็บ ค่าข้อมูล การเข้าถึง และ การมีอยู่ของแอทริบิวต์ในคลาสซึ่งเป็นผลมาจากการสืบทอดคุณสมบัติ	UA1	แอทริบิวต์ A ของคลาส C สามารถเรียกใช้หรือแก้ไขจากคลาสอื่นโดยตรงได้หรือไม่	แอทริบิวต์ artist ของคลาส CD สามารถถูกเรียกใช้หรือแก้ไขจากคลาสอื่นโดยตรงได้หรือไม่ เพราะเหตุใด	ไม่ได้ เพราะเป็น private			
		UA2	แอทริบิวต์ใดของคลาสใดบ้าง ที่มีผลต่อค่าของแอทริบิวต์ A ของคลาส C	แอทริบิวต์ใดของคลาสใดบ้าง ที่มีผลต่อค่าของแอทริบิวต์ totalPrice ของคลาส Order	shippingFee ของคลาส Order , price ของคลาส CD, amount ของคลาส OrderItem			
		UA3	คลาส C มีแอทริบิวต์อะไรบ้าง	คลาส Organization มีแอทริบิวต์อะไรบ้าง	aurthorizeName, customerID, customerName, password, email และ creditcardNo โดย 5 แอทริบิวต์หลังเป็นผลมาจากการสืบทอดคุณสมบัติจากคลาส Customer			

Table B.2: Validation form of template questions for assessing understandability of class diagram (continued).

คำถามวัดความสามารถในการทำความเข้าใจแผนภาพคลาส								
ส่วนประกอบ	วัตถุประสงค์	หมายเลขคำถามต้นแบบ	คำถามต้นแบบ	ตัวอย่างคำถามจากระบบการขายซีดีออนไลน์	คำตอบ	ความสอดคล้องของรูปแบบคำถามกับวัตถุประสงค์		
						สอดคล้อง	ไม่แน่ใจ	ไม่สอดคล้อง
Methods	เพื่อวัดความเข้าใจเกี่ยวกับเมทธอด ได้แก่ การทำงาน การเรียกใช้ และ การมีอยู่ของเมทธอดในคลาสซึ่งเป็นผลมาจากการสืบทอดคุณสมบัติ	UM1	เมทธอด M ของคลาส C ทำงานอะไร	เมทธอด Search() ของคลาส ArtistSearch ทำงานอะไร	ทำการค้นหาข้อมูล cd ตามชื่อศิลปิน			
		UM2	เมทธอด M ของคลาส C สามารถเรียกใช้จากคลาสอื่นโดยตรงได้หรือไม่	เมทธอด search() ของคลาส SearchREQ สามารถถูกเรียกใช้จากคลาสอื่นได้หรือไม่ เพราะเหตุใด	ได้ เพราะเป็น public			
		UM3	เมทธอดใดของคลาสใด ควรจะเรียกใช้เมทธอด M ของคลาส C	เมทธอดใดของคลาสใดควรจะเรียกใช้เมทธอด find_CD() ของคลาส CDList	เมทธอด search() ของคลาส SearchREQ			
		UM4	คลาส C มีเมทธอดอะไรบ้าง	คลาส Individual มีเมทธอดใดบ้าง	updateCustomer, order() และ search() โดยเมทธอดทั้งหมดเป็นผลมาจากการสืบทอดคุณสมบัติจากคลาส Customer			

Table B.3: Validation form of template questions for assessing understandability of class diagram (continued).

คำถามวัดความสามารถในการทำความเข้าใจแผนภาพคลาส								
ส่วนประกอบ	วัตถุประสงค์	หมายเลขคำถามต้นแบบ	คำถามต้นแบบ	ตัวอย่างคำถามจากระบบการขายซีดีออนไลน์	คำตอบ	ความสอดคล้องของรูปแบบคำถามกับวัตถุประสงค์		
						สอดคล้อง	ไม่แน่ใจ	ไม่สอดคล้อง
Relationships	เพื่อวัดความเข้าใจเกี่ยวกับความสัมพันธ์ประเภทต่าง ๆ และ multiplicity ของความสัมพันธ์	UR1	จงอธิบายความสัมพันธ์ระหว่างคลาส ที่มีความสัมพันธ์แบบ Generalization	จงอธิบายความสัมพันธ์ระหว่างคลาส CDCategories และ Jazz	หมวดหมู่ของซีดีมี 4 หมวด ซึ่ง Jazz เป็นหมวดหมู่หนึ่ง หรือ Jazz เป็นแบบเฉพาะของ CDCagegories นั่นเอง			
		UR2	จงอธิบายความสัมพันธ์ระหว่างคลาส ที่มีความสัมพันธ์แบบ Association	จงอธิบายความสัมพันธ์ระหว่างคลาส Supplier และ CD	Supplier จะเป็นผู้จัดส่งซีดีมาให้แก่ระบบ โดยสามารถส่งซีดีได้หลายแผ่นหรือไม่ส่งเลยก็ได้			
		UR3	จงอธิบายความสัมพันธ์ระหว่างคลาส ที่มีความสัมพันธ์แบบ Aggregation/Composition	จงอธิบายความสัมพันธ์ระหว่างคลาส ShoppingCart และ Order Item	ShoppingCart หนึ่งประกอบไปด้วย Order Item หลายรายการหรือไม่มีเลย			
		UR4	จากความสัมพันธ์ R ระหว่างคลาส C1 และคลาส C2 จงอธิบายความหมายของ multiplicity	จากความสัมพันธ์ Places ระหว่างคลาส Customer และคลาส Order ลูกค้าจะต้องสั่งซื้อซีดีอย่างน้อยกี่แผ่น	ลูกค้าไม่ต้องสั่งซื้อเลยก็ได้ (0 แผ่น) แต่เมื่อสั่งซื้อจะต้องซื้ออย่างน้อย 1 แผ่น			

Table B.4: Validation form of template questions for assessing understandability of sequence diagram.

คำถามวัดความสามารถในการทำความเข้าใจแผนภาพซีเควนซ์								
ส่วนประกอบ	วัตถุประสงค์	หมายเลขคำถามต้นแบบ	คำถามต้นแบบ	ตัวอย่างคำถาม จากระบบการขายซีดีออนไลน์	คำตอบ	ความสอดคล้องของรูปแบบคำถามกับวัตถุประสงค์		
						สอดคล้อง	ไม่แน่ใจ	ไม่สอดคล้อง
Messages	เพื่อวัดความเข้าใจเกี่ยวกับเมสเสจ ได้แก่ การทำงาน การเรียกใช้ และการคืนค่าแอทริบิวต์	UMs1	การทำงาน W มีเมสเสจใดบ้างที่เกี่ยวข้อง	จากแผนภาพซีเควนซ์ *การค้นหาและสั่งซื้อซีดี* ในการสร้างรายการสั่งซื้อ มีเมสเสจใดบ้างที่เกี่ยวข้อง	addCD() และ createOrder()			
		UMs2	เมสเสจใดบ้างที่สามารถมีการส่งซ้ำได้หลาย ๆ ครั้ง และมีความหมายว่าอย่างไร	จากแผนภาพซีเควนซ์ *การค้นหาและสั่งซื้อซีดี* เมสเสจใดที่สามารถมีการส่งซ้ำได้หลาย ๆ ครั้ง	createSR() หมายความว่าการค้นหาซีดีสามารถทำได้หลาย ๆ ครั้ง			
		UMs3	การเรียกใช้เมสเสจ สามารถสลับลำดับการเรียกใช้ได้หรือไม่	จากแผนภาพซีเควนซ์ *การรับข้อมูลการตลาด* การส่ง message get_reveiw(), get_artist() และ get_sample_clip สามารถสลับลำดับการเรียกใช้ได้หรือไม่	**ได้**			
		UMs4	ผลมาจากการเรียกใช้เมสเสจ M จะมีการคืนค่าแอทริบิวต์ใดกลับ	จากแผนภาพซีเควนซ์ *การรับข้อมูลการตลาด* ผลของการเรียกใช้เมสเสจ get_basic_info จะได้รับข้อมูล (แอทริบิวต์) ใดกลับมาบ้าง	**title, price และ artist**			

Table B.5: Validation form of template questions for assessing understandability of sequence diagram (continued).

คำถามวัดความสามารถในการทำความเข้าใจแผนภาพซีเควนซ์								
ส่วนประกอบ	วัตถุประสงค์	หมายเลขคำถามต้นแบบ	คำถามต้นแบบ	ตัวอย่างคำถามจากระบบการขายซีดีออนไลน์	คำตอบ	ความสอดคล้องของรูปแบบคำถามกับวัตถุประสงค์		
						สอดคล้อง	ไม่แน่ใจ	ไม่สอดคล้อง
Conditions	เพื่อวัดความเข้าใจเกี่ยวกับเงื่อนไข	UCo1	จงอธิบายความหมายของเงื่อนไข C	จากแผนภาพซีเควนซ์ *การค้นหาและสั่งซื้อซีดี* จงอธิบายความหมายของเงื่อนไข [if found]	เงื่อนไข [if found] หมายถึงในกรณีที่ค้นหาซีดีแล้วค้นพบซีดีที่ต้องการค้นหา			
		UCo2	ถ้าเงื่อนไข C เป็นจริง จะมีการส่งเมสเสจใด	จากแผนภาพซีเควนซ์ *การค้นหาและสั่งซื้อซีดี* ถ้าเงื่อนไข [if found] เป็นจริง จะมีการส่ง message ใด	create_cd_list()			
Scenarios	เพื่อวัดความเข้าใจโดยรวมเกี่ยวกับแผนภาพซีเควนซ์	US1	จงอธิบายแผนภาพซีเควนซ์นี้แบบคร่าว ๆ	จงอธิบายแผนภาพซีเควนซ์ *การค้นหาและสั่งซื้อซีดี* แบบคร่าว ๆ	แผนภาพซีเควนซ์นี้เป็นการค้นหาซีดีของลูกค้า ซึ่งสามารถทำการค้นหาได้หลาย ๆ ครั้ง ตามเงื่อนไขที่ต้องการ จากนั้นจะทำการสั่งซื้อซีดีโดยการเลือกซีดีลงในรถเข็นชอปปิ้ง แล้วจึงสร้างรายการสั่งซื้อซีดี			
		US2	มีคลาสใดบ้างที่เกี่ยวข้องกับแผนภาพซีเควนซ์นี้	มีคลาสใดบ้างที่เกี่ยวข้องกับแผนภาพซีเควนซ์ *การค้นหาและสั่งซื้อซีดี*	Customer, CD, MKTInfo,Review, Artist Info และ SampleClip			
		US3	จากแผนภาพซีเควนซ์นี้ คลาสใดบ้างที่จะถูกทำลาย หลังสิ้นสุดการทำงาน	จากแผนภาพซีเควนซ์ *การค้นหาและสั่งซื้อซีดี* คลาสใดบ้างที่จะถูกทำลาย หลังสิ้นสุดการทำงาน	CD List และ ShoppingCart			

Table B.6: Validation form of template questions for assessing modifiability of class diagram.

คำถามวัดความสามารถในการปรับเปลี่ยนแผนภาพคลาส								
ส่วนประกอบ	วัตถุประสงค์	หมายเลขคำถามต้นแบบ	รูปแบบคำถาม	ตัวอย่างคำถาม จากระบบการขายซีดีออนไลน์	คำตอบ	ความสอดคล้องของรูปแบบคำถามกับวัตถุประสงค์		
						สอดคล้อง	ไม่แน่ใจ	ไม่สอดคล้อง
Classes	เพื่อวัดความสามารถในการปรับเปลี่ยนคลาส	MC1	ให้ปรับเปลี่ยนคลาส โดยการเพิ่ม ลบ หรือ แก้ไข	หากต้องการจัดเก็บรายละเอียดของการรับสินค้า (ซีดี) จากผู้ขาย (Supplier) โดยข้อมูลที่นำมาจัดเก็บได้มาจากใบส่งสินค้า และนำข้อมูลของการจัดส่งดังกล่าวมาปรับปรุงจำนวนของซีดีที่มีอยู่ในร้าน จะต้องแก้ไขแผนภาพคลาสอย่างไร จงวาดภาพคลาสเฉพาะในส่วนที่เกี่ยวข้อง	- เพิ่มคลาส Receipt และคลาส CDItem - ลากเส้นความสัมพันธ์แบบ Association ระหว่างคลาส Receipt กับ Supplier - ลากเส้นความสัมพันธ์แบบ Association ระหว่างคลาส Receipt กับ CDItem - ลากเส้นความสัมพันธ์แบบ Association ระหว่างคลาส CDItem กับ CD			
Attributes	เพื่อวัดความสามารถในการปรับเปลี่ยนแอทริบิวต์	MA1	ให้ปรับเปลี่ยนแอทริบิวต์ โดยการเพิ่ม ลบ หรือ แก้ไข	หากต้องการให้เปอร์เซ็นต์ส่วนลดสำหรับลูกค้าที่เป็นลูกค้าประจำ โดยลูกค้าแต่ละคนมีเปอร์เซ็นต์ส่วนลดที่แตกต่างกัน จะต้องแก้ไขแผนภาพคลาสอย่างไร	เพิ่มแอทริบิวต์ discountPercent: Integer ลงในคลาส Customer			
Methods	เพื่อวัดความสามารถในการปรับเปลี่ยนเมทธอด	MM1	ให้ปรับเปลี่ยนเมทธอด โดยการเพิ่ม ลบ หรือ แก้ไข	หากต้องการให้ลูกค้าสามารถทำการแก้ไขรายการสั่งซื้อได้ จะต้องแก้ไขแผนภาพคลาสอย่างไร	เพิ่มเมทธอด update_order ลงในคลาส Order			

Table B.7: Validation form of template questions for assessing modifiability of class diagram (continued).

คำถามวัดความสามารถในการปรับเปลี่ยนแผนภาพคลาส								
ส่วนประกอบ	วัตถุประสงค์	หมายเลขคำถามต้นแบบ	รูปแบบคำถาม	ตัวอย่างคำถามจากระบบการขายซีดีออนไลน์	คำตอบ	ความสอดคล้องของรูปแบบคำถามกับวัตถุประสงค์		
						สอดคล้อง	ไม่แน่ใจ	ไม่สอดคล้อง
Relationships	เพื่อวัดความสามารถในการปรับเปลี่ยนความสัมพันธ์	MR1	ให้ปรับเปลี่ยนความสัมพันธ์ โดยการเพิ่ม ลบ หรือ แก้ไข	หากร้านขาย ซีดี แห่งนี้ ต้องการจะเพิ่มการขายเทปเพลงด้วย โดยลูกค้าสามารถจะค้นหาข้อมูลของเทปเพลงและกลวิธีทางการตลาดของเทปเพลงเหมือนกับของซีดี ทุกประการ จะต้องแก้ไขแผนภาพคลาสอย่างไร	- เพิ่มคลาส Product และคลาส Tape - สร้างความสัมพันธ์แบบ Generalization โดยให้คลาส CD และคลาส Tape เป็นซับคลาสของคลาส Product - ย้ายเส้นความสัมพันธ์ทุกเส้นที่เคยเชื่อมกับคลาส CD มาเชื่อมกับคลาส Product แทน			
		MR2	ให้แก้ไข multiplicity ของความสัมพันธ์	หากต้องการให้คนที่มาสมัครสมาชิกเพื่อเป็นลูกค้าของทางร้าน ต้องสั่งซื้อซีดี และสั่งซื้ออย่างน้อยคราวละ 10 แผ่น จะต้องแก้ไขแผนภาพคลาสอย่างไร	- แก้ไข multiplicity ของความสัมพันธ์ระหว่างคลาส Customer กับ Order เป็น 1 และ 1..n - แก้ไข multiplicity ของความสัมพันธ์ระหว่างคลาส Order กับ Order Item เป็น 1 และ 10..n			

Table B.8: Validation form of template questions for assessing modifiability of sequence diagram.

คำถามวัดความสามารถในการปรับเปลี่ยนแผนภาพคลาส								
ส่วนประกอบ	วัตถุประสงค์	หมายเลขคำถามต้นแบบ	รูปแบบคำถาม	ตัวอย่างคำถาม จากระบบการขายซีดีออนไลน์	คำตอบ	ความสอดคล้องของรูปแบบคำถามกับวัตถุประสงค์		
						สอดคล้อง	ไม่แน่ใจ	ไม่สอดคล้อง
Messages	เพื่อวัดความสามารถในการปรับเปลี่ยนเมสเสจ	MMs1	ให้ปรับเปลี่ยนการส่งเมสเสจ โดยการเพิ่ม ลบ หรือแก้ไข	จากแผนภาพซีเควนซ์ *การค้นหาและสั่งซื้อซีดี* ภายหลังการสร้างรายการสั่งซื้อ ถ้าต้องการให้ลูกค้าทราบรายละเอียดของการสั่งซื้อทั้งหมด จะแก้ไขแผนภาพซีเควนซ์อย่างไร จงอธิบาย	หลังเมสเสจ create_order() เพิ่มการส่ง เมสเสจ display() จากคลาส Customer มายังคลาส Order			
		MMs2	ให้ปรับเปลี่ยนการคืนค่าแอทริบิวต์ โดยการเพิ่ม ลบ หรือแก้ไข	จากคำถามข้อ MA1 หากต้องการให้ลูกค้าสามารถทราบเปอร์เซ็นต์ส่วนลดของตัวเองได้ จะมีการสร้างแผนภาพซีเควนซ์อย่างไร	- สร้างเมสเสจ getDiscountPercent - คืนค่า discountPercent จากคลาส Customer			
Conditions	เพื่อวัดความสามารถในการปรับเปลี่ยนเงื่อนไข	MCo1	ให้ปรับเปลี่ยนเงื่อนไข โดยการเพิ่ม ลบ หรือแก้ไข	จากแผนภาพซีเควนซ์ *การค้นหาและสั่งซื้อซีดี* ลูกค้าที่จะทำการสั่งซื้อซีดี ได้จะต้องผ่านการตรวจสอบเครดิตการ์ดก่อนว่าใช้ได้จริงหรือไม่ จะแก้ไขแผนภาพซีเควนซ์อย่างไร	- หลังเมสเสจ create_order เพิ่มการส่งเมสเสจ checkCredit() - ต่อจากนั้นส่งเมสเสจ confirm_order () โดยบนเมสเสจนี้ ให้เพิ่มเงื่อนไข [if checkCredit is ok]			
Scenarios	เพื่อวัดความสามารถในการปรับเปลี่ยน scenario	MS1	ให้สร้างแผนภาพซีเควนซ์ตาม scenario ที่เกิดขึ้นใหม่	จากคำถามข้อ MC1 จงสร้างแผนภาพซีเควนซ์เพื่ออธิบาย scenario ดังกล่าว	(คำตอบขึ้นอยู่กับการออกแบบ)			

Table B.9: The IOC values of all template questions.

Template question No.	The IOC value	Validation result	Template question No.	The IOC value	Validation result
UC1	1	accepted	UMs4	1	accepted
UA1	1	accepted	UCo1	1	accepted
UA2	0.67	accepted	UCo2	1	accepted
UA3	1	accepted	US1	1	accepted
UM1	1	accepted	US2	1	accepted
UM2	1	accepted	US3	1	accepted
UM3	0.67	accepted	MC1	1	accepted
UM4	1	accepted	MA1	1	accepted
UR1	1	accepted	MM1	1	accepted
UR2	1	accepted	MR1	1	accepted
UR3	1	accepted	MR2	1	accepted
UR4	1	accepted	MMs1	1	accepted
UMs1	1	accepted	MMs2	1	accepted
UMs2	1	accepted	MCo1	1	accepted
UMs3	1	accepted	MS1	1	accepted

B.3 An Example of the Examinations

The examination of each software design model contained 20 questions for assessing understandability of the class and sequence diagrams. All of questions related to understanding of software structure and behavior described by the elements in class and sequence diagrams including attributes, methods, classes, relationships, messages and conditions. There are 10 questions for assessing modifiability of the class and sequence diagrams. Subjects were asked to modify design diagrams with tasks covering on changing and adding software functionality. The following is the examination of a sample software named 'Online CD shop system'.

Table B.10: General description of Online CD shop system.

ข้อมูลระบบร้านขายซีดีออนไลน์
ฟังก์ชันการทำงานของระบบ
ระบบนี้เป็นระบบจัดการการขายซีดีเพลงทางอินเทอร์เนต โดยลูกค้าสามารถสั่งซื้อสินค้าผ่านทางเว็บไซต์ของระบบ โดยมีฟังก์ชันรองรับการทำงานดังนี้ ฟังก์ชันการค้นหาข้อมูลสินค้า ,ฟังก์ชันการสั่งซื้อสินค้า และฟังก์ชันการโฆษณาส่งเสริมสินค้า ลูกค้าทุกคนจะต้องทำการกรอกข้อมูลส่วนตัว และสามารถสั่งซื้อสินค้าโดยใช้บัตรเครดิต

แผนภาพคลาส

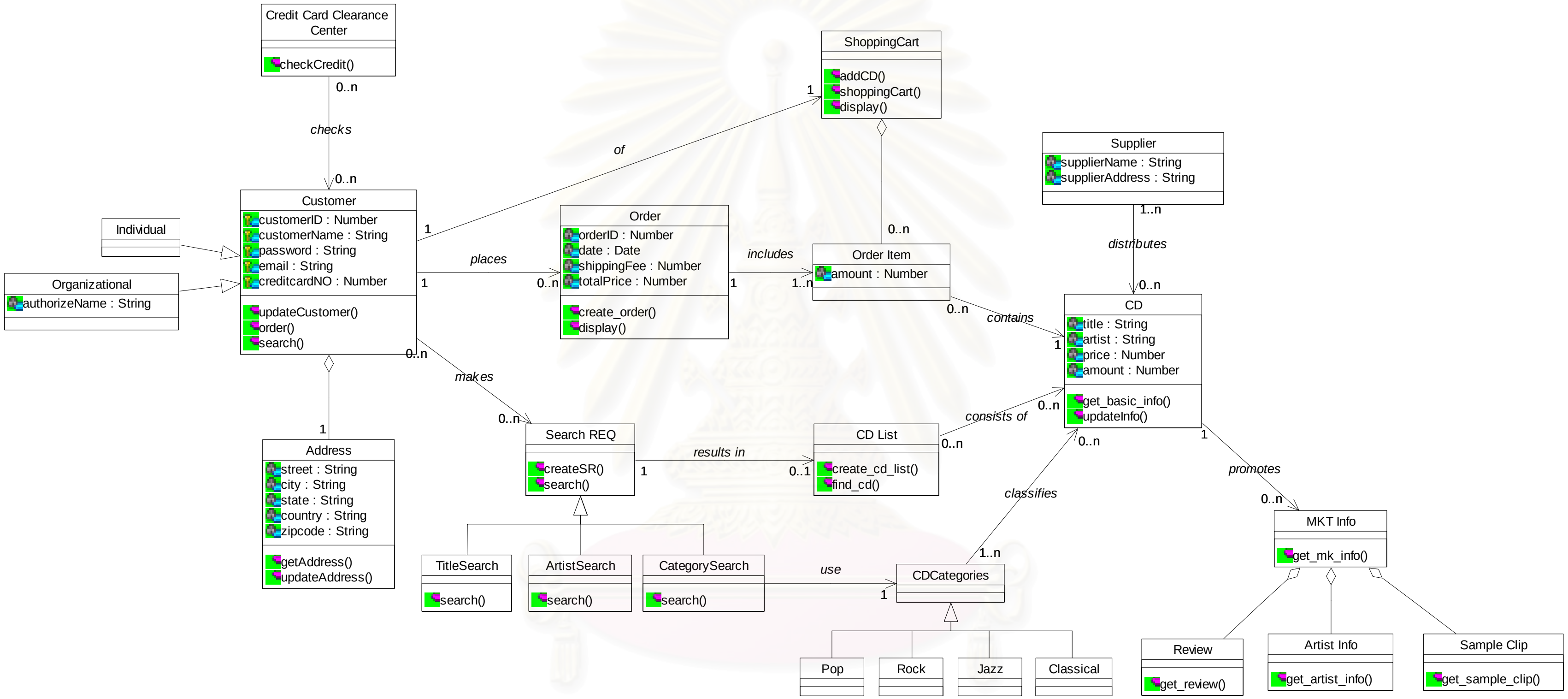


Figure B.1: Class diagram of Online CD shop system.

แผนภาพซีเควนซ์

แผนภาพซีเควนซ์การค้นหาและสั่งซื้อซีดี

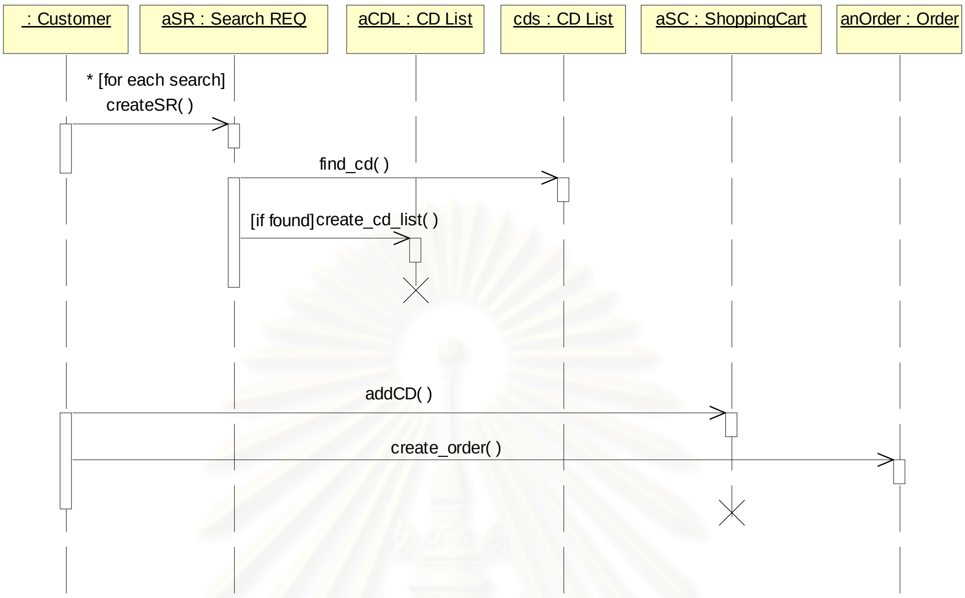


Figure B.2: Sequence diagram of searching and ordering CD.

แผนภาพซีเควนซ์การรับข้อมูลการตลาด

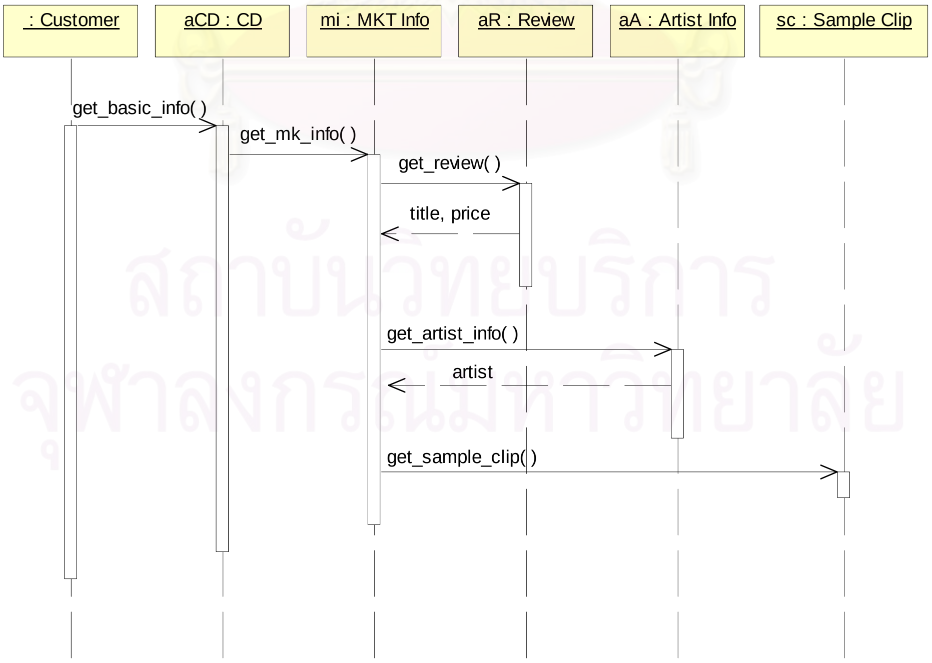


Figure B.3: Sequence diagram of marketing information.

ชื่อ ....นามสกุล...

เวลาเริ่มทำ ....เวลาสิ้นสุดการทำ ....

ชุดที่ 1 ข้อละ 1 คะแนน

ชื่อระบบ: ร้านขายซีดีออนไลน์

คำถามที่เกี่ยวข้องกับแผนภาพคลาส

จงเติมคำตอบลงในช่องว่าง

1. คลาส Order สามารถให้บริการอะไรได้บ้าง

   ……………………………………………………………………………………………………...

2. แอทริบิวต์ artist ของคลาส CD สามารถถูกเรียกใช้หรือแก้ไขจากคลาสอื่นโดยตรงได้หรือไม่

   ……………………………………………………………………………………………………...

3. แอทริบิวต์ใดของคลาสใดบ้าง ที่มีผลต่อค่าของแอทริบิวต์ totalPrice ของคลาส Order

   ……………………………………………………………………………………………………...

4. คลาส Organization มีแอทริบิวต์อะไรบ้าง

   ……………………………………………………………………………………………………...

5. เมทธอด Search() ของคลาส ArtistSearch ทำงานอะไร

   ……………………………………………………………………………………………………...

6. เมทธอด search() ของคลาส SearchREQ สามารถถูกเรียกใช้จากคลาสอื่นได้หรือไม่

   ……………………………………………………………………………………………………...

7. เมทธอดใดของคลาสใดควรจะเรียกใช้เมทธอด find_CD() ของคลาส CDList

   ……………………………………………………………………………………………………...

8. คลาส Individual มีเมทธอดใดบ้าง

   ……………………………………………………………………………………………………...

9. จงอธิบายความสัมพันธ์ระหว่างคลาส CDCategories และ Jazz

   ……………………………………………………………………………………………………...

10. จงอธิบายความสัมพันธ์ระหว่างคลาส Supplier และ CD

    ……………………………………………………………………………………………………...

11. จงอธิบายความสัมพันธ์ระหว่างคลาส ShoppingCart และ Order Item

    ……………………………………………………………………………………………………...

12. จากความสัมพันธ์ Places ระหว่างคลาส Customer และคลาส Order ลูกค้าจะต้องสั่งซื้อซีดี อย่างน้อยกี่แผ่น

    ……………………………………………………………………………………………………...

คำถามที่เกี่ยวข้องกับแผนภาพซีเควนซ์

จงเติมคำตอบลงในช่องว่าง

13. จากแผนภาพซีเควนซ์ *การค้นหาและสั่งซื้อซีดี* ในการสร้างรายการสั่งซื้อ มีเมสเสจใดบ้างที่เกี่ยวข้อง

……………………………………………………………………………………………………...

14. จากแผนภาพซีเควนซ์ *การค้นหาและสั่งซื้อซีดี* เมสเสจใดที่สามารถมีการส่งซ้ำได้หลาย ๆ ครั้ง และมีความหมายว่าอย่างไร

……………………………………………………………………………………………………...

15. จากแผนภาพซีเควนซ์ *การรับข้อมูลการตลาด* การส่ง message get_reveiw(), get_artist() และ get_sample_clip สามารถสลับลำดับการเรียกใช้ได้หรือไม่

……………………………………………………………………………………………………...

16. จากแผนภาพซีเควนซ์ *การรับข้อมูลการตลาด* ผลของการเรียกใช้เมสเสจ get_basic_info จะได้รับข้อมูล (แอทริบิวต์) ใดกลับมาบ้าง

……………………………………………………………………………………………………...

17. จากแผนภาพซีเควนซ์ *การค้นหาและสั่งซื้อซีดี* จงอธิบายความหมายของเงื่อนไข [if found]

……………………………………………………………………………………………………...

18. จากแผนภาพซีเควนซ์ *การค้นหาและสั่งซื้อซีดี* ถ้าเงื่อนไข [if found] เป็นจริงจะมีการส่ง message ใด

……………………………………………………………………………………………………...

19. จงอธิบายแผนภาพซีเควนซ์ *การค้นหาและสั่งซื้อซีดี* แบบคร่าว ๆ

……………………………………………………………………………………………………...

20. จากแผนภาพซีเควนซ์ *การค้นหาและสั่งซื้อซีดี* คลาสใดบ้างที่จะถูกทำลาย หลังสิ้นสุดการทำงาน

……………………………………………………………………………………………………...

ชื่อ ....นามสกุล...

เวลาเริ่มทำ ....เวลาสิ้นสุดการทำ ....

ชุดที่ 2

ชื่อระบบ: ร้านขายซีดีออนไลน์

จงเขียนอธิบาย หรือวาดแผนภาพในส่วนที่เกี่ยวข้อง

คำถามที่เกี่ยวข้องกับแผนภาพคลาส

1. หากต้องการจัดเก็บรายละเอียดของการรับสินค้า (ซีดี) จากผู้ขาย (Supplier) โดยข้อมูลที่นำมาจัดเก็บได้มาจากใบส่งสินค้า และนำข้อมูลของการจัดส่งดังกล่าวมาปรับปรุงจำนวนของซีดีที่มีอยู่ในร้าน จะต้องแก้ไขแผนภาพคลาสอย่างไร จงวาดภาพคลาสเฉพาะในส่วนที่เกี่ยวข้อง (3 คะแนน)
2. หากต้องการให้เปอร์เซ็นต์ส่วนลดสำหรับลูกค้าที่เป็นลูกค้าประจำ โดยลูกค้าแต่ละคนมีเปอร์เซ็นต์ส่วนลดที่แตกต่างกัน จะต้องแก้ไขแผนภาพคลาสอย่างไร (1 คะแนน)
3. หากต้องการให้ลูกค้าสามารถทำการแก้ไขรายการสั่งซื้อได้ จะต้องแก้ไขแผนภาพคลาสอย่างไร (1 คะแนน)
4. หากร้านขายซีดีแห่งนี้ ต้องการจะเพิ่มการขายเทปเพลงด้วย โดยลูกค้าสามารถจะค้นหาข้อมูลของเทปเพลง และกลวิธีทางการตลาดของเทปเพลง เหมือนกับของซีดีทุกประการ จะต้องแก้ไขแผนภาพคลาสอย่างไร (3 คะแนน)
5. หากต้องการให้ระบบสามารถให้ข้อมูลทางการตลาดเพิ่มเติม โดยให้ลูกค้าสามารถเข้ามาตั้งคำถามเพื่อขอข้อมูลตามที่ต้องการได้ จะต้องแก้ไขแผนภาพคลาสอย่างไร (2 คะแนน)
6. หากต้องการให้คนที่มาสมัครสมาชิกเพื่อเป็นลูกค้าของทางร้าน ต้องสั่งซื้อซีดี และสั่งซื้ออย่างน้อยคราวละ 10 แผ่น จะต้องแก้ไขแผนภาพคลาสอย่างไร (2 คะแนน)

คำถามที่เกี่ยวข้องกับแผนภาพซีเควนซ์

หมายเหตุ : กรณีที่ไม่มีแอทริบิวต์หรือเมทธอดในแผนภาพคลาส สามารถเพิ่มเติมได้เอง โดยให้เขียนอธิบายการเพิ่มเติมดังกล่าวด้วย

7. จากแผนภาพซีเควนซ์ *การค้นหาและสั่งซื้อซีดี* ภายหลังการสร้างรายการสั่งซื้อ ถ้าต้องการให้ลูกค้าทราบรายละเอียดของการสั่งซื้อทั้งหมด จะแก้ไขแผนภาพซีเควนซ์อย่างไร (1 คะแนน)
8. จากคำถามข้อ 2) หากต้องการให้ลูกค้าสามารถทราบเปอร์เซ็นต์ส่วนลดของตัวเองได้ จะมีการสร้างแผนภาพซีเควนซ์อย่างไร (2 คะแนน)
9. จากแผนภาพซีเควนซ์ *การค้นหาและสั่งซื้อซีดี* ลูกค้าที่จะทำการสั่งซื้อซีดี ได้ จะต้องผ่านการตรวจสอบเครดิตการ์ดก่อนว่าใช้ได้จริงหรือไม่ จะแก้ไขแผนภาพซีเควนซ์อย่างไร (2 คะแนน)
10. จากคำถามข้อ 1) จงวาดแผนภาพซีเควนซ์เพื่ออธิบาย scenario ดังกล่าว (3 คะแนน)

APPENDIX C

QUESTIONNAIRE

Personal Details and Experience

1. What is your age?
2. How long have you worked in field of software engineering?

Please answer the following 4 questions based on this experience scale:

None	Little	Average	Substantial	Professional
1	2	3	4	5

3. What is your experience with software engineering practice?
4. What is your experience with design documents in general?
5. What is your experience with modeling with UML for objected-oriented design?
6. What is your experience with software maintenance?

Motivation and Performance

Please answer the following 3 questions based on this scale:

Not	Poorly	Fairly	Well	Highly
1	2	3	4	5

7. Estimate how motivated you were to perform well in this experiment.
8. Estimate how well you understood what was required of you.
9. Estimate your overall understanding of the design documents.

10. Can you complete all the tasks within time out period?
11. If you could not complete all the tasks, please indicate why.
12. In your opinion, what caused you the most difficulty to understand the design documents?

13. In your opinion, what caused you the most difficulty to modify the design documents?
14. Estimate the accuracy (in percent) of your answer to the examinations.

Miscellaneous

15. Have you learned anything from participating in this experiment? Please specify.
16. Any additional comments?

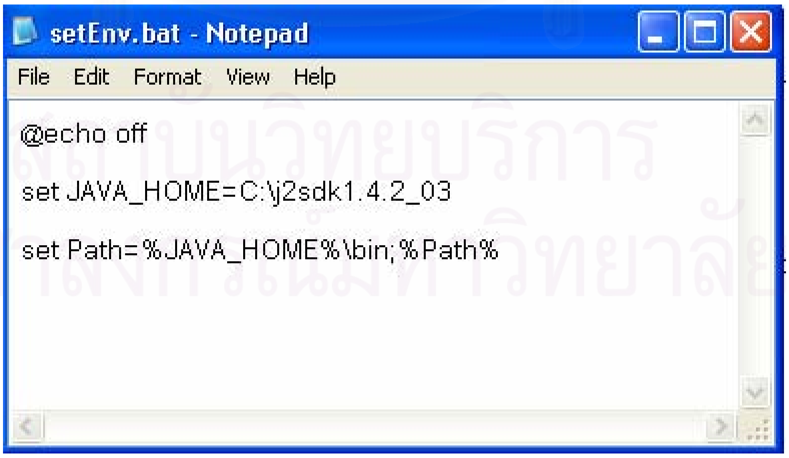
APPENDIX D

USER MANUAL OF A TOOL FOR MEASURING STRUCTURAL COMPLEXITY METRICS AND PREDICTING MAINTAINABILITY

D.1 Tool Installation

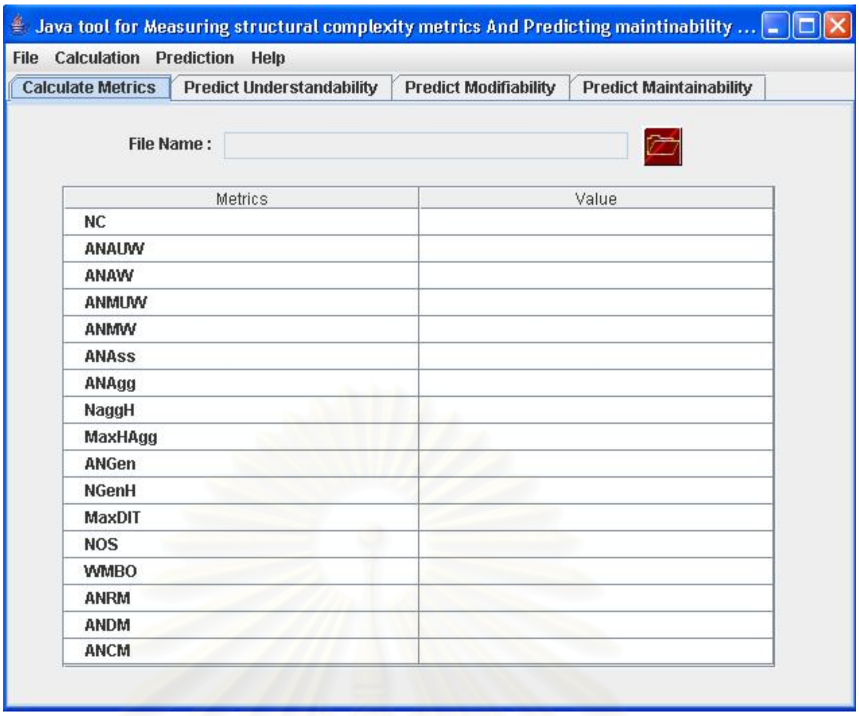
Steps for installing a Java tool for Measuring structural complexity metrics And Predicting maintainability (JMAP) are listed as follows.

1) Install Java™ 2 Standard Edition Runtime Environment (JRE). This program can be downloaded from http://java.sun.com.
2) Copy folder MLP to C:\
3) Copy folder JMAP to C:\
4) Open C:\JMAP\setEnv.bat with Notepad or other text editors. The window as shown in Figure D.1 will be appeared. Edit value of JAVA_HOME in line 2 to identify the directory in which JRE is installed. In this case, value of JAVA_HOME is C:\j2sdk1.4.2_03.
5) Double click C:\JMAP\run.bat for running the tool. The window as shown in Figure D.2 will be displayed.



```
@echo off
set JAVA_HOME=C:\j2sdk1.4.2_03
set Path=%JAVA_HOME%\bin;%Path%
```

Figure D.1: Content of JMAP.bat.


Java tool for Measuring structural complexity metrics And Predicting maintainability ...
File
Calculation
Prediction
Help
Calculate Metrics
Predict Understandability
Predict Modifiability
Predict Maintainability

File Name :

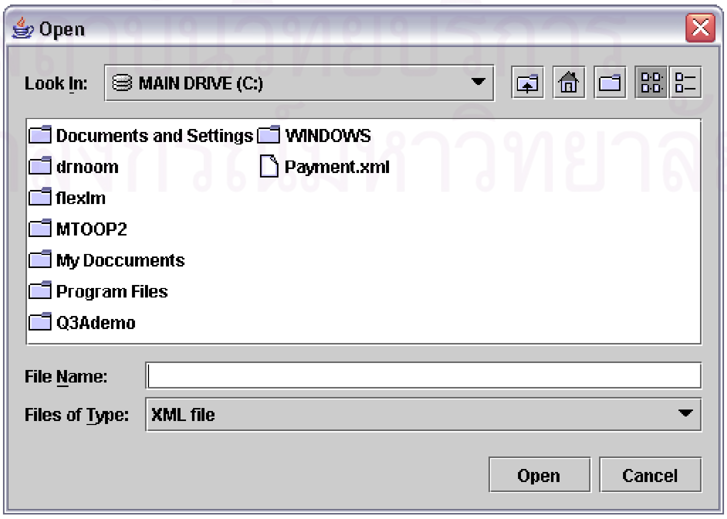
Metrics	Value
NC	
ANAUW	
ANAW	
ANMUW	
ANMW	
ANAss	
ANAgg	
NaggH	
MaxHAgg	
ANGen	
NGenH	
MaxDIT	
NOS	
WMBO	
ANRM	
ANDM	
ANCM	

Figure D.2: A tool for measuring structural complexity metrics and predicting maintainability.

D.2 User Manual for Utilizing a Tool for Measuring Structural Complexity Metrics and Predicting Maintainability

D.2.1 Open Input File

Input file of this tool is in a format of XML file which contains the detail of class and sequence diagrams of a software. From Figure D.2, choose File> Open or click 📂 to open an input file. The window as shown in Figure D.3 will be displayed. Select an input file and click Open.



Look In: MAIN DRIVE (C:)

Documents and Settings
drnoom
flexlm
MTOOP2
My Doccuments
Program Files
Q3Ademo
WINDOWS
Payment.xml

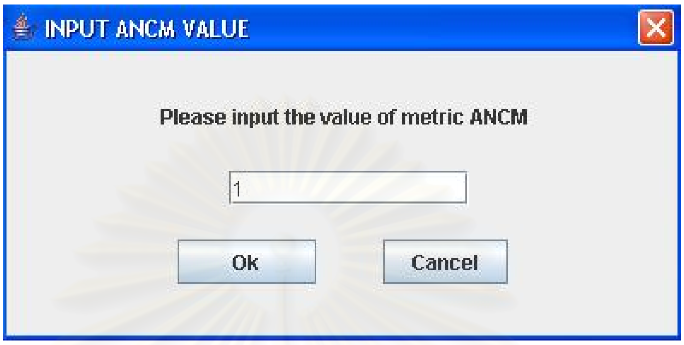
File Name:

Files of Type: XML file

Figure D.3: Window for open input file.

D.2.2 Measuring Structural Complexity Metrics

Choose Calculation > Calculate Metrics. The window as shown in Figure D.4 will be displayed. Input value of ANCM and click Ok. The values of structural complexity metrics will be computed and shown as Figure D.5.

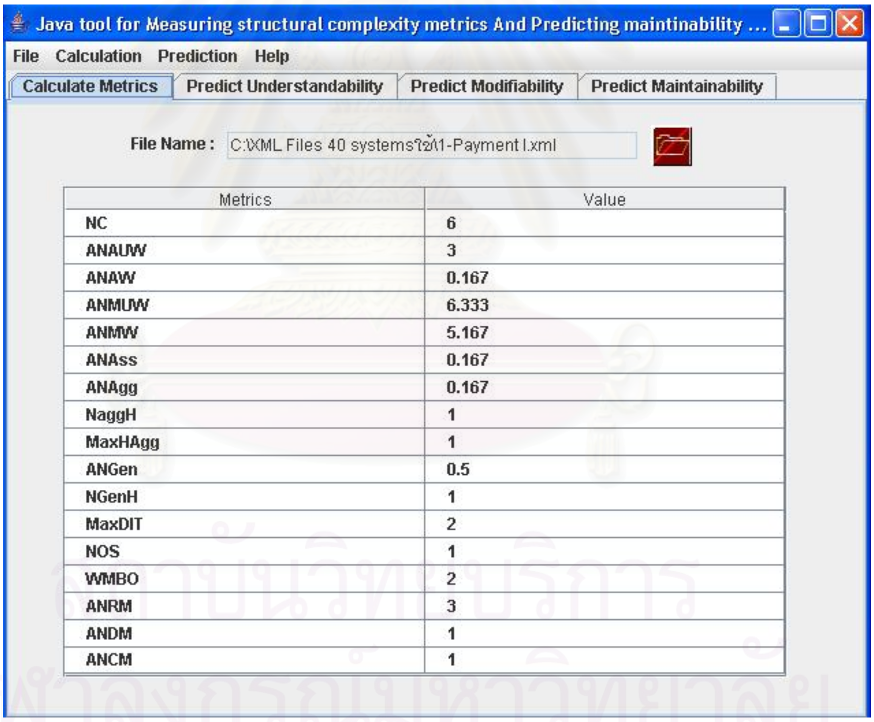


INPUT ANCM VALUE

Please input the value of metric ANCM

1

Figure D.4: Window for input value of ANCM.



Java tool for Measuring structural complexity metrics And Predicting maintinability ...

File Name : C:\XML Files 40 systemsใช้\1-Payment I.xml

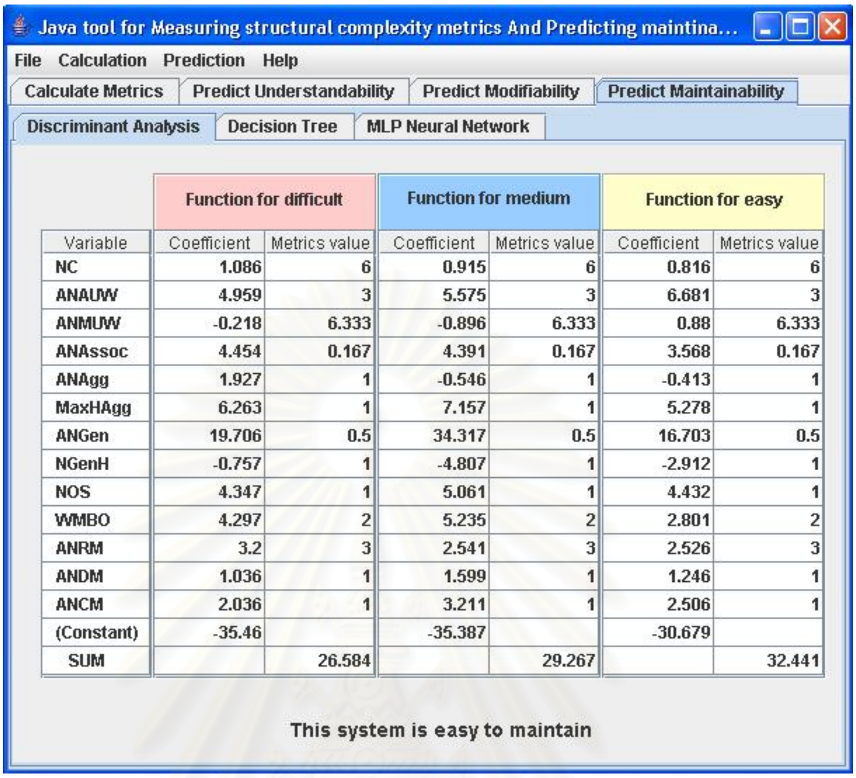
Metrics	Value
NC	6
ANAUW	3
ANAW	0.167
ANMUW	6.333
ANMW	5.167
ANAss	0.167
ANAgg	0.167
NaggH	1
MaxHAgg	1
ANGen	0.5
NGenH	1
MaxDIT	2
NOS	1
WMBO	2
ANRM	3
ANDM	1
ANCM	1

Figure D.5: Window for display the values of structural complexity metrics.

D.2.3 Predicting Understandability, Modifiability and Maintainability

Choose Prediction > Predict Understandability or Prediction > Predict Modifiability or Prediction > Predict Maintainability according to a quality that you want to predict. Select a prediction technique by choosing Discriminant Analysis or Decision

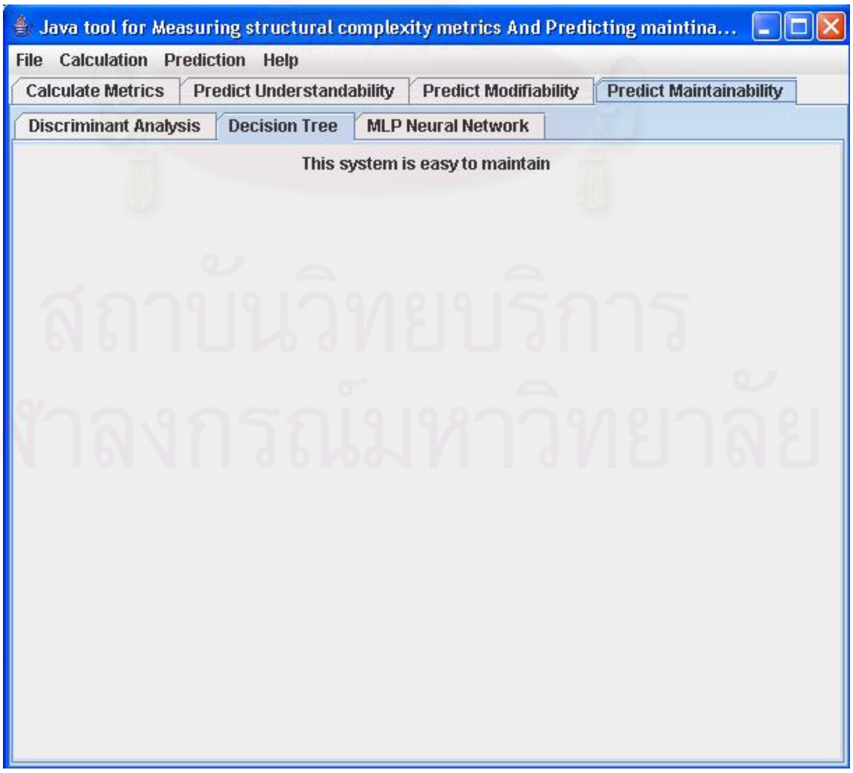
Tree or MLP Neural Network. The result of prediction will be presented as Figure D.6 – Figure D.8.


Java tool for Measuring structural complexity metrics And Predicting maintina...
File Calculation Prediction Help
Calculate Metrics
Predict Understandability
Predict Modifiability
Predict Maintainability
Discriminant Analysis
Decision Tree
MLP Neural Network

	Function for difficult		Function for medium		Function for easy	
Variable	Coefficient	Metrics value	Coefficient	Metrics value	Coefficient	Metrics value
NC	1.086	6	0.915	6	0.816	6
ANAUW	4.959	3	5.575	3	6.681	3
ANMUW	-0.218	6.333	-0.896	6.333	0.88	6.333
ANAssoc	4.454	0.167	4.391	0.167	3.568	0.167
ANAgg	1.927	1	-0.546	1	-0.413	1
MaxHAgg	6.263	1	7.157	1	5.278	1
ANGen	19.706	0.5	34.317	0.5	16.703	0.5
NGenH	-0.757	1	-4.807	1	-2.912	1
NOS	4.347	1	5.061	1	4.432	1
WMBO	4.297	2	5.235	2	2.801	2
ANRM	3.2	3	2.541	3	2.526	3
ANDM	1.036	1	1.599	1	1.246	1
ANCM	2.036	1	3.211	1	2.506	1
(Constant)	-35.46		-35.387		-30.679	
SUM		26.584		29.267		32.441

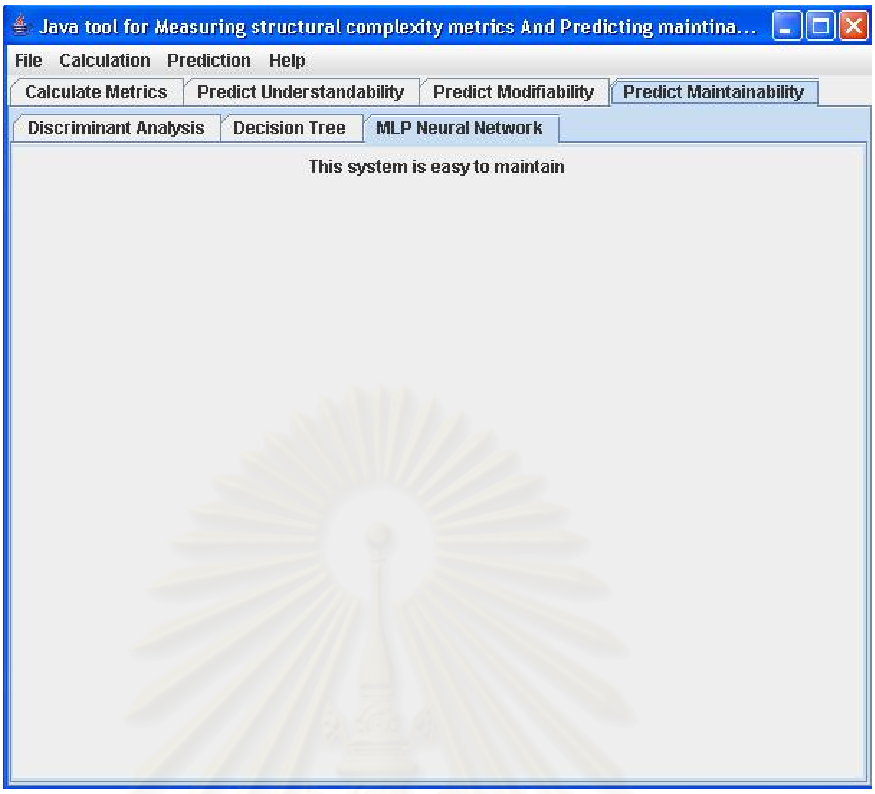
This system is easy to maintain

Figure D.6: Window for display the result of prediction by technique of Discriminant Analysis.


Java tool for Measuring structural complexity metrics And Predicting maintina...
File Calculation Prediction Help
Calculate Metrics
Predict Understandability
Predict Modifiability
Predict Maintainability
Discriminant Analysis
Decision Tree
MLP Neural Network

This system is easy to maintain

Figure D.7: Window for display the result of prediction by technique of Decision Tree.



This system is easy to maintain

Figure D.8: Window for display the result of prediction by technique of Neural Network.

D.2.4 Exit Program

Choose File > Exit

BIOGRAPHY

Name Matinee Kiewkanya

Sex Female

Date of Birth November 8,1974

Place of Birth Chiang Mai

Education:

2006 Ph.D. in Computer Engineering, Chulalongkorn University
Funding source : The Royal Thai Government Scholarship

2001 M.Sc. in Computer Science, Prince of Songkla University
Funding source : The Royal Thai Government Scholarship

1996 B.Sc. in Computer Science, Chiang Mai University
Funding source : The Royal Thai Government Scholarship

Experience:

Jun 2004 – Sep 2004 National ICT Australia, Sydney, Australia
Function: Research Student

Apr 2002 – Jul 2003 Department of Computer Engineering, Chulalongkorn University, Thailand
Project: Object-Oriented Software Metrics (A Collaborative Project between Stock Exchange of Thailand and Department of Computer Engineering)
Function: Research Assistant

Apr 1998 – Mar 1999 Department of Computer Science
Faculty of Science, Prince of Songkhla University, Thailand
Function: Teacher Assistant

Jun 1996 – Present Department of Computer Science,
Faculty of Science, Chiang Mai University, Thailand
Function: Lecturer