FORMAL METRICS FOR QUANTITATIVE ASSESSMENT OF THE QUALITY OF EXPERT SYSTEMS

ZHISONG CHEN

A THESIS

IN

THE DEPARTMENT

OF

COMPUTER SCIENCE

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY AT
CONCORDIA UNIVERSITY
MONTREAL, QUEBEC, CANADA

SEPTEMBER 1994

ISBN 0-315-97686-1

Abstract

Formal Metrics for Quantitative Assessment of the Quality of Expert Systems

Zhisong Chen, Ph.D.

Concordia University, 1994

Various systems have to be assessed and evaluated from time to time to assure their quality, especially in the engineering and scientific disciplines. As a result, many metrics have been defined and used. In software engineering, it has been shown that the use of poor quality software can become very costly in the long run. To overcome this, a large number of metrics have been proposed to quantify the different aspects of conventional software and much progress has been made.

Expert systems are a special type of software, whose main components are the *knowledge base* and *inference engine.* They are applied to solve complex problems that need human expertise, and their applications have dramatically increased in recent years in many different disciplines. However, due to their imprecise and iterative nature, expert systems, especially their knowledge bases are subject to more quality problems than conventional software. Because of this, expert system metrics are urgently needed to assess and predict the quality of expert systems. Unfortunately, so far little work has been done in this area, hence techniques in assessing the quality of expert systems fall far behind those in other disciplines. In view of this, an investigation into the metric measurements for expert systems was proposed as research for this doctoral thesis, and it is hoped this work will stimulate more research and attract more attentions to this area.

Several issues related to expert system metrics are addressed and discussed in this thesis, such as the appropriate formulation and definition of expert system metrics, and the validity assessment of the metrics. In order to formally describe and measure the characteristics of expert systems, an AND/OR digraph is presented. Based on this digraph and the contents of expert systems, new metrics for measuring ES complexity have been proposed, which are RC (Rule Base Complexity) and ERC (Entropy-Based Rule Base Complexity). Five other metrics are formally presented for the measures of the *size* and *search space*, which are NR (Number of Rules), $ADSS$ (Average Depth of Search Space), $ABSS$ (Average Breadth of Search Space), BC (Buchanan's Complexity) and NAC (Number of Antecedents and Consequents). For validating the expert system metrics, this thesis proposes the metric evaluation techniques from two perspectives: (1) empirical evaluation that is based on the statistical analysis and testing of the measuring results; and (2) theoretical evaluation, that is, evaluating the metrics in an abstract way. Four general criteria and eleven desired properties regarding the expectation of metric performance are proposed for this purpose, against which metrics are further evaluated. The evaluation results reveal that RC metric, designed as a hybrid metric that takes into account the *matching patterns, size* and *search space* of expert systems, is most effective, giving the best performance among all the presented metrics, and it can serve as a useful tool in developing quality expert systems.

Acknowledgements

First, and foremost, I would like to express my sincere gratitude and appreciation to my supervisor, Professor Ching Y. Suen for his invaluable guidance, advice, and support throughout this research. Without his encouragement and understanding, the completion of this thesis would be impossible.

Grateful thanks are extended to Dr. P. Grogono, Dr. R. Shinghal, Dr. S. Lin of Mechanical Engineering Department, and Dr. M. G. Strobel of Psychology Department of University of Montreal, for their assistance, suggestions and comments. I am also thankful to Dr. Alun Preece and students of the expert system classes for providing valuable data for the experiments conducted in this research.

To all the staff, students at Centre for Pattern Recognition and Machine Intelligence (CENPARMI), I wish to say thank you all for your friendship, help and cooperation. I will never forget the beautiful memory working at CENPARMI.

Special appreciation goes to my wife, Ying Wang, for her love and wholehearted support, and to my parents for their patience and encouragement.

This research was funded by Bell Canada. The assistance of Messrs. J. N. Drouin and F. Coallier are gratefully acknowledged.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Measures and Their Role

Precise and well understood metrics can be seen everywhere in our daily life, temperature, size, distance, inflation rate, *etc.* They are essential and form the basis for making decisions. Generally, we need metrics because they can extend and enhance our ability to sense things not accessible to our innate ability by quantitatively capturing and providing the attribute information of the target objects. Metrics are especially useful in scientific and engineering fields, for example, measures in electronic systems allow us to design a circuit to function properly. Measures of the building enable us to allocate space. In order to make so-called software engineering qualify as a real engineering discipline, much effort has been spent in quantitative metric measurements of conventional software and much progress has been made. Software engineering is still in a developing process with new metrics being proposed, examined and evaluated. Some of the best known metrics for conventional software

are (1) volume-based "software science" [Halstead, 1977], (2) control flow-based cyclomatic complexity metric [McCabe, 1976], and (3) Lines of Code.

The major role that metrics play in many applications is to provide a means of quantitatively assessing the objects and predicting the characteristics associated with the objects. *Assessment* implies the situations where the metric values are used to evaluate objects. *Prediction* denotes the application that the known metric values are used to further foretell some important characteristics. The applications of metrics will result in many benefits, for example, lower development cost, higher productivity, improved and reliable quality of the products, satisfaction of results, and so on. Therefore, metrics enable a research area to evolve from an art to an engineering and scientific discipline.

1.2 Software Concerns

In the literature, it has been shown many times that software with poor quality has resulted in high development costs. In order to deal with this problem effectively, quantitative measures on the software development processes and products are needed. Already, for conventional software, many studies and research activities in this area have been carried out and a rapidly growing number of metrics have being proposed and applied to predict and assess the different characteristics of software quality. The followings list some usages of the metric values:

- To characterize the quality of the products and development processes
- To decompose a complex software system into several manageable subparts

- To predict software attribute values
- To reveal potential trouble spots (unusual components) of the software
- To aid in the distribution of resources to different parts of a system
- To restructure the software to obtain better results
- To estimate the development cost
- To evaluate the product

Different metrics are defined and used according their specific needs and purposes, and there exist many types of measures, such as process measures, product measures, static measures, descriptive measures, code measures, design measures, date-flow measures, inter-and-intra modular measures, *etc.*, they all aim to provide information for various quality assurance activities.

1.3 Motivation and Goals of This Research

Expert systems (ES), whose main components are the *knowledge bases* and *inference engines*, are a special type of software applied to solve complex problems that need human expertise. They possess some features in common with conventional software, but also some special features of their own, especially their knowledge bases. For example, the complex *interactions* between different pieces of knowledge in the knowledge base is one such distinctive feature. Recently, the applications of ES have dramatically increased. Some of the application areas include telecommunication, medicine, geology, business, finance, military and others [Suen and Shinghal, 1991, Liebowitz, 1991]. However, the solutions and domains for these problems are usually imprecise and ill-defined because of their complexity. This makes the knowledge bases, one of the main components of ES, which are used to capture the human expertise, subject to more quality problems than other parts of the systems.

Using *rule bases* to implement the knowledge bases is a prevalent technique in knowledge engineering because of the easy construction of rules. But, the loose construction requirements for the rules may also lead to a complicated rule base which contains many potential problems such as *inconsistency* and *redundancy*, resulting in an increase in the risk of applying expert systems. Due to the distinctive characteristics of rule bases and their strong influence on the quality of the whole expert systems, they have been the focus of many studies. In this thesis, we will also place our emphasis on rule bases, especially, methods of measuring them.

Because of the widespread use of ES and the great concern about ES quality, ES metrics are urgently needed to quantify the quality of ES, especially their rule bases. Unfortunately, so far little work in this area has been done, even though the crucial role of measurements has been widely recognized for conventional software in software engineering. Also, the few existing proposed ES metrics were designed for some particular interests only, and they have not been validated by comprehensive experiments and tests. The lack of study on ES metrics therefore motivated this doctoral research to investigate the metrics that are related to the qualitative characteristics of rule bases, especially, the three important quality factors: *maintainability*, *reliability* and *testability*. The objective of our research is to study the application and applicability of metric measures to ES from both theoretical and empirical grounds. We hope this research will lay a foundation for future research in which our long-term goal is expected — building a system of metrics ranging from basic ones to the complex ones to completely assess and predict the quality of products developed in the knowledge engineering environment.

The tasks to be done in this thesis include:

- Studying and analyzing metrics measurements.
- Conducting comparative investigation between conventional software and ES.
- Studying ES quality and measures.
- Investigating and building a formalism for describing ES attributes.
- Presenting formally some important ES quality metrics.
- Proposing new ES complexity metrics.
- Implementing a tool for automatically performing the metric measurements on ES.
- Presenting methods for validity assessment and comparison of ES metrics
- Using metrics for applications.

1.4 Organization of The Dissertation

The organization of this thesis is as follows.

Chapter 1 discusses some background about metric measurements. The motivation and goals of the thesis are also provided, which indicate the focus of our research.

Chapter 2 introduces some fundamental concepts and principles of measurement theory, factors, scope and organization of metric measurements and applications. This chapter provides the rationale for this research.

Chapter 3 investigates the work on conventional software measures. Several main types of conventional software metrics are presented, from which we can gain the experience and practice of existing research on the metric measurements, and utilize them to design ES metrics.

Chapter 4 studies and describes the characteristics of ES. The *dependency*, the *rule base description languages* are discussed, and the *anomaly rate* in rule bases is also presented, which is often used as synonymous with the quality from a narrower point of view. At last, a comparison between ES and conventional software is given, which can be used as a guide in the adaptation of existing conventional software measuring techniques for ES measures.

Chapter 5 examines the construction and measures of ES quality model. Three quality factors that are important to ES quality are highlighted. They are the *Maintainability*, *Reliability* and *Testability*. Quality metrics will form the focus of the following chapters.

Chapter 6 proposes some general guidelines for the measures of ES. They include the general criteria and strategies used. Some current ES metrics are examined, and current research on ES metrics is reviewed. The issue related to a mechanism for formally describing the rule bases is also addressed with the proposition of an AND/OR digraph.

Chapter 7 presents several quality metrics that are related to the two important characteristics of rule bases: *size* and *search space*. Formal descriptions are also given for these metrics.

Chapter 8 investigates the crucial characteristic of rule bases which possess many potential values: the *complexity* that plays an important role in the formulation of ES quality. Two new complexity metrics are proposed, RC and ERC which are based on the presence of several system characteristics and information content, respectively.

Chapter 9 provides two examples to illustrate the above concepts and measures.

Chapter 10 presents our implementation of a tool for automatically measuring the ES. This tool was written in PROLOG and C/C++, and it will produce the measuring results and analyze their results based on statistical analysis. One of the design features of the tool is that it can easily accommodate the addition of new functions (methods) into the system to make new metric measurements.

Chapter 11 shows the data collected to test our proposed metrics, and the measuring results (values) by applying the metrics to the test data. Some of metric applications are discussed, such as the use of metric values to determine the usual ranges of metrics, to identify the unusual ES or profile ES quality. The measuring values can also be used to further evaluate the performance of the metrics.

Chapter 12 describes the statistical techniques that are proposed to assess the validity of the metrics, they include *correlation analysis*, *statistical testing* and *regression analysis*. Several statistical concepts and testing methods are introduced.

Chapter 13 presents the evaluation results of assessing the metrics and comparing their performance from two perspectives: (1) by applying the presented methods which are based on statistics to the metric measuring results; (2) by evaluating the metrics against the proposed criteria and properties that are generally required and desired for the performance of metric measurements. The results lead to the conclusion that our proposed complexity metric — *RC* has the best performance and behavior among all the tested metrics and can indeed be used as a good tool to aid in assessing ES quality. In order to use *RC* to predict the quality of ES or its characteristics, the relationship between the *RC* metric and the *anomaly rate* is further quantified based on our data.

Chapter 14 draws conclusions from our research. It has been found that well-formulated metric, like *RC*, would be useful for developing quality ES. Finally, it is hoped that this research will provide the basis for future work on ES metrics which will meet our long-term goal.

Chapter 2

Fundamentals of Software Metrics

2.1 Introduction

Software metric[1] is usually described as a general term to cover all aspects related to the quantification of software products and processes. As indicated in the previous chapter, the application of quantitative methods or metrics are needed to reduce the cost of software resulting from poor quality, which was wasted at approximately one million pounds per hour of effort in the United Kingdom alone [Möller and Pauliosh, 1993]. Successful application and implementation of software metrics can be found in a large number of software development organizations. Möller listed several examples with such experience [Möller and Pauliosh, 1993] which have proven the effectiveness of the application of software metrics.

It is expected that the use of software metrics, together with other techniques can resolve the so called "software crisis" which challenges the software engineering community.

[1]Here, the term "software" denotes both conventional software and expert systems.

This chapter discusses the fundamental concepts and principles of software measures[2], and relevant issues.

2.2 Applying Measurement Theory

According to IEEE Standard Glossary of Software Engineering [IEEE, 1983], the term "software measure" is described as:

> *A quantitative assessment of the degree to which software possesses a given attribute that affects its quality.*

Also, in the IEEE Standard Dictionary of Measures to Produce Reliable Software [IEEE, 1989], the software measure is defined as:

> *A quantitative assessment of the degree to which software product or process possesses a given attribute.*

From these definitions, we can see that a software measure mainly concerns the assessment of a software or process attribute, which is usually achieved by assigning a number or symbol to the software to characterize that attribute. Measurement denotes such assignment process or activity.

By using the measurement theory, we can formally and concretely describe the above concept. There are three basic parts in the measurement theory: *empirical relation system* which represents the empirical world to be measured; *formal relation*

[2]In some research, metric and measure are used as different terms. In our study, we regard them as synonyms as most other researchers.

system that contains a set of formal objects and ***mapping*** which indicates the relationships between these two systems. Mapping represents the measures. The following sections will discuss and present the application of general measurement theory to the software measures.

2.2.1 Empirical Relation Systems

When applied to software, the empirical relation system represents our empirical knowledge about the software world, which includes the *entities (software or its derivation)*, the *attributes* possessed by the entities, and empirical *properties* which are held among the entities based on the *attributes.* This can be described as:

$$\mathcal{S} = (\mathcal{E}, A, P)$$

where,

$\mathcal{S}$ — the empirical relation system

$\mathcal{E}$ — the set of entities i.e. the set of software or its derivation

A — the set of attributes associated with each piece of software or its derivation

$\mathcal{P}$ — the set of properties held among the entities

The above empirical model which precedes objective measurement is usually established initially by subjective means which reflect the commonly accepted views about the entities and the related attributes and properties.

Example 2.1 Consider the case in which the software complexity is to be measured, then we have:

Entities $\mathcal{E}$: software programs

Attribute A: complexity

Property P: $(x, y) \in \mathcal{P}$ if program x is more complex than program y.

2.2.2 Formal Relation Systems

In order to express the measures on the entities of an empirical relation system based on the attributes of the entities, we need to find a corresponding formal relation system for the empirical relation system, such as a *numerical relation system*, into which the entities, attributes and properties in the empirical relation system are mapped.

Like the empirical relation system, a formal relation system $\mathcal{F}$ can be described as:

$$\mathcal{F} = (\mathcal{N}, T, R)$$

where,

$\mathcal{F}$ — the formal relation system

$\mathcal{N}$ — the set of the formal entities

T — the set of attributes associated with the entities

R — the set of properties held among the entities

In the case of *numerical relation system*, $\mathcal{N}$ is the set of numerical numbers, T is usually the values and $\mathcal{R}$ is the properties among the numbers.

Example 2.2 For the empirical relation system presented in Example 2.1, a corresponding numerical relation system with the following attribute and property can be formed:

Entities: numerical numbers

Attribute T_1: values

Property $\mathcal{R}$: '$(x,\ y) \in \mathcal{R}$' if $x > y$ (Corresponding to program x being more complex than program y).

2.2.3 Mapping and Software Metrics

A software metric or measure Γ is defined as a mapping from $\mathcal{S}$ to $\mathcal{F}$, that is,

$$\Gamma\colon \mathcal{S} \rightarrow \mathcal{F}$$

with the following conditions:

$$\begin{gathered}\forall e \cdot (e \in \mathcal{S}) \wedge (e = (\mathcal{E}_i, A_i, P_i)) \longrightarrow \exists f_i \cdot (f_i \in \mathcal{F}) \wedge (f_i = (N_i, T_i, R_i)) \\ \wedge(\Gamma(\mathcal{E}_i) = N_i) \wedge (\Gamma(A_i) = T_i) \wedge (\Gamma(P_i) = R_i)\end{gathered}$$

The above conditions, called *representation conditions* require that the mapping should reflect the correspondence between the entities, attributes and properties of $\mathcal{S}$ and $\mathcal{F}$, that is, mapping preserves all the components in an empirical system by establishing the corresponding components in the formal relation system, so that the empirical meanings in $\mathcal{S}$ will not be lost. The representation condition entails us to capture the entities, attributes and properties in question and represent them in a formal system where there exist formal descriptions about its components.

In most software measures, the formal relation system $\mathcal{F}$ is the numerical relation system, and the empirical relation system S is the software relation system. The mapping Γ represents the measurements for the software attributes. It maps the software source codes or control flows in $\mathcal{S}$ into numbers, i.e. entities in $\mathcal{F}$. Figure 1 illustrates such a mapping.

Example 2.3 For measuring conventional software, many measures have been proposed, which represent the different mappings based on different purposes and needs. Some of them are: Number of Lines of Codes, Program Volume, Effort and Level [Halstead, 1977], and Cyclomatic Complexity [McCabe, 1976].

The formulation of a mapping function denotes the formulation of a software metric, and the entities in $\mathcal{F}$ usually denote the measuring values. More metrics can be defined based on the mappings. We refine such a structure as the *layer structure.*

2.3 Layer Structure of Metrics

Further analysis of measures indicates that the mapping from $\mathcal{S}$ to $\mathcal{F}$ reflects only one type of metrics, the *direct metrics*, which are obtained directly in the manner discussed above. However, there is another kind of metrics, called indirect metrics, which are defined in terms of existing metrics obtained previously. We can refine this structure of metrics as the *Layer Structure* proposed as follows:

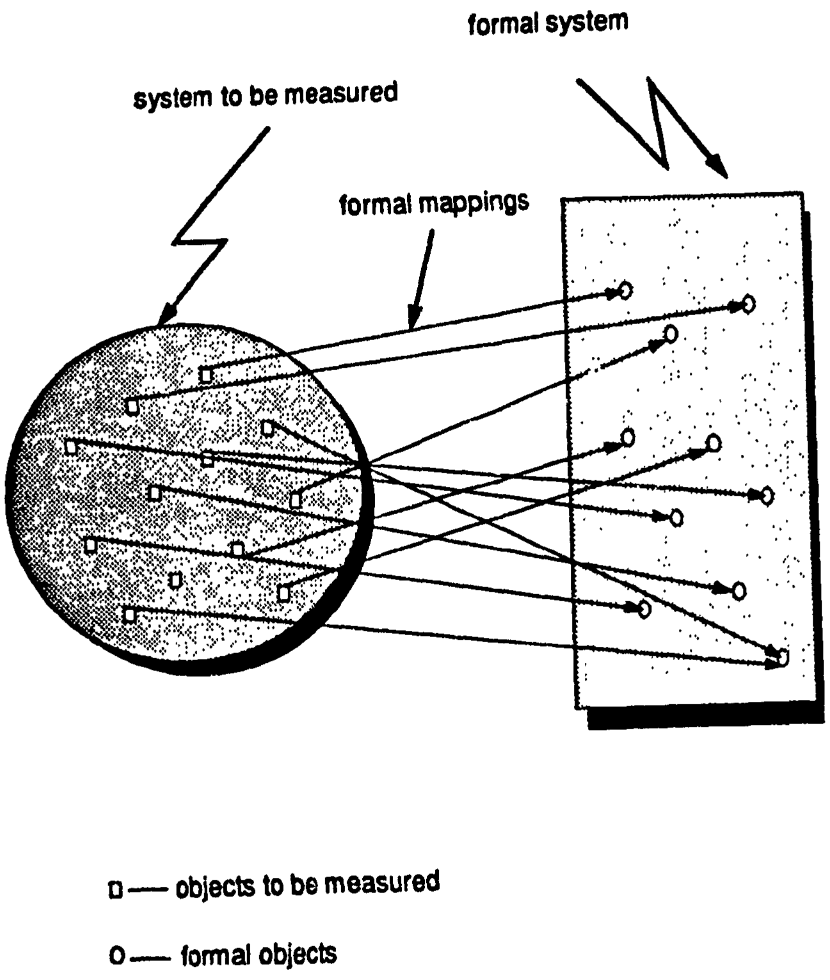


Figure 1: Mapping Between Empirical Relation System and Formal Relation System

1. Layer S_1: the set of direct mappings from $\mathcal{S}$ to $\mathcal{F}$, such as "number of variables", "number of distinct operators and operants" and program "size".

2. Layer S_n ($n \geq 2$): the set of metrics which involves at least one metric from layer $n-1$, that is:

$$S_n = \{\Gamma \mid \Gamma : \underbrace{S_1 \times S_1 \cdots \times S_1}_{k_1} \times \cdots \times \underbrace{S_{n-1} \times S_{n-1} \cdots S_{n-1}}_{k_{n-1}} \longrightarrow \mathcal{F}\}$$

   where, $k_{n-1} \neq 0$.

   The higher layer metrics are usually software complexity, maintainability, reliability and testability and so on.

The metrics in the first layer, i.e., the direct metrics, form the fundamental and primitive measures, and the metrics in the higher layers are usually defined in a broader sense, that is, they imply more meanings.

Example 2.4 Figure 2 shows the structure of some measures on the conventional software characteristics. Usually, the measures on *software quality factors* such as the *maintainability* lie in a higher layer, whereas the first layer metrics consist of the *number of lines of codes, number of operators, number of operands, etc.*

The layer structure clearly describes the structure of software metrics, and, in general, the formulations of different layer metrics can be basically described as follows:

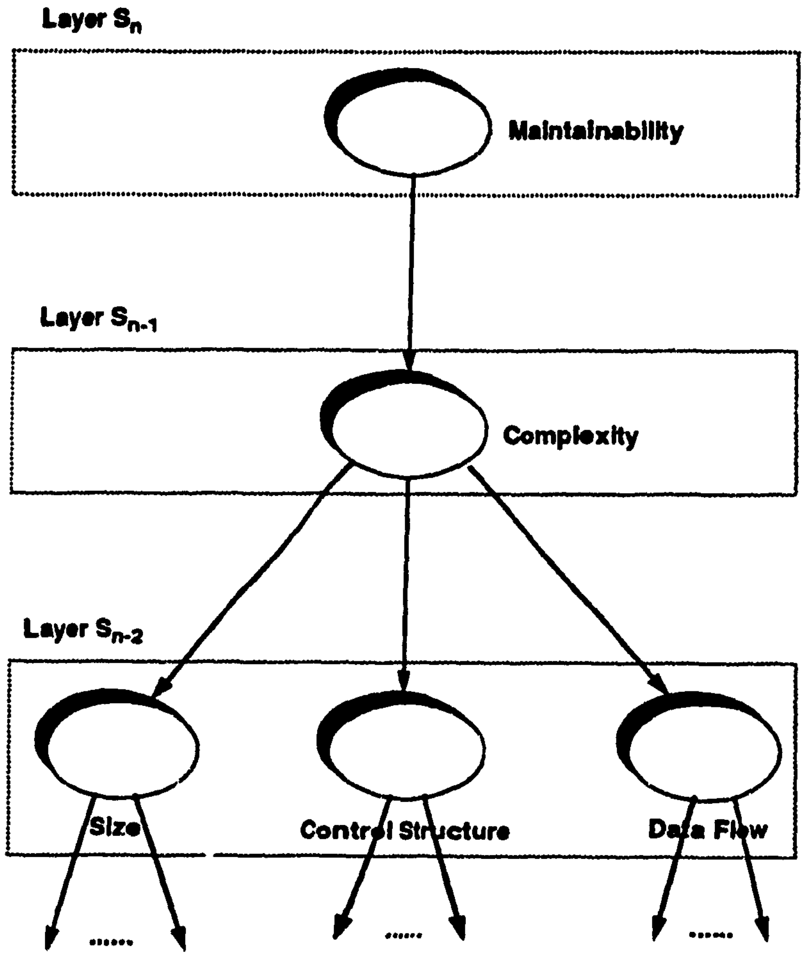


Figure 2: An Example of Metric Layer Structure

For first layer metrics:

1. Identify the empirical relation system and formal relation system.
2. Form proper mapping functions which satisfy the "representation conditions".

For higher layer metrics:

1. Identify the direct metrics in questions.
2. Weight the above metrics.
3. Form a proper function to summarize the direct metrics.

2.4 Metric Factors

There are several factors which affect the metric measurements, they are:

- Objectives. The reason for defining metrics is to get some feedback from the measures for further needs. The metrics without useful feedback are meaningless. However, having different objectives, some of them may be contradictory to each other, may require different types of feedbacks. Even for the same objective, the viewpoints on it could be quite different. This will lead to different measurement approaches. For example, there exist many metrics for measuring the complexity of conventional software, including (i) McCabe's control flow-based cyclomatic complexity [McCabe, 1976] which aimed at identifying the basic control paths in a program for "walking through"; data flow and massive computation are not the concern; (ii) Henry's information flow-based complexity metric [Henry and Kafura, 1981] that was designed to reflect the occurrence of changes, which was declared to be important; (iii) the complexity metric for software maintenance cost, which can be simply defined as the number of source lines of codes based on the claim that a larger program has a higher maintenance cost [Harrison et al., 1982].

- Scales. There exist five kinds of scales for metric measurements: nominal scale, ordinal scale, interval scale, ratio scale, and absolute scale [Fenton, 1991, Zuse, 1991]. It is more practical to have the ordinal scale for some higher layer (above first layer) metrics, because it provides just a comparison between different objects. For example, when comparing the maintainability of two programs A and B, it is feasible to give the result that A is more difficult to maintain than B. The reason is that higher layer metric measures depend on many factors including human aspects about which little is known. It is hard, sometimes impossible, to exactly measure these factors. However, relative comparisons can be made based on some existing attributes of the objects. But this kind of result has only limited use because some further operations cannot be performed on them [Zuse and Bollmann, 1989].

- Operations. Owing to the different natures of various metrics, it is not always meaningful to have some operations on these metrics, such as the addition of different metrics and statistical calculations. For the former, it should be done in an admissible way, at least it has an intuitive persuasion. Some metrics may have different multi-values depending on different points of views. So simply adding several measures together to obtain a single value for such metrics cannot reflect their real values adequately. For the statistical calculations, even for the measure on the same characteristic, we should still be cautious in applying them, because of restrictions imposed on them [Zuse and Bollmann, 1989].

- Use. Metric measures give only quantitative values for some attributes of the objects. They help to analyze the problems but cannot solve them. To further apply such metric measures may involve many other aspects such as the characteristics of the environment (object features, maintainer's experience, available tools, analysis of the relationship between the metric measures and the problems, *etc.*) in which such metrics are used and more work needs to be done.

- Precision vs. Expense. There is a trade-off between the precision of the measure and the expense required to obtain such a measure. It is obvious that the more precise the metrics are, the more expensive it usually is to get such metrics, since more information and processing are involved. For higher layer metrics, they are defined in terms of several other measures, so the measure of such higher layer metrics will involve some low layer metrics as well. If we consider too many factors, hence too many lower layer metrics, then the cost will be high for a higher layer metric. On the other hand, if too few factors are considered, it may be too coarse to be meaningful and useful.

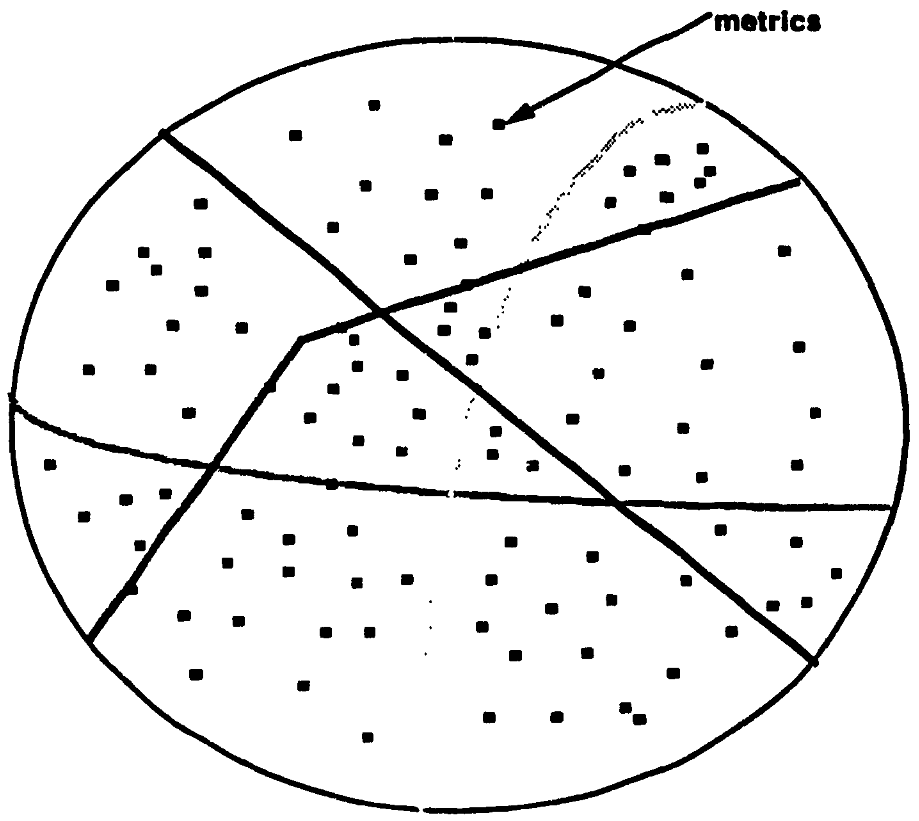
So, even if metrics are useful, there exist some conditions for them to be effective. The above factors must be considered before applying the actual measurements.

2.5 Metric Classification

In addition to the the systemic description of the metric structure — the layer structure, metrics can also be roughly classified into several categories in terms of their use:

- **Direct metrics vs indirect metrics** As discussed in the above section.
- **Static metrics vs dynamic metrics** Static metrics focus on the measures of static attributes of software products or processes that can be derived from the end products or process, while the dynamic metrics are obtained at run-time of the software that aim to measure the dynamic performance of the software.
- **Process metrics vs product metrics** Process metrics represent the measures on the features of software development processes, and product metrics are the measures of the attributes of the software products which could be both intermediate and final.
- **Control metrics vs predictor metrics** Control metrics are designed for the control and management of software, and predictor metrics are used to estimate some characteristics of software. In practice, the same metrics may be used for both purposes, but the ways in which they are used are different.

More classifications such as *volume measures, control flow measures, and date flow measures* [Basili, 1980] and *Code measures, design measures, and specification measures* [Sheppard, 1988] can also be seen. The above classifications are not disjoint. They may overlap among themselves as shown in Figure 3.



Boundary between the process and product metrics

Boundary between direct and indirect metrics

Boundary between static and dynamic mertics

Boundary between control and predictor merics

Figure 3: Software Metric Classifications

2.6 Scopes of Software Metrics

As mentioned before, the term "software metrics" include a wide range of diverse activities. They are denoted by the entire process of planning for measurement (identification of measurement goals, derivation of related metrics, etc), performing measurements (data collection and validation) and learning through data analysis. Fenton [Fenton, 1991] described the activities as follows:

- Cost and effort estimation models and measures
- Productivity measures and models
- Quality control and assurance
- Data collection
- Quality models and measures
- Reliability models
- Performance evaluation and models
- Algorithmic/computational complexity
- Structural and complexity metrics

It is obvious that these activities are not disjoint. They represent the main areas studied by the software metric researchers. In our study, focus is placed on the static metrics related to the quality model and measures of ES.

2.7 Use of Software Metrics

In general, the ultimate objective for developing software metrics is that they can assist both managers and software engineers in making decisions by providing valuable quantitative information. Figure 4 shows a general approach to use metrics, which includes the following steps:

- Set goals. The goals are established with respect to the desired performances of the objects to be measured. A quantifiable representation of the goals allows the precise examination of the metric measuring results. The goals will differ according to different needs and purposes.

- Formulate metrics. The metrics will be applied to the target objects and aim to provide quantitative values of the attributes concerned, which reflect the achievements towards the goals. They can be newly defined according to the measurement requirements of the goals or selected from existing metrics.

- Apply metrics. This denotes the measurements on the target objects. The metric measuring results are expected to be produced.

- Analyze results. Two tasks are to be performed here: (1) the interpretation of the result values, which can be performed by comparing them with the expectations and similar results obtained from other objects, (2) the assessment of the validity of the results, which can be achieved by statistical analysis of the metric measuring results.

- Adjust metrics. Analyzing metric values may reveal the deficiencies of the metric formulations, and this leads to the adjustment or re-formulation of appropriate metrics.

- Make decisions. Based on the measured results and predefined goals, the decisions could be made, such as to formulate a plan to correct any observed deviation from the goals.

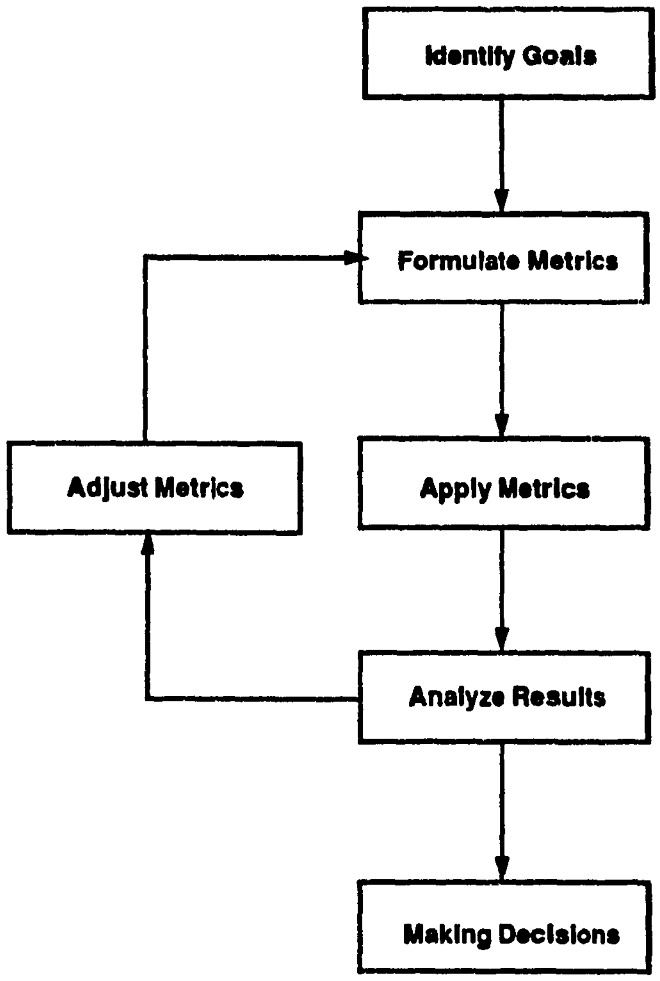


Figure 4: The Use of Metrics

Chapter 3

Conventional Software Measures

3.1 Introduction

Software engineering as a discipline has made considerable progress during the past few years. Researchers have gained much experience and practice during the past two decades. A significant issue in software engineering is the identification of the life-cycle model of conventional software development, which contains Requirement, Specification, Design, Implementation and Maintenance phases. As an effective way to predict and measure the characteristics of conventional software development in all phases, metric measurements have been applied throughout the life-cycle, and the Metrics Guided Methodology has been proposed [Ramamoorthy et al., 1985]. The design and use of metrics to assess and predict the quality of end products of conventional software is one of these applications, which is widely used, and empirical studies have already been conducted using such techniques [Li and Cheung, 1987, Elshoff, 1984, Harrison et al., 1982]. The typical software attributes measured are:

size, control flow, data flow, information content or a combination of the above attributes.

ES, as a special type of software, possess both conventional and unconventional portions, so the measures on them will be divided into those related to their conventional aspects and the ones on their unconventional aspects. In addition, there exist some similarities in the structures of conventional software and ES, so it would be helpful to study some mature techniques of conventional metric measurements used in software engineering, and utilize them to design ES measures. This also allows us to gain the experience and practice of conventional software measures for formulating ES metrics. In this chapter, we will review some conventional software measures. Measurement theory will be used to formally describe them. The comparisons between ES and conventional software and some characteristics of ES will be discussed in the next chapter.

3.2 Size-Oriented Metrics

This kind of measures focuses on the *size* of software products, which is the most basic and primary attribute of software. It is generally believed that *size* is a key component in the formulation of many higher layer measures, especially those related to software quality. Hence to date, many *size metrics* have be proposed and used. Some of them are:

Program length is obtained by counting the *lines of code (LOC)* in a program.

LOC is the commonly used size measure. The problems with it is that there does not exist a unique formalism (standard) for counting *LOC*. A diversity of size metrics was presented, each of which was based on certain assumptions and circumstances. A popular definition for *LOC* [Mőller and Pauliosh, 1993] is:

the count of program lines of code excluding comments or blank lines, that is typically given in units of thousands of lines of code.

Obviously, *LOC* is defined as the first layer metric and can be formally described as:

$$LOC: (\mathcal{E}, A_1, P_1) \longrightarrow (\mathcal{N}, T_1, R_1)$$

where:

$\mathcal{E}$: programs

A_1: size of a program

P_1: $(x, y) \in P_1$ if size of program x is greater than or equal to that of program y.

$\mathcal{N}$: natural numbers

T_1: values

R_1: $(x, y) \in R_1$ if $x \geq y$

$LOC(\mathcal{E})$: as defined above

Another question about the *LOC* is its reliability and applicability. In some situations, this kind of measure does not reflect the real features of the software, so its results sometimes could not be further applied for other uses. Consider the case in which a measure on program maintainability is needed, since *LOC* does not take into account the internal control structure of the programs, it is not appropriate to use *LOC* alone to predict or assess the maintainability of the program which is largely affected by the control paths in the program.

Software Science was proposed by Halstead [Halstead, 1977]. Aiming to capture more information than *LOC* as the size measure, it has been widely used as a typical software measure. In this measure, the basic elements are *operators* and *operands*, where *operators* denote the arithmetic and logic operators, keywords, and names of functions and procedures; *operands* represent the the variable names, constants and labels. The basic measures (first layer metrics) involved are:

$$n_1, n_2, N_1, N_2 \ : \ (\mathcal{E}, \{A_1, A_2\}, \{P_1, P_2\}) \quad \longrightarrow \quad (\mathcal{N}, T_1, R_1)$$

where

$\mathcal{E}$: programs

A_1: volume of operators in a program

A_2: volume of operands in a program

P_1: $(x, y) \in P_1$ if volume of operators in program x is greater than or equal to that of program y.

P_2: $(x, y) \in P_2$ if volume of operands in program x is greater than or equal to that of program y.

$\mathcal{N}$: natural numbers

T_1: values

R_1: $(x, y) \in R_1$ if $x \geq y$

$n_1(\mathcal{E})$= number of distinct operators

$n_2(\mathcal{E})$= number of distinct operands

$N_1(\mathcal{E})$= number of total operators

$N_2(\mathcal{E})$= number of total operands

Then, the second layer metrics are defined. They are:

$$n, N: \text{ Source Codes} \longrightarrow \text{Numerical Numbers}$$

$n = n_1 + n_2$ (vocabulary of a program)

$N = N_1 + N_2$ (length of a program)

Finally, the volume metric (V) (third layer metric) of a program is expressed as the necessary bits to represent the program, that is calculated as follows:

$$V: \text{ Source Codes} \longrightarrow \text{Numerical Numbers}$$

$$Volumn(V) = N \log_2 n$$

which provides an alternative measure for the size of a program. The volume measure has attained some prominence. Figure 5 summarizes the dependency structure of the above metrics. Several other measures were also suggested by Halstead such as ***Effort***, *Level* and *Difficulty*.

Software Science has been well-accepted in software engineering, even though it is still an active area of research.

3.3 Control Flow-Oriented Metrics

Control flow has been recognized as an important feature formed by the source code of the programs. A directed graph, $\mathcal{G}=(\mathcal{V}, E)$, has been used as the flow graph for representation of a program's control flow, where $\mathcal{V}$ represents a set of basic blocks of code, which contain no internal transfers of control and have a unique entrance and exit, E is the set of flows of control among the blocks shown by the edges connecting the different blocks. The control flow metrics are usually based on an analysis of this digraph $\mathcal{G}$. In the following, we will discuss some typical ones.

Number of decision nodes (ND) is the simplest control flow-based metric. It can be formally represented as:

$$ND: \ (\mathcal{E}, A_1, P_1) \longrightarrow (\mathcal{N}, T_1, R_1)$$

where

$\mathcal{E}$: control flows of programs

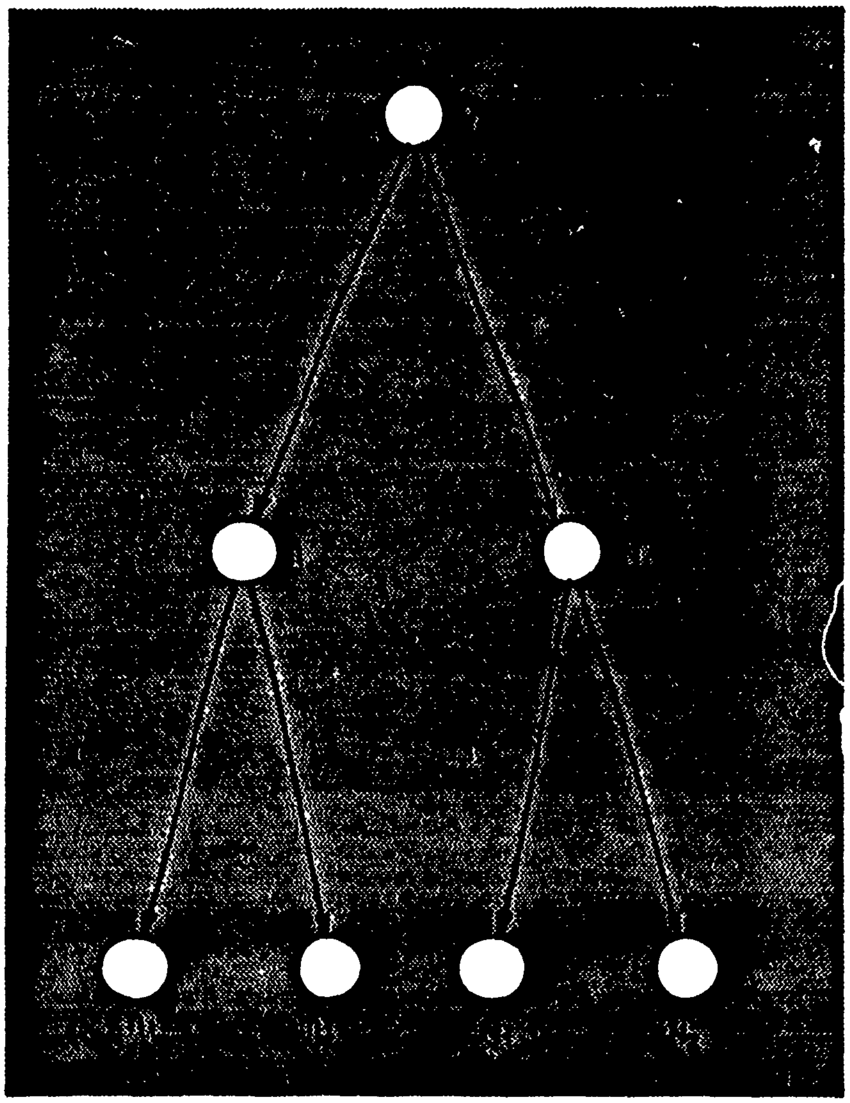


Figure 5: Layer Structure of Several Software Science Measures

A_1: size of a control flow

P_1: $(x, y) \in P_1$ if size of control flow x is greater than or equal to that of control flow y.

$\mathcal{N}$: natural numbers

T_1: values

R_1: $(x, y) \in R_1$ if $x \geq y$

$ND(\mathcal{E})$ = number of decision nodes in control flows

The decision nodes are sometimes represented as "diamonds" in the graph that denote the conditional and loop constructs in a program.

Cyclomatic number (v) is the best known control flow-based metric. It was proposed by McCabe [McCabe, 1976] and is based on two first layer metrics: *volume of edges (e)* and *volume of nodes (n)* which are:

$$e,\ n:\ (\mathcal{E},\ \{A_1,\ A_2\},\ \{P_1,\ P_2\}) \longrightarrow (\mathcal{N},\ T_1,\ R_1)$$

where

$\mathcal{E}$: control flows of programs

A_1: volume of edges in a control flow

A_2: volume of nodes in a control flow

P_1: $(x, y) \in P_1$ if volume of edges in control flow x is greater than or equal to that of control flow y.

P_2: $(x, y) \in P_1$ if volume of nodes in control flow x is

greater than or equal to that of control flow y.

$\mathcal{N}$: natural numbers

T_1: values

R_1: $(x, y) \in R_1$ if $x \geq y$

$e(\mathcal{E})$ = number of edges in a control flow

$n(\mathcal{E})$ = number of nodes in a control flow

Then, the *cyclomatic number* (v) (second layer metric) is defined as:

$$v : \text{Control Flows} \longrightarrow \text{Numerical Numbers}$$

$$v = e - n + 2$$

The valuable contribution of *cyclomatic number* is that it identifies the basic execution paths in a program, hence, it could be used to assess and predict the prog.am's testability.

The number of decision nodes and cyclomatic number are related. In the structural programs (i.e. single entry, single exit), the following relation exists.

$$v = ND + 1$$

The relationship can be even extended to multiple exit programs [Harrison, 1984].

So, decision nodes are actually the fundamental elements of control flow-based metrics which can be used as an objective means to characterize the control flow of a program.

The main problem with the control flow-based metrics is that they do not take into account the complexity of each node, that is, the contents of conventional statements. Thus, two modules or programs may have the same measuring results and yet be widely different due to the different uses of boolean operators, arithmetic expressions, nested and non-nested decisions, as well as the computation. Some attempts have been made to rectify this shortcoming [Fenton, 1991, Tai, 1984], which show promise.

3.4 Data Flow-Oriented Metrics

These metrics focus on the the use and flow of data, which are reflected by the data references. The work includes Elshoff's *span of data reference* [Elshoff, 1976] and Henry's *information flow* [Henry and Kafura, 1981], which are popular in this kind of measures.

Span of Data is based on the data reference points in a program, this method is a measure of the *distance* between references (Figure 6). One way to quantify the *distance* is to count the *number of statements* between the references to the same data with no intervening references to that data, and the final measuring result can be obtained by computing the mean of the maximum span for each data. The above idea can be formally expressed as follows:

For each data d_i, the *span* (D_i) is defined as:

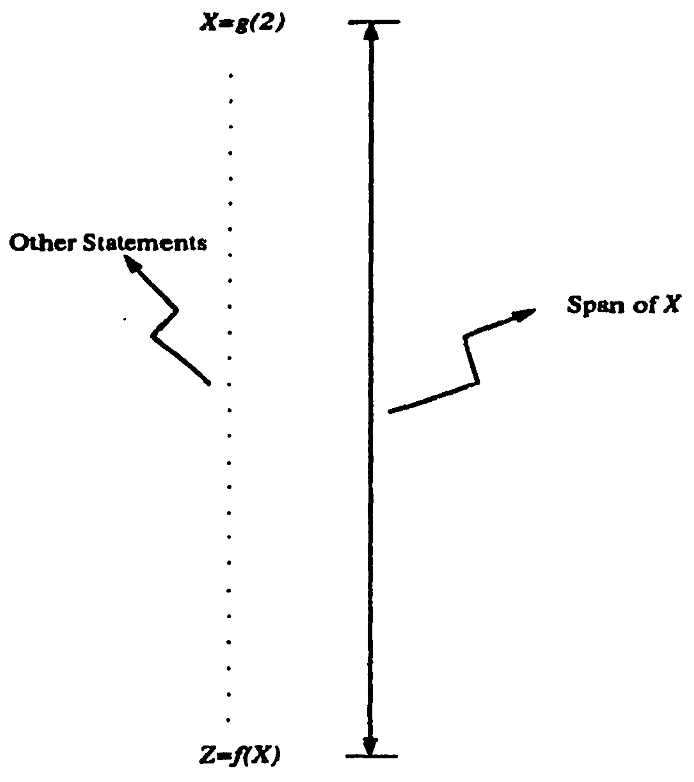


Figure 6: Span of Data Between References

$$D_i : (\mathcal{E}, A_1, P_1) \longrightarrow (\mathcal{N}, T_1, R_1)$$

where

$\mathcal{E}$: data in programs

A_1: reference distance for each data in a program

P_1: $(x, y) \in P_1$ if the reference distance for data x is greater than or equal to that of data y.

$\mathcal{N}$: natural numbers

T_1: values

R_1: $(x, y) \in R_1$ if $x \geq y$

$D_i(\mathcal{E})$ = number of statements between references for some data without intervention.

From the definition, we can see that there may exist several D_i values for each data, suppose the maximum of them is denoted as $max(D_i)$, then *span of data* (SD) for a program can be defined as:

$$SD : \text{Data} \longrightarrow \text{Numerical Numbers}$$

$$SD = \frac{1}{n}\sum_{i=1}^{i=n} max(D_i)$$

where,

n is the number of data contained in a program

It is obvious that the order of statements plays an important role in calculating the value of this measure. Since the more references to each data in a program, the more complicated the program is, the measure seems intuitively appealing, even though there is not much empirical evidence to support it.

Information Flow has received considerable attention and the flow between modules has been the focus. Different attempts have been made to quantify the information flow between the modules, hence the interactions among the modules have also been investigated. There are several activities with a module, such as *calling some module* or *being called*, and *retrieving or updating some data.* The two principle attributes associated with each module and used to describe these activities are *fan-in* and *fan-out.* They form the basis for defining information-based measures and can be described as:

$$fan-in, fan-out: \ (\mathcal{E}, \{A_1, A_2\}, \{P_1, P_2\}) \longrightarrow (\mathcal{N}, T_1, R_1)$$

where

$\mathcal{E}$: modules

A_1: calling and updating

A_2: being called and retrieving

P_1: $(x, y) \in P_1$ if module x performs more "calling and updating" activities than those of module y.

P_2: $(x, y) \in P_2$ if module x performs more "being called" and "retrieving" activities than those of module y.

$\mathcal{N}$: natural numbers

T_1: values

R_1: $(x, y) \in R_1$ if $x \geq y$

$fan\text{-}in(\mathcal{E})$ = number of flows that terminate at a module.

$fan\text{-}out(\mathcal{E})$ = number of flows that emanate from a module.

It is obvious that the flow is caused by *calling* some module or *being called* by some module, and *retrieving or updating* some data.

Henry established a measure (inf) for describing the information flow complexity of a module M [Henry and Kafura, 1981], which is:

$$inf: \text{ Modules } \longrightarrow \text{ Numerical Numbers}$$

$$inf(M) = length(M) \times (fan\text{-}in(M) \times fan\text{-}out(M))^2$$

The formula is expected to measure the "degree of complexity of relationships between subsystems". It depends on (1) the complexity of a module which is simply given by the length of the module in the measure; (2) the complexity of the interrelations between the module and other modules, which are determined by the *fan-in* and *fan-out*. The validity of the measure was shown by applying the measure to the industrial software — *UNIX* operating systems and exhibiting a strong correlation between the measuring results and the occurrences of changes in the *UNIX* system. It was further found that actually the measure of the connections of a module to its environment ($(fan\text{-}in \times fan\text{-}out)^2$) is an extremely good indicator of the modular complexity, and the length measure is not so reliable and necessary.

3.5 Information Content-Based Metrics

Since Channon [Channon, 1974] applied the information theory to analyze the design of software structure, information theoretic concepts have been widely used for different purposes.

In the field of *information theory*, a message is made of a string of alphabet symbols s_1, s_2, ..., s_m. The term *information* refers to some things in a message that are uncertain and unlikely to occur. And the *information content* is denoted as a function that decreases in numerical value as the probability of occurrence of s_i $(1 \leq i \leq m)$ increases [Ingels, 1971]. The information content I_i conveyed by a single symbol s_i is inversely proportional to the probability of occurrence associated with s_i. The formula for calculating this amount is:

$$I_i = -\log p_i \ (bits)$$

where p_i is the probability of occurrence of s_i. Information measure is additive; i.e., the average information (I) of the combination of s_1, s_2, ..., s_m is the weighted average of information content per symbol s_i.

$$I = -\sum_{i=1}^{m} p_i \log p_i \ (bits)$$

The term *entropy* is used to represent such average information content.

There are several properties associated with *entropy*. For example,

- I achieves its maximum value $\log m$ when $p_1 = p_2 = \ldots = p_m = \frac{1}{m}$; i.e., when the source symbols are equally likely.
- I achieves its minimum value 0 when $p_i = 1 (1 \leq i \leq m)$, $p_j(1 \leq j \leq m \land j \neq i) = 0$; i.e., one symbol always occurs, and all others do not.

Entropy-Based Complexity Measure: A piece of software can be considered as a message in which the information is conveyed in the source code. One metric for measuring such information was proposed and used by Harrison [Harrison, 1992]. The idea is to examine the empirical distribution of *operators* within a program, which can be a special symbol, a reserved word, or a function call.

First, the measure (f_i) of occurrences of an operator is performed, which is:

$$f_i : \ (\mathcal{E}, A_1, P_1) \ \longrightarrow \ (\mathcal{N}, T_1, R_1)$$

where

$\mathcal{E}$: operators in programs

A_1: occurrence of operators in a program

P_1: $(x, y) \in P_1$ if operator x in a program occurs more often than or as often as that of operator y.

$\mathcal{N}$: natural numbers

T_1: values

R_1: $(x, y) \in R_1$ if $x \geq y$

$f_i(\mathcal{E})$ = number of occurrences of i-th operator

Then, the total number (second layer metric N) of occurrences of all operators is obtained by

$$N : \text{Source Codes} \longrightarrow \text{Numerical Numbers}$$

$$N = \sum f_i$$

And the probability p_i (third layer measure) for an operator is

$$p_i : \text{Source Codes} \longrightarrow \text{Numerical Numbers}$$

$$p_i = \frac{f_i}{N}$$

Finally, the entropy (I) (fourth layer measure) of the program is

$$I : \text{Source Codes} \longrightarrow \text{Numerical Numbers}$$

$$I = -\sum p_i \log p_i$$

3.6 Hybrid Metrics

The purpose of defining hybrid metrics is to avoid the shortcomings of single-factor measurements. Several program properties are taken into account in this type of measures, such as the program size, control flow, data reference, which are thought to contribute to the characteristics of a program.

Li and Cheung presented a high layer hybrid metric called New_1 with some comprehensive results [Li and Cheung, 1987]. Their measure combined the scope idea [Harrison and Magel, 1981a, Harrison and Magel, 1981b] with "software science" [Halstead, 1977], and was represented by

$$New_1 : \text{Source Codes} \times \text{Control Flows} \longrightarrow \text{Numerical Numbers}$$

$$New_1 = (1.0 - \sum RawComplexities / \sum AdjustedComplexities) * 100\%$$

where

The *raw complexity* measure is related to the volume measure of a node and its calculation is based on the *operators* and *operands* contained in a node.

The *adjusted complexity* measure is expected to act as a measure of the control structure and the calculation is in terms of the individual "influence" of a selected node on other nodes in the control flow.

Li demonstrated that his measure is sensitive to many factors of a program, which may reflect different aspects of software, but, no experimental evidence has been given to show the relationship between the NEW_1 and some system quality characteristics such as the *error rate*, which is also an important part for assessing the validity of a measure.

3.7 Other Issues

It is generally believed that the cost of software vastly exceeds that of hardware. Indeed, it has been estimated [Rushby, 1988] that software accounts for 80% of the total computer system budget of the U.S. Department of Defense, and 60% of the software budget may be spent on maintenance. This leads to the formulation of different procedures and methodologies for the measurement of software products, in order to cure the software quality problems which produce the cost [Suen et al., 1989].

Software quality is represented by its characteristics and software metrics refer to a broad range of measures applying to the software to quantify these characteristics. Most of the techniques and approaches proposed to measure the software products are based on static analysis.

Empirical studies support the notion that hybrid metrics measurements which take into account different factors of the software product, could give more reliable and valid results for general cases [Li and Cheung, 1987, Tai, 1984, Harrison et al., 1982]. Measurements based on only one factor certainly cannot characterize the software product adequately.

Software measures, like others, form the basis for estimating, evaluating, and explaining some phenomena. However, it should be noted that measures only identify problems; and they do not solve the problems [Card and Glass, 1981]. Software metrics measurement can only assist the software maintainer or developer in making decisions and improving effectiveness.

Chapter 4

Characteristics of Expert Systems

4.1 Introduction

Expert Systems are a special type of software, designed to capture valuable human expertise. They have been increasingly used in a variety of domains. Successful commercial applications include Digital Equipment Corporation's XCON configuration advisor, which saves $18 million a year [Williams, 1986]. However, the special features of ES such as the complex *dependency* among the rules, make them subject to more quality problems than the conventional software. Besides, many matured techniques and methodologies used for the development of conventional software cannot be applied to expert systems directly [Grogono et al., 1991], so a lot of research needed to be done in order to assure their quality. One kind of research deals with the verification and validation of ES, and some issues regarding it were already introduced [Liebowitz, 1990]. Another kind of research presented in this thesis focuses on the measurement of ES, which is also critical and important. To measure ES, we must first understand the characteristics of ES, especially those that are different from conventional software. The following sections will address them.

4.2 System Components

A typical ES consists of four essential components:

1. Acquisition Module
2. Knowledge Base
3. Inference Engine
4. Explanatory Mechanism

The relations among these components are shown in Figure 7. Of the components, the Acquisition Module is responsible for acquiring the knowledge (expertise) from the experts, and the Explanatory Mechanism will provide the answers and explanation about the reasoning made by ES to the users. The kernel parts of an ES are the *knowledge bases* and the *inference engines* [Williams, 1986, Forsyth, 1984], which will be described in the following sections.

4.2.1 Knowledge Base

A knowledge base of an ES captures and represents the knowledge for solving problems, and it is unique to a particular domain, i.e. it varies as the domain differs. The past experience indicates that the performance of an ES largely depends on the quality of the implemented knowledge base [Gevarter, 1982].

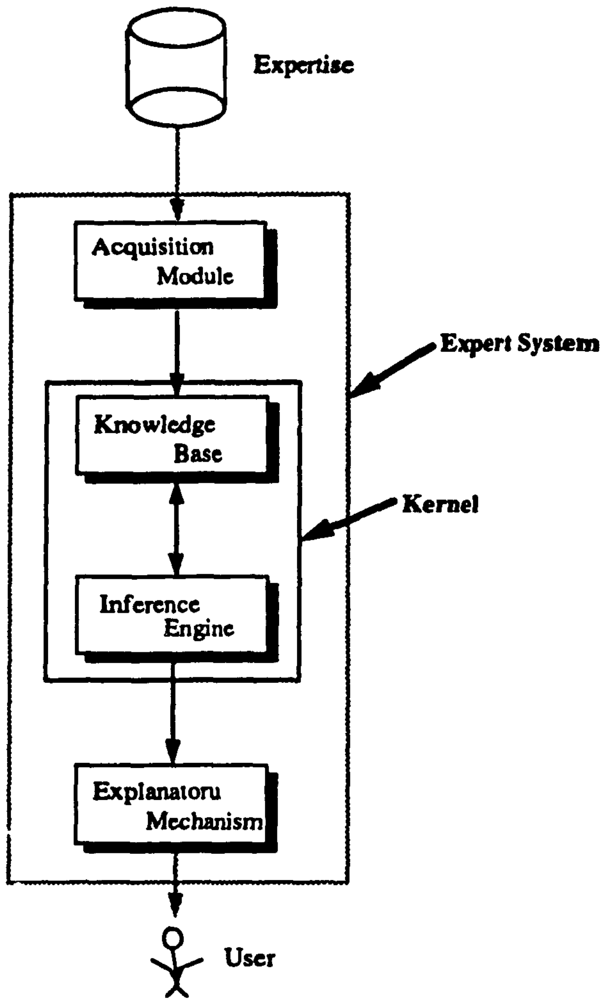


Figure 7: A Typical Expert System

The most prevalent approach to represent knowledge is by ***production rules***, hence, a knowledge base (rule base) is made of ***if LHS then RHS*** rules, where the Left-Hand-Side (*LHS*) represents the conditions and the Right-Hand-Side (*RHS*) is a list of actions to be taken when the rule fires. The *LHS* and *RHS* existing in different rules construct the ***dependency*** or the chaining of rules. This kind of ES is also called rule-based ES which has been proven particularly useful because of its ease of development [Kiper, 1992]. Some advantages [Landauer, 1990] with them are:

1. local specification of the relation by means of rules, which allows for the decoupling of different kinds of knowledge,

2. making domain knowledge explicit.

The main disadvantage is that the complicated interactions resulting from the rules may cause severe quality problems because of the potential errors (anomalies) that may be introduced during the construction of rule bases. These errors may come from different sources associated with the rule bases. Some of them are never noticed or are noticed too late to make amendments. So, even though the rule bases may be easy to build and use, it is sometimes risky to use them, especially in critical situations.

The rules in rule bases are the data to be used by the inference engine.

4.2.2 Inference Engine

For rule-based ES, the inference engine represents the control strategies for the "execution order" of the ***firable rules*** whose *LHS* parts matched some currently available facts, and the search strategies for seeking an answer, such as *forward chaining* and *backward chaining* or the integration of both. *Forward chaining* forms a line of reasoning from the existing facts, and new facts are derived by firing the firable rules. *Backward chaining* moves from a given goal, searches the paths (facts and rules) that derive the goal. In addition to forward and backward chaining, there also exist other strategies. Details and formal descriptions of these strategies can be found in the AI literature, see e.g. [Shinghal, 1992].

The development of an inference engine is independent of a rule base, and an inference engine may be applied to a number of knowledge bases, that is, the inference engine is not specific to one application. Also, because of its similarities in function to conventional software; that is, both are the implementations of some algorithms for the purpose of control, the traditional software measuring techniques may be applied to the inference engine [Barrett, 1990, Kiper, 1992].

From the above discussion, we can see that it is the knowledge bases that distinguish ES significantly from the conventional software, hence, they play an important role in the formulation of ES quality. This also explains why many studies place their emphases on the investigation of knowledge bases. In our study, we will focus on the rule bases, the popular implementation of knowledge bases, and the measures of rule base attributes, especially those related to their complexity.

$\langle rulebase\rangle ::= \langle rule\rangle \mid \langle rule\rangle\langle rulebase\rangle$
$\langle rule\rangle ::= \langle rule-id\rangle$ if $\langle LHS\rangle$ then $\langle RHS\rangle$
$\langle LHS\rangle ::= \langle pattern\rangle \mid \langle pattern\rangle\langle logic-op\rangle\langle LHS\rangle$
$\langle RHS\rangle ::= \langle pattern\rangle \mid \langle pattern\rangle\langle logic-op\rangle\langle RHS\rangle$
$\langle pattern\rangle ::= \langle items\rangle \mid not\ \langle items\rangle$
$\langle items\rangle ::= (\langle qualifier\rangle, \langle value\rangle)$
$\langle qualifier\rangle ::= \langle symbols\rangle$
$\langle value\rangle ::= \langle symbols\rangle$
$\langle symbols\rangle ::= alphabetstring \mid number \mid alphabetstring\langle symbols\rangle \mid number\langle symbols\rangle$
$\langle logic-op\rangle ::= and \mid or$
$\langle rule-id\rangle ::= alphabetstring$

Table 1: Formal Description of the Generic Language

4.3 Rule Base Description Languages

Like conventional software that is implemented by using programming languages, rule bases are built by using different rule base description languages. So far, many description languages exist, such as those used in EXSYS, LEVEL 5, OPS5, EMYCIN and CLIPS. The analysis of these languages is not the focus of this thesis. Our concern relates to the rules and the interactions among the rules, which would affect the quality of the systems. In order to study rule bases written in different languages, we propose a simple and generic rule base description language into which other languages can be converted. Table 1 gives the syntax definition of our generic language. The *LHS* and *RHS* are made of *pattern* lists and each rule is expressed as '*if* $\langle pattern\rangle$

and/or ⟨***pattern***⟩ ***and/or*** ...... ⟨***patterns***⟩ ***then*** ⟨***patterns***⟩ ***and/or*** ⟨***patterns***⟩ ***and/or*** ......⟨***patterns***⟩". The operator precedence is: ***or, and, not,*** from low to high. A matching happens between a pair of patterns existing in the *LHS* and *RHS* parts of some rule(s).

4.4 Rule Dependency

The most distinctive and important feature of ES is the ***dependencies (interactions)*** among the rules, which are caused by the matching of the existing ***patterns*** that represent the conditions and consequents of different rules. That is, suppose R represents a rule base, then the *dependency* between a pair of rules, r_i and r_j, can be formally described as:

$$\exists r_i \cdot (r_i \in R)\, \exists r_j \cdot (r_j \in R) \cdot (LHS_i \in r_i \wedge RHS_j \in r_j) \wedge (\exists pat_i \cdot (pat_i \in LHS_i)$$
$$\exists pat_j {\cdot} (pat_j \in RHS_j) \exists \psi {\cdot} (pat_i \psi = pat_j \psi))$$

where

LHS_k = the LHS part of the rule r_k $(k = i, j)$

RHS_k = the RHS part of the rule r_k $(k = i, j)$

ψ = a general substitution $\{t_1/v_1, t_2/v_2, ..., t_n/v_n\}$. t's are terms, v's are distinct variables.

The overall dependencies of an ES form a search space and different search paths, which could sometimes be enormous and hard to follow.

Because of the critical role of the search space and its characteristics, a mechanism is first needed to formally describe them. This thesis presents an extended AND/OR digraph for such a need which will be discussed later.

4.5 Rule Base Anomalies

Because of evolutionary feature of rule bases, anomalies, i.e. unusual structures, are introduced during the development of the system. Anomalies may reveal many faults. However, the anomalies do not necessarily mean real errors, they indicate *potential errors*. They also reflect the quality of rule bases.

Four classes of anomalies are defined by Preece *et al* [Preece and Shinghal, 1991]. They are:

1. Redundancy: An expression in ES, whose presence or absence will not affect the inferring of any conclusions. Redundant situations can be further classified as: *unusable rules* and *redundant rules*.

2. Ambivalence: An impermissible (incompatible) set of conclusions inferred by the ES. Ambivalence includes *impermissible set of hypotheses inferred* and *inconsistency* existing in the ES.

3. Circularity: A loop occurring in some rule chain of ES; i.e., a circular dependency in a set of rules.

4. Deficiency: A set of permissible inputs for which ES will infer no conclusion. Some symptoms of deficiency in an ES are the presence of *unused literals* and

missing values.

Anomaly detection can be performed by the static inspection of ES.

4.6 Comparisons With Conventional Software

In order to adapt many relatively mature techniques used in software engineering for conventional software, including some metric measuring techniques, it is essential to recognize the differences between ES and conventional software.

In general, ES possess many special features that distinguish them from the conventional software. Some typical aspects are:

- Application domains. The purpose for using ES is to apply them to solve difficult problems in the complex domains. Some typical application domains of ES are *fault diagnosis* in the telecommunication or aerospace equipment, *disease diagnosis* in the hospital, *system configuration* in the engineering design and *financial advising* in the business. Due to the difficulty in expressing these complicated domains in a formal way, the knowledge about the domains is usually ill-defined and imprecise. On the contrary, the application domains in software engineering are well understood and specified. There are formal and precise quantitative ways to describe the domains.

- Solutions. The solutions represented by the knowledge for solving complex problems in ES are usually based on heuristics, experience and intuition. Most of them are the rules of thumb, and no one understands beforehand exactly how human experts form conclusions, not even the experts themselves. While in software engineering, the solutions represented by the different formulas and equations are purely based on theoretical grounds. The software for solving these problems is a straightforward implementation of the theory, that is, turning known procedures or algorithms into codes.

- Development processes. The development process for most conventional software can be described by the "life-cycle" phases. The completion of a phase is the basis for starting the next one. However, in the development of ES, the iterative method is widely used in order to support the exploratory nature of the knowledge and domain, for example, rapid prototyping has been the de facto method for developing ES.

- System structure. This includes the system constituents and organization. The major components of ES are the *knowledge bases* and *inference engines*. However, for general conventional software, there are no definite components that must be present to form the systems. Different software may have different system configurations that are determined by the application requirements. In addition, the basic elements of designing ES are the *rules*, whereas it is the individual conventional statements that constitute the element of conventional software design in software engineering. Each rule in ES acts like a decision node, but is different from the conventional statement. The *dependencies* among the rules constitute more dynamic search paths than the relatively static characteristics of conventional software; the number of paths may sometimes be enormous

Expert System	Conventional Software
rules	conventional statements
rule dependency	data reference
rule bases	source code
inference engine	source code
forward/backward chaining	sequential execution
reasoning	calculation
shells	programming languages
imprecise specification	precise specification
complex application domain	clearly defined domain
expert-level performance	data processing
iterative development	life cycles

Table 2: A Comparison Between Expert System and Conventional Software

because of the combinatorial explosion of the possibilities of the paths.

- Execution order. In conventional software, it is the statements themselves that decide the order of the execution of the systems. But, for ES, the rules themselves do not directly specify the order of the execution, the order is mainly decided by the separate inference engine.

- and other aspects. Some other differences between ES and conventional software include the development team organization, system construction and so on.

Even though ES have so many features that distinguish them from conventional software, they also possess some characteristics in common with conventional software, for example, they contain the arithmetic operations, function or procedure calls and they employ variables as well. Table 2 summarizes a comparison of rule-based ES and conventional software on some characteristics.

Bearing the above similarities and differences in mind will help the correct analysis of the problems, appropriate design of ES metrics and proper implementation of the measurements.

Chapter 5

Quality Model of Expert Systems

5.1 Introduction

The quality of software (conventional software and expert system) is an important feature, which shows the degree of excellence that a software product or development process possesses. However, the viewpoints about it are subjective and they vary from system to system. Traditionally, quality is described as a mix of factors that reflect the different facets of products or processes. The measuring of these facets relies on the primitive software quality metrics such as the *size*, and *data span*. In this chapter, we will discuss and present the different quality facets of software products, especially the the *maintainability, testability* and *reliability* that significantly contribute to the ES quality.

5.2 Software Quality Facets

In general, *quality* is a term which could be applied to all the target objects. According to the *IEEE Standard Glossary of Software Engineering Terminology* [IEEE, 1983], software quality means:

The degree to which software possesses a desired combination of attributes.

Based on this definition, a quality model can be derived using a hierarchy structure as shown in Figure 8, where the nodes represent three types of characteristics: *quality, quality criteria* and *quality metrics*; the arrows indicate that the presence of the characteristics on the left-hand side of the arrows depends on the presence of the characteristics on the right-hand side. The characteristics on the first level are usually called *quality factors* which characterize the quality directly. The characteristics on the second level are defined as *quality criteria* which specify the factors, and the characteristics on the last level represent *quality metrics* which are the basic elements of the hierarchy structure. The basic idea behind the hierarchy structure is *characteristic decomposition*, that is, breaking down the higher level characteristic into several other more precise characteristics in the lower levels. There exist several commonly accepted *quality factors* [Manns and Coleman, 1988, Deutsh and Willis, 1988, Arthur, 1985, Schulmeyer, 1987], which are:

- Correctness. The extent to which the software design and implementation satisfy user's requirement specifications.

- Efficiency. The amount of resources needed by the software to perform its intended functions.

- Flexibility. The effort required to modify the software.

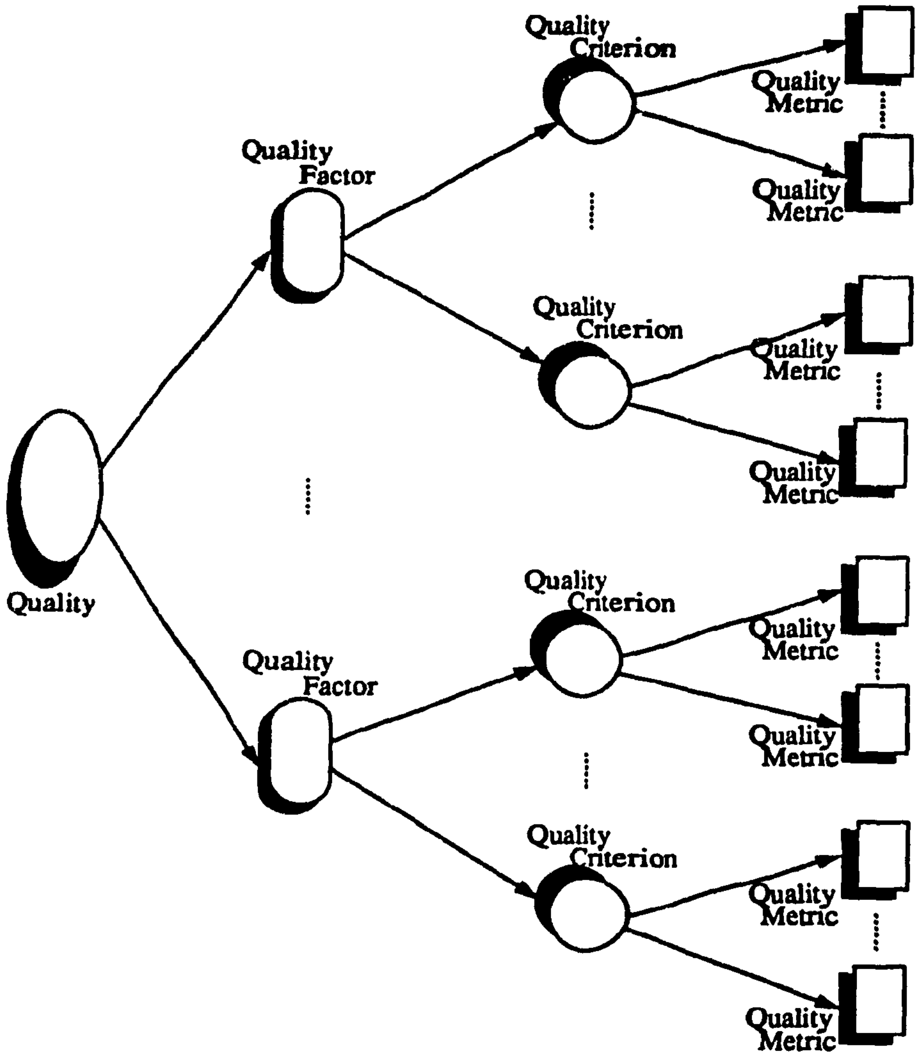


Figure 8: Software Quality Model

- Integrity. The extent to which the software can be protected against either overt or covert access to the software without authorization.
- Interoperability. The effort required to couple the software with software in other systems or applications.
- Maintainability. The ability to modify the software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a changed environment.
- Manageability. The degree to which the environment supports the software.
- Portability. The effort required to transfer the software from one environment to another.
- Usability. The effort required to learn, and the recurring effort to use, the functionality of the software.
- Reliability. The ability of a program to perform a required function under stated conditions for a stated period of time.
- Reusability. The extent to which the software can be reused in other applications.
- Survivability. The extent to which the software could continue reliable execution in the presence of a system failure.
- Testability. The effort required to test a program in order to ensure that it performs its intended functions.

So, in general,

$$\begin{aligned} quality = \mathcal{F}(&correctness, efficiency, expendability, flexibility, integrity,\\ &interoperability, maintainability, manageability, portability,\\ &usability, reliability, reusability, safety, survivability, testability) \end{aligned}$$

where $\mathcal{F}$ represents some function.

These factors represent the different facets of the software quality, and there may exist conflicts or overlapping among these facets. The desirability of the facets varies and depends on the different quality requirements.

Example 5.1 The software which is used by a government agency run solely on one machine, and is not shared with other agencies. The *portability* has a lower value in this situation, whereas for a software library, the *portability* is important [Schneiderman, 1980].

The *quality criteria* may be further identified as:

- Accuracy. Software attributes that achieve the required precision in calculation, data and output,
- Completeness. Software attributes that demonstrate a full implementation of the required functions,
- Complexity. Software attributes that reflect the difficulty to understand the software,

- Conciseness. Software attributes that implement a function in the minimum amount of code,
- Consistency. Software attributes that provide uniform design and implementation techniques and documentation,
- Modularity. Software attributes that provide a structure of highly functional independence of modules,
- Self-documentation. Software attributes that explain the functions of the software,

and so on.

The final *quality metrics* denote the quantitative measures applied to the target objects, which may help to evaluate the extent to which the criteria exist. They vary as the needs differ.

There are two components that may affect the assessment of the quality factors and criteria (such as the maintainability and complexity). One is the *external components* which reflect the influences of the development environment, such as the use of tools, maintainer experience and skill, cooperation among maintainers, documentation, and so on. The other is the *internal components* which purely relate to the software itself (Figure 9). Since the *environment* is sensitive to many factors and hard to measure, this thesis focuses on the studies of some static internal factors. Comparative results will be given for such studies.

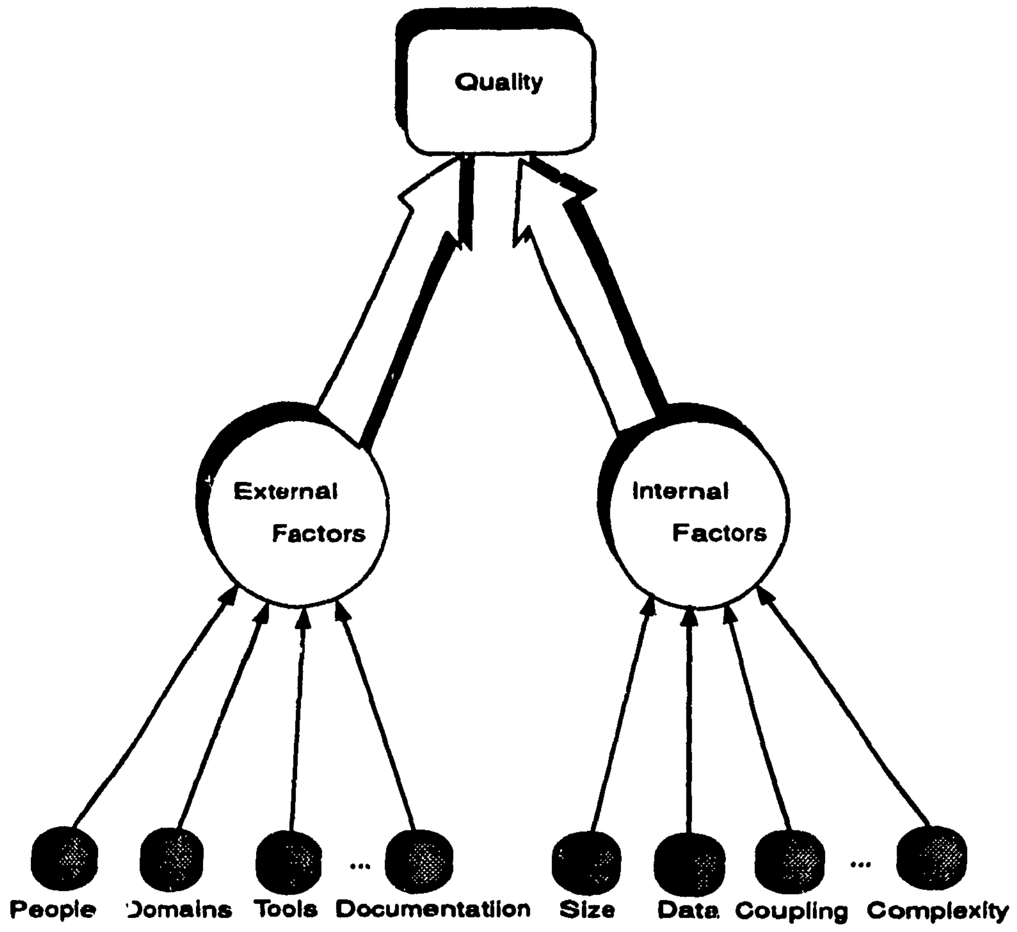


Figure 9: Components Affecting Software Quality

5.3 Expert System Quality

The general software quality model (Figure 8) can also be applied to ES. However, due to the differences between conventional software and ES, the quality requirements may be different. This is reflected in the shift of emphasis of the quality factors, that is, the degree of the presence of the different quality factors. So far, there is not one standard model to which all software must adhere [Arthur, 1985].

As discussed in the previous chapters, ES possess many distinctive features that make them different from other software systems. Because of these special features (such as the imprecise and ill-defined nature of the knowledge), certain quality factors are required to be presented in sufficient degrees in order to assure ES quality. Three main quality factors for ES are *maintainability, reliability* and *testability.* Figure 10 indicates the features that make maintainability, reliability and testability important, which will be briefly discussed in the following sections.

5.4 Maintainability

ES maintenance is harder and more expensive than that of conventional software, due to the special features discussed above. Since ES are subject to continuous modifications, changes and corrections, so maintenance of ES is critical and it plays a key role in ES quality.

Example 5.2 The *iterative development* feature of ES can be seen from the XCON development. XCON, a rule-based expert system that configures computer systems,

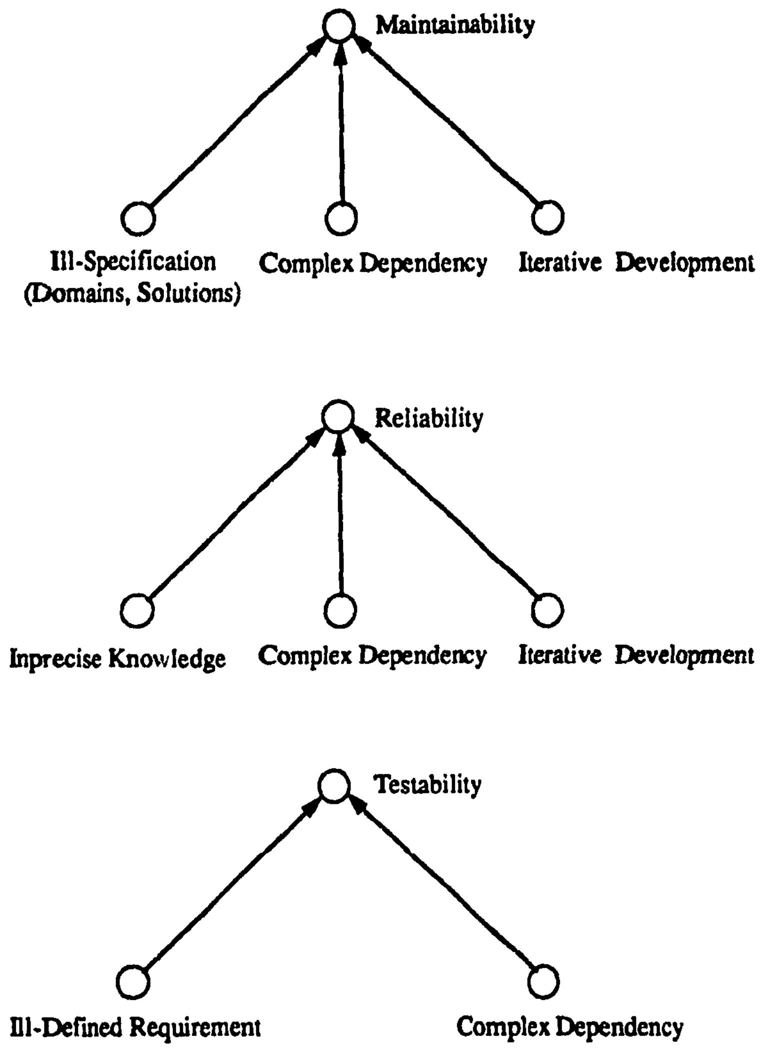


Figure 10: Features and Quality Factors

has grown from 700 rules to 6200 rules, of which approximately 50% change every year in order to reflect new products and new computing concepts coming out of DEC [Soloway, 1987]. Now, XCON has over 11,000 rules.

So far, several attempts have been made to increase the maintainability of ES [Philip, 1993, Jacob and Froscher, 1990, Prerau et al., 1990, Pedersen, 1991, Soloway, 1987], but, because of the lack of comprehensive experiments and test data, neither quantitative nor comparative results were given to show how much the maintainability has been improved.

5.5 Reliability

When applied to ES, reliability denotes the correct portion of the knowledge contained in an ES and is reflected by the variance of the inferred results from the ES for the same environment (data). Due to the heuristic nature of the knowledge (expertise) and the vague requirement specification, ES is usually less reliable than conventional software developed from precise specification and well-defined solutions. So, they are "fragile" in the sense that they are sensitive to many external factors, and that they may therefore contain many potential problems. Also, as an ES changes over time, it can lose its integrity [Carrico et al., 1989], and hence its reliability.

5.6 Testability

Generally, the testing concepts and methods applied to conventional software can also be borrowed or adapted to expert systems [Rushby, 1988]. However, there exist several difficulties that may obscure the testing of ES, including the following:

- Lack of explicit specification for the requirements, implemented functions and methods used to solve problems.
- Complex rule dependencies which may produce exponential search paths for solving problems.

In general, the more effort we spend on the testing, the greater confidence we can have in the ES performance, but the more expense we must pay to gain this confidence.

It is obvious that the *maintainability, reliability* and *testability* of ES are all affected by three primitive and static characteristics of ES: *size, search space* and *complexity.* Hence, the measures of them are investigated in this thesis research. Figure 11 shows some relationships among them.

5.7 A Narrower View of Expert System Quality

The software quality model presented in Figure 8 denotes the ideal theoretical consideration. However, in the real world, extra resources and project overheads are required in order to assess and measure the model using the decomposition approach, even for just a small number of quality factors; managers are often reluctant to accept these extra costs [Fenton, 1991]. This situation also occurs in ES quality assurance. So it is sometimes feasible to have a rough measure of the quality, which costs less. In software engineering, successful applications of this idea can be seen,

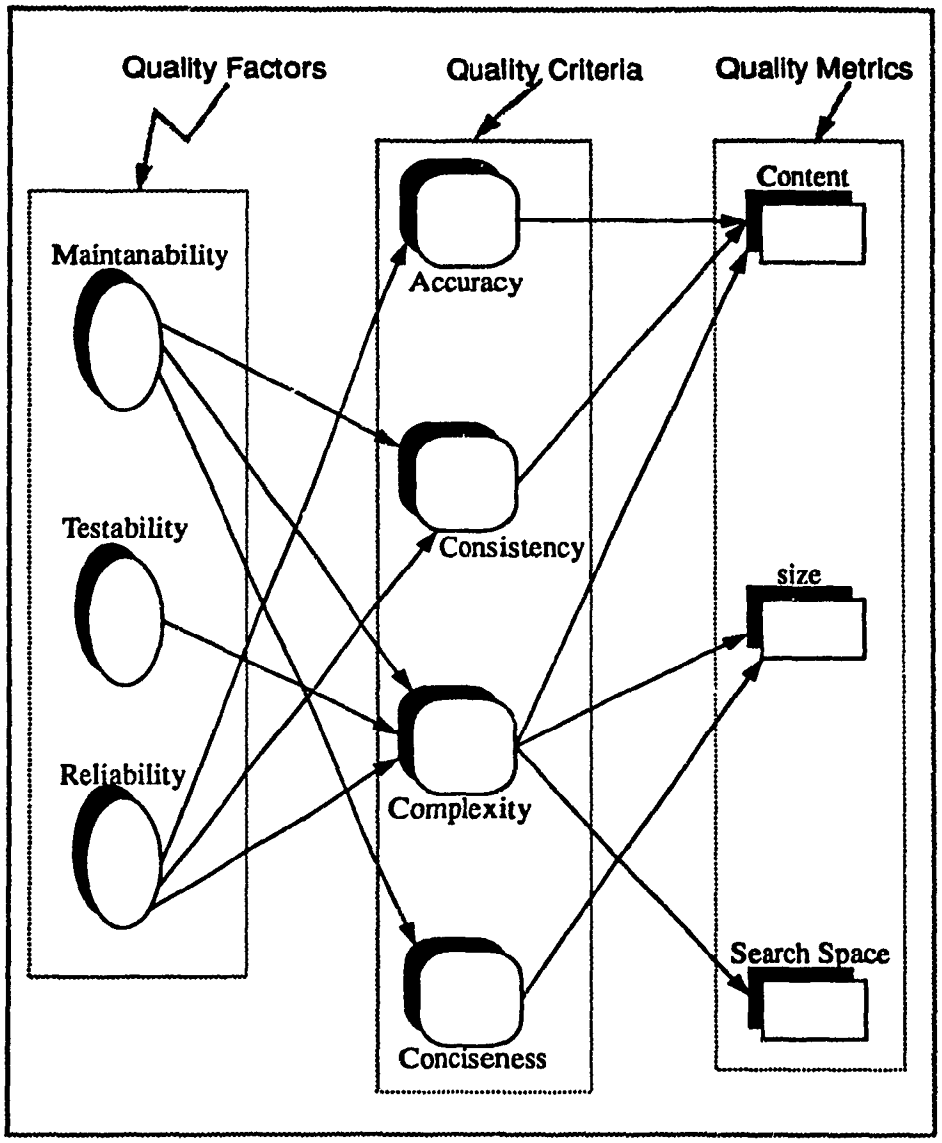


Figure 11: Relationship Between Several ES Characteristics

where the *software defects* were used as being synonymous with the software quality [Grady and Caswell, 1987, Inglis, 1985, Tajima, 1981]. From this narrower viewpoint, *ES anomalies,* the counterpart of *software defects,* can also be used as the *indicator* of ES quality. The basic rationale for choosing *anomaly rate* also includes:

1. Certain relationship existing between *anomalies* and *quality.* That is, the higher the anomaly rate of an ES, the lower the ES quality.

2. Reasonable cost to detect the anomalies contained in an ES, because only static analysis and checking of the ES are needed.

So, anomaly is also an important feature of an ES. It has attracted more and more attention in research on verification, validation and testing of ES. It can be used either as an attribute of an ES or quality indicator from a narrower point of view.

Chapter 6

Issues Related to Measures of Expert Systems

6.1 Introduction

Even though ES metrics are urgently needed, there are several related issues that have to be addressed in the development of ES measures. These issues deal with the different aspects of ES metrics and help the proper design of the metrics. In this chapter, we will discuss and present (1) the general criteria for the definitions of ES metrics, which can also be used to evaluate the existing metrics; (2) the strategies used in the formulation of ES measures, which can lead us to an effective design of the metrics; (3) the formal mechanism for the formal description of ES characteristics, which are the basis of measures; (4) the current work on the ES measures, which keeps us informed of the research status of this subject.

6.2 General Criteria

For ES metrics, It is necessary to have some criteria regarding their formulation. We propose the following general criteria in this thesis.

1. **Meaningful** This requires that the measure of a metric should at least conform to human intuition. For example, the rule bases which are seemingly more complex should also be declared as more complex by the defined metrics. The bottom line is that the metric measuring results should confirm to the observation.

2. **Reasonable** The range of the metric measuring results should not be too wide, so as to be useful and credible. For example, a result ranging from zero to billion would usually give an impression that the measures are not well defined.

3. **Reliable** The measures should be as reliable as possible. This means that the effect on the measures caused by some trivial factors or arrangements of rule bases should be as little as possible. For example, some " unimportant words" in the rules or the meaningless re-organization of rule bases should not affect the measurements.

4. **Cost-effective** The efforts for obtaining a measuring result should not be high, that is, the measure should be practical, otherwise it would be useless. This is extremely important in the measures of rule bases, since the complex interactions among the rules usually produce enormous (exponential) paths. So, it would not be feasible for a measure to follow all the actual execution paths

in the rule bases.

These criteria form the basic principles that are followed in our design of ES metrics.

6.3 Strategies

Since ES, more precisely the rule bases in them contain both (1) the inherent features which are special to the particular ES; and (2) the common features that are similar to conventional software, it is clear that an effective way to formulate the measures is to identify the above two portions, and

- develop new approaches to measure the inherent or particular facets of ES, such as the measure of *dependency* among the rules,
- use or adapt existing and well-established measuring techniques applied to conventional software for measuring conventional portions of ES, such as the *size* measure.

Figure 12 shows such a concept that will guide our metric design for ES.

In general, the development methods, measuring techniques, and tools for conventional software cannot be directly applied to ES. Due to their specific nature, measurements on them need to be extended or created both in their semantic meanings and syntactic construction.

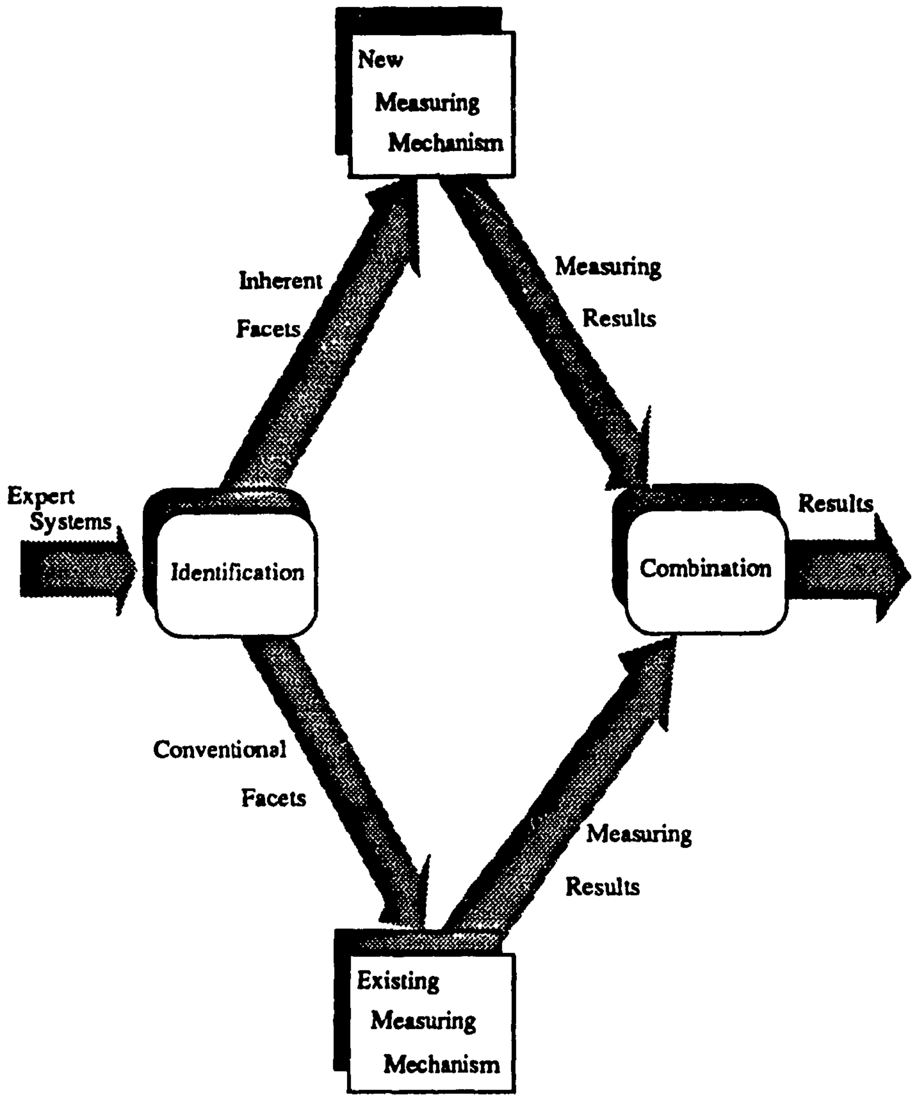


Figure 12: Strategy for the Formulation of Expert System Measures

6.4 Formal Description of Rule Bases

In order to formally measure the rule bases, first we must be able to formally describe the characteristics of rule bases in a formal way.

In Chapter 2, a formal language was proposed to describe the static and syntactic structures of rule bases, but we also need a formal mechanism to describe the inter-relation among the rules, that is, the *dependencies* which form the search space of an ES. Graphs have been proven to be an effective tool in the structural analysis of different systems. Their property of connectivity makes the graphs more attractive for the studies of rule bases [Nazareth and Kennedy, 1990]. Hence, a rule dependency AND/OR digraph is proposed and used to present the rule bases and their "causality" relations among rules in our analysis.

Given a rule base, a corresponding AND/OR digraph $< B, A, O, L, I, E >$ will be constructed, where B is the set of starting nodes at which there is no edge incident, A represents the set of AND nodes from which there are several edges that are jointly incident at another node whose firing depends on these nodes, O the set of OR nodes from which there are several edges that are incident at another node whose firing depends on only one of these nodes, L indicates the set of terminal nodes from which there is no edge emanating, and I labels the isolated nodes to which there is no edge relating. Each node is a rule in R and its formal description is shown in Table 1. E is the set of directed edges connecting the nodes, which represents the "causality" relation.

The following is the formal description of our AND/OR digraph and the notion will be used in our discussion:

r = a rule

$cons(r)$ = the set of patterns contained in RHS of a rule r.

$ante(r)$ = the set of patterns contained in the LHS part of a rule r.

ψ = a substitution $\{t_1/v_1, t_2/v_2, \ldots.., t_n/v_n\}$. t's are terms, v's are distinct variables.

$|S|$ = the cardinality of set S.

$\prod(R)$ - the set of all subsets of R.

Then, we have:

$$E = \{(r_i\ r_j) | r_i, r_j \in R \wedge \exists\psi \cdot (cons(r_i)\psi \cap ante(r_j)\psi \neq \emptyset)\}$$

$$A = \{r_i | \exists r_j \in R \wedge \exists S \subseteq R \cdot (|S| \geq 2 \wedge enf(r_j\ S) \wedge r_i \in S)\}$$

$$O = \{r_i | r_i \in R\ \wedge \exists r_j \in R \cdot enf(r_j\ \{r_i\}))\}$$

$$B = \{r_i | r_i \in R \wedge \neg\exists r_j \in R \cdot (r_j\ r_i) \in E \wedge \exists r_k \in R \cdot\ (r_i\ r_k) \in E\}$$

$$L = \{r_i | r_i \in R \wedge \neg\exists r_j \in R \cdot (r_i\ r_j) \in E \wedge \exists r_k \in R \cdot (r_k\ r_i) \in E\}$$

$$I = \{r_i | r_i \in R \wedge \forall r_j \in R \cdot (\neg\exists\psi \cdot (cons(r_j)\psi \cap ante(r_i)\psi \neq \emptyset \vee$$
$$cons(r_i)\psi \cap ante(r_j)\psi \neq \emptyset))\},$$

where the "enable firing" predicate (enf) is given by:

enf: $R \times \prod(R) \longrightarrow \{true,\ false\}$

$$enf(r_i\ S) = \begin{cases} true & \text{if } \forall r \in S\ \exists\psi \cdot (cons(r)\psi \cap ante(r_i)\psi \neq \emptyset) \wedge (firing(S) \\ & \rightarrow firable(r_i)) \wedge \neg\exists S' \subset S \cdot (firing(S') \rightarrow firable(r_i)) \\ false & \text{otherwise} \end{cases}$$

firing: $\prod(R) \longrightarrow \{true,\ false\}$

$$firing(S) = \begin{cases} true & \text{if all rules in S are fired} \\ false & \text{otherwise} \end{cases}$$

firable: $R \longrightarrow \{true,\ false\}$

$$firable(r_i) = \begin{cases} true & \text{if all antecedents in } r_i \text{ are satisfied and } r_i \text{ is waiting to be} \\ & \text{executed} \\ false & \text{otherwise} \end{cases}$$

Definition: the *fan-in* number ($f_i(r_i)$) of a node r_i is defined as number of edges terminating at the node r_i, or the number of rules r_j that meet the following condition:

$$cons(r_j)\psi \wedge ante(r_i)\psi \neq \emptyset$$

That is the number of rules upon which the firing of r_i depends.

Definition: the *fan-out* number $f_o(r_i)$ of a node r_i is defined as the number of edges emanating from r_i, or the number of rules r_j that meet the following condition:

$$ante(r_j)\psi \wedge cons(r_i)\psi \neq \emptyset$$

This is the number of rules whose firing depends on r_i.

For terminal or isolated nodes $r_i \in L \cup I$, $f_o(r_i) = 0$.

From the above, we can also see that it is possible that $A \cap O \neq \emptyset$. This means that some nodes may act as both AND and OR nodes. Also possibly

$$B \cap A \neq \emptyset$$

$$B \cap O \neq \emptyset$$

So a starting node could also be either an AND or an OR node, or both.

Some other properties are:

$$A \cap L = \emptyset$$

$$A \cap I = \emptyset$$

$$O \cap L = \emptyset$$

$$O \cap I = \emptyset$$

$$B \cap I = \emptyset$$

$$B \cap L = \emptyset$$

$$I \cap L = \emptyset$$

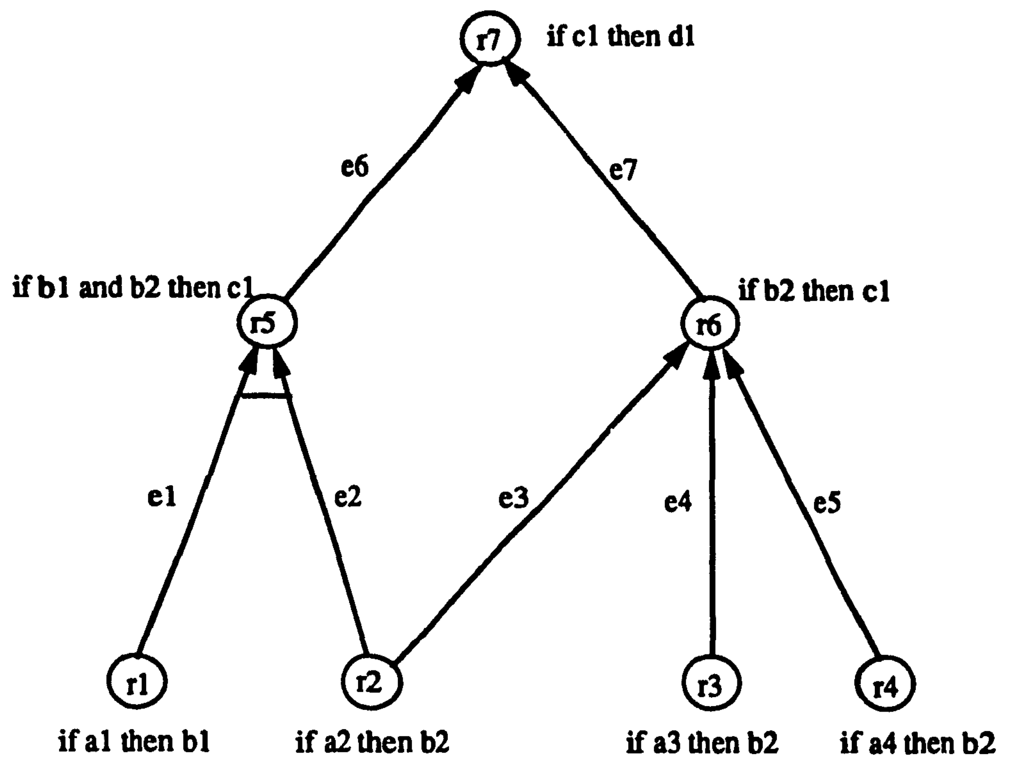
In our digraph, AND nodes are marked by placing an arc on the corresponding edges; while "OR" nodes remain unmarked.

The AND/OR digraph characterizes the individual rules and their inter-relation. It also illustrates the search path formed by rules for problem solving.

Example 6.1 Figure 13 shows an AND/OR representation and its components for an example rule base.

6.5 Current Work

Already, some work has started. For example, Plant presented a rigorous methodology that used a set of formal specifications toward the implementation of knowledge-based systems [Plant, 1991b], and the quality improvement through this methodology was shown by the effect of this methodology on some quality factors such as correctness, reliability, efficiency, integrity, testability, usability and maintainability, which can be measured by the metrics [Plant, 1991a]. Kiper attempted to extend McCabe's cyclomatic metric to measure the basic search paths contained in rule bases [Kiper, 1992]. Buchanan suggested the complexity metric of solution space to be the measurement of the width and depth of the rule bases [Buchanan, 1987]. Mehrotra defined and used a "distance metric" to group the rule bases so as to increase the expert system's comprehensibility, maintainability and reliability [Mehrotra, 1991]; Preece suggested that attention should also be paid to the differences in data-to-rule ratio [Preece, 1990].



B={ r1, r2, r3, r4 }

A={ r1, r2 }

O={ r2, r3, r4, r5, r6 }

L={ r7 }

I= ϕ

E={ e1, e2, e3, e4, e5, e6, e7 }

Figure 13: An AND/OR Representation of A Rule Base

In the VALID project, several issues concerning the quality control and evaluation of knowledge-based systems were addressed [Cardeñosa, 1995]. And in the UK, the Gateway project which aimed at constructing a coherent set of metrics for knowledge-based systems was developed, and some preliminary findings about the development of knowledge-based systems by applying Gateway were shown [Behrendt et al., 1991].

To date, several measurements have been suggested for rule bases, such as:

- object volume [Kaisler, 1986],
- number of rules [Suen et al., 1990],
- number of antecedents and consequents (items) in a rule [Kaisler, 1986],
- breadth of the knowledge base [Suen et al., 1990],
- depth of the search space [Suen et al., 1990],
- complexity of individual rules [Miller, 1990],
- vocabulary of the knowledge base [Buchanan, 1987],
- dynamic metrics, for example, the number of rules executed per cycle, or average time required to solve the problem [Kaisler, 1986].

The problem lies in the validity and organization of these measurements; some of them may be highly inter-related and measure the same factors of the rule bases. Also, no sufficient evidence and experiments have been shown to support these measures, even though some are intuitive.

Chapter 7

Quality Metrics for Expert Systems

7.1 Introduction

As indicated in Chapter 5, quality metrics are the basic and primitive elements for assessing the quality model. They provide the quantitative information about the attributes of the objects to be measured. The quality metrics of ES are the measures of ES characteristics. So, the characteristics of ES are the targets of the measures, based on which different metrics can be defined and formulated. In this thesis, three important, quality-related and static characteristics of ES are examined, they are: *size, complexity* and *search space.* The *size* and *search space* will be studied in this chapter, and the *complexity* in the next chapter. The corresponding metrics are also presented.

7.2 Size Measures

From Chapter 3, we know that the common metric used in conventional software engineering for measuring the size of the software products is the LOC, the *lines of code*. This kind of measure is simple and easy to calculate, however, its simplicity may also bring many drawbacks as discussed before. The counterpart of LOC in an ES is the number of rules (NR). NR has as many drawbacks as LOC. However, since it is easy to count, it is still used. But caution must be taken when inferring other attributes from it. Based on the LOC, we define NR as:

$$NR: (\mathcal{E}, A_1, P_1) \quad \longrightarrow \quad (\mathcal{N}, T_1, R_1)$$

where

$\mathcal{E}$: rule bases

A_1: sizes

P_1: $(x, y) \in P_1$ if size of rule base x is greater than or equal to that of rule base y.

$\mathcal{N}$: natural numbers

T_1: values

R_1: $(x, y) \in P_1$ if $x \geq y$

$NR(\mathcal{E})$ = number of rules in a rule base excluding the control rules.

Another measure of the size of an ES is the count of the total patterns (NAC)

contained in the *LHS* and *RHS* parts of all the rules in an ES, which is based on the following direct metric *num*, *the number of patterns* in each rule:

$$num : (\mathcal{E}, A_1, P_1) \longrightarrow (\mathcal{N}, T_1, R_1)$$

where

$\mathcal{E}$: rules

A_1: sizes

P_1: $(x, y) \in P_1$ if size of rule x is greater than or equal to that of rule y

$\mathcal{N}$: natural numbers

T_1: values

R_1: $(x, y) \in P_1$ if $x \geq y$

$num(\mathcal{E})$ = total number of patterns contained in the *LHS* and *RHS* parts of a rule.

Then,

$$NAC\text{: Rule Bases} \longrightarrow \text{Numerical Numbers}$$

$$NAC(R) = \sum num(r_i)$$

where:

r_i = some rule in a rule base

NAC is a second layer metric, and it seems to consider more details of rule bases, hence, to be more precise than NR. But our results reveal that NR and NAC have the same performance for assessing or predicting the ES quality. Also, NAC is obtained at the expense of extra calculation of the number of patterns in each rule. Hence most people are more willing to choose NR instead of NAC.

7.3 Search Space

An important and distinctive feature of ES is their *search space* formed by the rules and the dependencies among them. Because of its critical role, the search space has become the focus of many studies. The challenge is how to formally and precisely represent the search space and how to capture its essential components. In Chapter 6, an AND/OR digraph was proposed for the formal description of the search space. Based on this AND/OR digraph, we can then formally describe some concepts and measures related to the search space.

In our discussion, the following notations will still be used:

$r =$ some rule.

$R =$ some rule base.

$B =$ the set of the starting nodes in the AND/OR digraph of R.

$cons(r) =$ the set of patterns contained in the RHS part of a rule r.

$ante(r) =$ the set of patterns contained in the LHS part of a rule r.

ψ = a substitution $\{t_1/v_1, t_2/v_2, ..., t_n/v_n\}$, t's are terms, v's are distinct variables.

$|S|$ = the cardinality of set S.

The important characteristics to be considered are then the attributes associated with the search paths such as the ***depth*** of search paths and ***breadth*** of search paths. To describe them, we first introduce a concept called *AND-GROUP* with the following definition:

AND-GROUP: a minimum set of rules which jointly make some rule firable, that is, all the conditions in the *LHS* part of some rule will be satisfied by the consequents of this set of rules. Formally, it can be represented as follows:

$$AND-GROUP_i = \{\{r_{i1}\ r_{i2}\ ......r_{im}\} | \exists r_j \in R \cdot\ enf(r_j,\ \{r_{i1}\ r_{i2}\ ......r_{im}\})$$

In R, there may exist several *AND-GROUPs*, each of which makes some rule firable, and the intersections among these *AND-GROUP* may not be empty. The difference between an *AND-GROUP* and predicate enf is that enf is an assertion that specifies both a set of rules and the rule that is made firable by this set of rules, while *AND-GROUP* is a set that contains only that set of rules that makes some rule firable.

Now, we can describe the search path as:

Search Path: a path Q is a set of $AND-GROUP$s w_i in R, that is:

$$Q = \{w_1,\ w_2,\ ......\ w_n\}$$

where:

1° $w_1 \subseteq B$ is a subset of the starting nodes (rules).

2° w_n is the set of some goal nodes (rules).

3° $\forall i \cdot (1 < i \leq n \rightarrow \forall u \in w_i \, \exists v \subseteq w_{i-1} \cdot enf(u, v))$

The above shows that:

(a) a path begins at some starting nodes and ends in some goal nodes.

(b) each rule in the path, except for the *starting rules*, is enabled by some *sub-AND-GROUP* and only by this *sub-AND-GROUP*.

(c) each *AND-GROUP* will enable some rule(s) firable in some *AND-GROUPs* in the path.

Based on the above description of search paths, all the paths in R can be identified. Suppose the there are n paths and $Q_i (1 \leq i \leq n)$ is some path, then the measures on the paths are:

Average Depth of Search Space (ADSS): *ADSS* denotes the average depth of all the search paths in an ES. To measure it, first we need to measure the length for each path Q_i. Suppose (DP) represents such measure on the length, then it can be described as:

$$DP: \ (\mathcal{E}, A_1, P_1) \quad \longrightarrow \quad (\mathcal{N}, T_1, R_1)$$

where

$\mathcal{E}$: search paths

A_1: lengths

P_1: $(x,\ y) \in P_1$ if length of path x is greater than or equal to that of path y

$\mathcal{N}$: natural numbers

T_1: values

R_1: $(x,\ y) \in P_1$ if $x \geq y$

$DP(\mathcal{E}) = |\mathcal{E}| - 1$, which equals the cardinality of each path minus one.

Then:

$$ADSS : \quad \text{Search Paths} \longrightarrow \text{Numerical Numbers}$$

$$ADSS(R) = \frac{\sum_{i=1}^{i=n} DP(Q_i)}{n}$$

Average Breadth of Search Space (ABSS): *ABSS* denotes the average fan-in number of nodes in different paths. So, its measure is based on the measure (AVB) of average fan-in number of nodes in each path, that is,

$$AVB : \quad \text{Search Paths} \longrightarrow \text{Numerical Numbers}$$

$$AVB(Q_i) = \frac{\sum_{i=1}^{i=|R_i|} f_i(r_i)}{|R_i| - |B_i|}$$

where

$Q_i =$ some path

$R_i =$the set of nodes in Q_i

$r_i \in R_i$

$f_i(r_i) =$ *fan-in of* r_i

B_i=the set of starting nodes in Q_i

Then:

$$ABSS: \text{ Search Paths } \longrightarrow \text{ Numerical Numbers}$$

$$ABSS(R) = \frac{\sum_{i=1}^{i=n} AVB(Q_i)}{n}$$

One approximate measure of $ABSS$ is by calculating the average fan-in number of nodes in the entire AND/OR digraph instead of each path, that is:

$$ABSS: \text{ AND/OR Digraphs } \longrightarrow \text{ Numerical Numbers}$$

$$ABSS(R) = \frac{\sum_{i=1}^{i=|R|} f_i(r_i)}{|R| - |B|}$$

where

$r_i \in R$

$f_i(r_i) = \textit{fan-in of } r_i$

B =the set of starting nodes in R

It is obviously that this approximate measure is less expensive, but can still achieve quite precise results.

Buchanan's Complexity (BC), which is a measure of the total size of the abstract search space. It can be described as:

$$BC: \text{ Search Paths} \longrightarrow \text{Numerical Numbers}$$

$$BC(R) = ABSS^{ADSS}$$

So, BC is actual based on the measures of $ADSS$ and $ABSS$.

Chapter 8

Complexity Measures of Rule Bases

8.1 Introduction

Complexity has been recognized as a crucial characteristic in a software system's development and quality, as a consequence, many conventional software complexity measures have been introduced in terms of different requirements, and many tools have been developed to use software complexity measures to analyze the software in order to reduce its cost and improve the quality [Zuse, 1991]. Basically, software complexity measures attempt to capture major internal and static attributes of the software products, and summarize these attributes by a numerical value. This value is expected to have the property of an indicator of some quality factors, such as *maintainability, testability, reliability.* A high value of complexity measure is supposed to be indicative of low maintainability, testability and reliability, etc.

Even though there exist many complexity metrics for conventional software, such measures on ES are lacking. By the nature of ES, complexity metrics for ES, like the complexity metrics for conventional software, are also critical and important. In the following sections, we will investigate complexity measures of rule bases, and propose our complexity metrics — RC and ERC for measuring the complexity.

8.2 Principles and Definition

As there is no standard definition and no comparative studies done on product complexity, various conventional complexity metrics have been proposed and designed for different applications in software engineering. There exist a number of complexity metrics in software engineering. The basic considerations are: What aspects (factors) should be measured? How to measure them? And for what purpose?

It is generally accepted that the psychological complexities[1] (as distinct from computational complexities which evaluate the theoretical cost of software with respect to efficiency, execution time, etc) are a measure of the degree of difficulty in comprehending and working with a piece of software [Li and Cheung, 1987, Harrison et al., 1982]. Many quality factors like maintainability (including software understandability, modifiability), reliability and testability, are affected by complexity.

The basic principles underlying the measurements used in software engineering could also be applied to rule bases. The key issue in complexity measurement is the measure of the difficulty in understanding the products. But the meanings implied

[1]In this thesis, the term "complexity" is used synonymously with "psychological complexity" if no particular specification is given.

by rule bases and conventional software are different. ES are applied to areas in which expertise is needed. A rule base is used only as a means of encapsulating such expertise, the control information for the execution (firing) of the rules in the rule base are not contained in the rules themselves, i.e., the rules are not for the control of the system, they are the representation of knowledge for solving problems in the application domains, whereas the statements in the conventional software are the control codes that decide the system's execution. In this thesis, the complexity of rule bases is therefore defined as:

"The measure of the degree of difficulty in understanding and managing the knowledge represented by the rule base."

This definition involves *external* and *internal* aspects which will be discussed in the next section.

8.3 Model of Complexity

Similar to other characteristics, the essence of the complexity of a rule base may be seen from two perspectives. One is due to the *external component* which influences the complexity and can only be measured with regard to its environment. Another is the *internal component* which reflects the complexity and can be measured in terms of the rule base itself. This concept model can be characterized by Figure 14. The *external component* consists of:

1. Experts. They provide the knowledge for solving problems, thus contributing to the complexity of the knowledge. It could also be shown that different experts

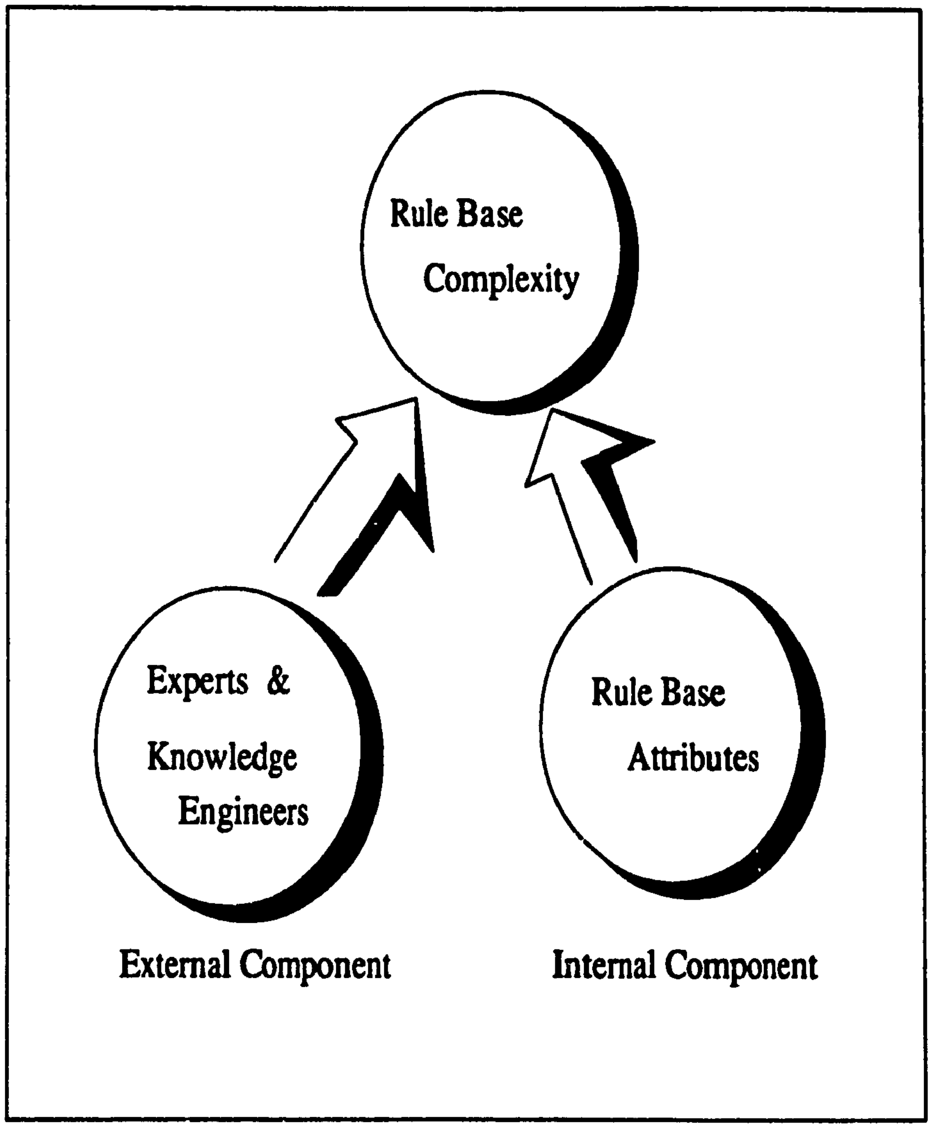


Figure 14: Complexity Model of Rule Bases

or expert groups may produce the knowledge with different complexities for the same problem, because of differences in their expertise, personalities and so on.

2. Knowledge Engineers. Since the knowledge engineers are responsible for knowledge acquisition and representation, it is obvious that their attributes such as experience, natural ability, and motivation, also contribute to the formulation of the complexity. In addition, both the environment in which they work and the tools they use influence the complexity.

Besides, the "difficulty" in understanding a rule base will also be different for different users who use the ES, because of individual differences in them, such as their backgrounds.

While it seems that the above *external component* is, by its nature, situation-dependent, most difficult to measure, and there is no agreed definitions, actually our focus is on the measurement of the *internal component.*

The *internal component* may be regarded as the attributes of the structure which captures the knowledge. In rule-based ES, a rule base serves this purpose, and its attributes characterize the *internal component.* There exist different kinds of rule base description languages, which also affect the complexity. But for solving problems, the rule base attributes are mainly determined by the knowledge itself. So in our analysis, we will convert different rule bases written in different description languages into those expressed by the generic form presented in Chapter 4, and concentrate on measuring the characteristics (attributes) that reflect the complexity of rule bases.

8.4 Hybrid Complexity Metric — RC

From the measuring experience in software engineering, we know that hybrid metrics can usually reflect the target software more subjectively. Hence, the formulation of *RC* will take into account the three important characteristics of rule bases: (1) contents of the matching patterns, (2) search space, and (3) size. These characteristics may correspond to the *data flow, control flow* and *size* characteristics of conventional software as discussed in Chapter 3, and our measure (*RC*) is designed to reflect the influences of the matching patterns, search space and size on the rule base complexity.

8.4.1 Contents

Miller identified 11 complexity features associated with each rule [Miller, 1990]. They reflect the content of each rule and are related to possible syntactic errors; that is, the more the measurement, the more the opportunity for making errors. For different implementations of the same problem, the measurements could be quite different, that is, they are sensitive to the implementation. For example, the feature "number of attributes" could be quite different for various implementations if the attribute names are allowed to be absent from the rules because of context-dependence. In our measurement, rules are described as a composition of the patterns which are expressed as $< qualifier, value >$[2] (Table 1), and the measurement is focused on the *connectors* which result in the interactions among the rules, hence, the complexity of the rule bases. It implies two cases:

[2]The concept used here is similar to that in the Exsys shell. It could increase the reliability of the measurement, since rule bases, like programs, can be trivially rewritten to give different content.

1. Pattern in the antecedent of a rule, which matches the patterns in the consequent of other rules.

2. Pattern in the consequent of a rule, which matches the patterns in the antecedent of other rules.

The above patterns, called *connectors*, form the inter-relations among the rules; these inter-relations correspond to the arcs in the AND/OR digraph.

The variables in the connectors form and produce various matching patterns, and therefore indicate the dynamically and potentially different chainings among the rules. So rule bases with different designed variables will be quite different in their complexities. Because of this reason, variables will be considered as a separate component affecting the complexity of rule bases. Consider a typical example of two rule bases, one with several variables and the other without any variable. If their sizes and other measurements are the same, it is obvious that the latter rule base is less complex than the former one.

Let M denote the set of connectors, and the following components in M: ν the set of variables; τ the set of qualifiers; μ the set of values, then the content measurement of a rule base will be a function of M, that is $f(M)$ or $f(\nu, \tau, \mu)$. One way to define f is to let it be the volume or amount measurement of ν, τ and μ. Then from this point of view, this measurement is similar to the one used in software engineering, that is, the volume measurement method developed in software engineering could be applied to this conventional portion. Halstead's volume metric, one of the widely accepted measurements in software engineering, is adopted here, but the scopes of operators and operands defined for conventional software need to be extended, which are composed of:

- variables
- qualifiers
- values, including
  - constants
  - basic arithmetic operators: $+$, $-$, $*$, $/$, $(\ )$
  - relational and Boolean operators: $>$, $>=$, $<$, $<=$, $=$
  - logic operators: *and, or, not*
  - function or procedure calls

By assigning different weights to the above items to reflect their different "importance" and introducing Halstead's volume metric, we have the *content measurement* π given by:

$$\pi : \quad \text{Rule Bases} \quad \longrightarrow \quad \text{Numerical Numbers}$$

$$\pi = N\, log\, (1 + n) \tag{1}$$

which is based on the following measures:

$$n_1, n_2, n_3, N_1, N_2, N_3 : \ (\mathcal{E}, \{A_1, A_2, A_3\}, \{P_1, P_2, P_3\}) \longrightarrow (\mathcal{N}, T_1, R_1)$$

where

$\mathcal{E}$: rule bases

A_1: volume of variables in a rule base

A_2: volume of qualifiers in a rule base

A_3: volume of values in a rule base

P_1: $(x, y) \in P_1$ if the volume of variables in rule base x is greater than or equal to that in rule base y.

P_2: $(x, y) \in P_1$ if the volume of qualifiers in rule base x is greater than or equal to that in rule base y.

P_3: $(x, y) \in P_1$ if the volume of values in rule base x is greater than or equal to that in rule base y.

$\mathcal{N}$: natural numbers

T_1: values

R_1: $(x, y) \in R_1$ if $x \geq y$

$n_1(\mathcal{E})$= number of distinct variables in a rule base

$n_2(\mathcal{E})$= number of distinct qualifiers in a rule base;

$n_3(\mathcal{E})$= number of distinctive values in a rule base;

$N_1(\mathcal{E})$= number of total variables in a rule base;

$N_2(\mathcal{E})$= number of total qualifiers in a rule base;

$N_3(\mathcal{E})$= number of total values in a rule base;

N and n are the second layer metrics defined as:

$$N,\ n: \text{ Rule Bases } \longrightarrow \text{ Numerical Numbers}$$

$$N = w1 \times N_1 + w2 \times N_2 + w3 \times N_3$$

$$n = w1 \times n_1 + w2 \times n_2 + w3 \times n_3$$

$$w_j = \text{weights. } (1 \leq j \leq 3)$$

It can be seen that the above calculations measure the content of each node of the AND/OR digraph. The next measurement will focus on the search space, which corresponds to the connections of AND/OR digraph.

8.4.2 Connectivity

It is the interaction or connection among rules that mainly makes the rule bases more complex. That is, the connectivity is the main factor which contributes to the complexity of the rule bases. Figure 15 summarizes the basic types of connections among the nodes which will be used to form the whole dependency digraph of the rule bases. Figure 15 (a) shows that $r_1, r_2, \ldots., r_n$ form one *AND-GROUP* with regard to r_i; Figure 15 (b) shows the "OR" relationship between the $AND-GROUP$s $G_1, G_2, \ldots\ldots, G_k$. When $|G_l| = 1$, it represents only one node (rule). Figure 15 (c) indicates the *"fan-out"* from rule r_i to rules $r_1, r_2, \ldots\ldots, r_n$. That is: $cons(r_i)\psi \cap ante(r_l)\psi \neq \emptyset$

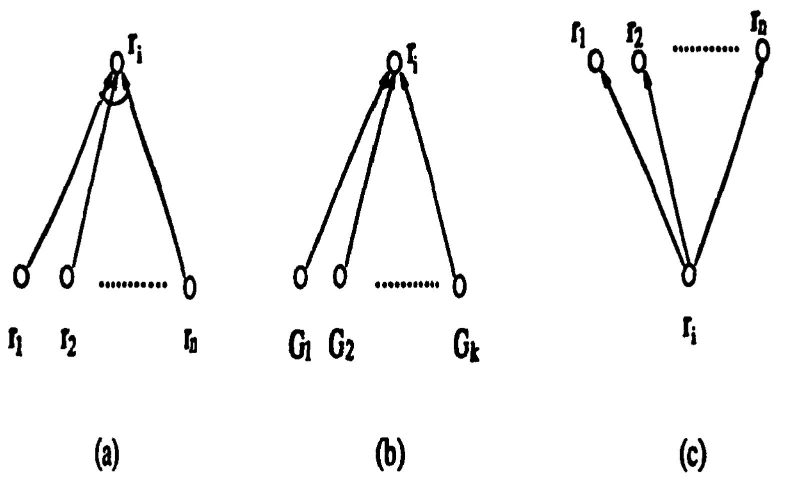


Figure 15: Basic Structures in The Rule Dependency Digraph

for l=1, 2, ..., n. The connectivity measurement, $\rho(r_i)$, is then designed to measure the "influence" of the other nodes upon node r_i.

For any node r_i, $\rho(r_i)$ is affected by two factors:

1. The depth from B to r_i

   As it increases, the possible connections from other nodes to r_i increase, which could be either direct or indirect, and the possible influence of other nodes upon r_i increases as well.

2. The width of the path leading to node r_i

   As it increases, it also increases the connections of other nodes to r_i, hence the influence of other nodes upon node r_i.

These considerations lead to the following approach in the formulation of $\rho(r_i)$:

$$\rho : (\mathcal{E}, A_1, P_1) \longrightarrow (\mathcal{N}, T_1, R_1)$$

where

$\mathcal{E}$: AND/OR digraphs

A_1: connectivity

P_1: $(x, y) \in P_1$ if node x has more connections with other nodes than node y, or the same connections as node y.

$\mathcal{N}$: natural numbers

T_1: values

R_1: $(x, y) \in R_1$ if $x \geq y$

$\rho(\mathcal{E})$ = as defined below

1. If r_i is an isolated node, that is $r_i \in I$, then $\rho(r_i) = 0$.
2. If r_i is a starting node, that is $r_i \in B$, then the measurement will be: $\rho(r_i) = 1$.
3. If r_i is not a starting node, that is $r_i \in A \cup O \cup L - B$, suppose there exist k $AND-GROUP$s G_1, G_2, ..., G_k w.r.t. r_i (like figure 3(b)), and each AND-$GROUP$ acts as an "OR" node, that is, firing all rules in one $AND-GROUP$ G_l $(1 \leq l \leq k)$ will make node (rule) r_i become "firable". Then there are two steps in calculating $\rho(r_i)$:

   First, obtain the maximal ρ value of each $AND-GROUP$ G_l $(1 \leq l \leq k)$: $\rho(G_l)$, which contains nodes r_{l1}, r_{l2}, ..., r_{ln}, $(n \geq 1)$ w.r.t. r_i.

$$\rho(G_l) = \max_{r_{lj} \in G_l} \rho(r_{lj})$$

where:

$$1 \leq j \leq n$$

The second step is to calculate the connectivity for each node r_i:

- If r_i is an internal node, that is $r_i \in A \cup O - B - L$, then

$$\rho(r_i) = \sum_{l=1}^{k} \rho(G_l) + f_o(r_i) - 1$$

- If r_i is a terminal node, that is $r_i \in L$, then

$$\rho(r_i) = \sum_{l=1}^{k} \rho(G_l)$$

Since for any internal node, there will be normally at least one edge connecting it with other nodes, which doesn't contribute to the comparison of the connectivity of rule bases. So the -1 term is added in the calculation of $\rho(r_i)$ to discard this normal situation, and the threshold of 1 is set for the $\rho(r_i)$ value, *i.e.* $\rho(r_i) > 1$, to reflect the degree of influence caused by connections which exceed the "normal value"[3]. Such a set of nodes is defined as the ***influenced node set*** P, that is:

$$P = \{r_i | r_i \in R \wedge \rho(r_i) > 1\}$$

8.4.3 Size

This measurement captures the size of the digraph which reflects both the size of the rule bases and the amount of edges of an AND/OR graph. It can be simply characterized as:

[3]We regard the number of edges connecting a node as normal if it is less than or equal to two (one edge emanates from the node, another one incidents at the node.)

$$v: \text{ AND/OR digraphs } \longrightarrow \text{ Numerical Numbers}$$

$$v = nn + ne. \tag{2}$$

where, nn and ne are defined as follows:

$$nn, ne: \ (\mathcal{E}, \{A_1, A_2\}, \{P_1, P_2\}) \longrightarrow (\mathcal{N}, T_1, R_1)$$

where

$\mathcal{E}$: AND/OR digraphs

A_1: volume of nodes in an AND/OR digraph

A_2: volume of edges in an AND/OR digraph

P_1: $(x, y) \in P_1$ if volume of nodes in AND/OR digraph x is greater than or equal to that of AND/OR digraph y.

P_2: $(x, y) \in P_2$ if volume of edges in AND/OR digraph x is greater than or equal to that of AND/OR digraph y.

$\mathcal{N}$: natural numbers

T_1: values

R_1: $(x, y) \in R_1$ if $x \geq y$

$nn(\mathcal{E})$ = number of nodes in an AND/OR digraph

$ne(\mathcal{E})$ = number of edges in an AND/OR digraph

It is obvious that when a rule base contains circularity, redundant rules or other anomalies, the value of v will increase. Also the "key" nodes which have more connections with other nodes contribute more to the value of v.

8.4.4 Calculation of Complexity

Having identified the v, π, ρ factors that constitute the complexity, and their corresponding measurements; the complexity metric (RC) of rule bases can now be defined as a function (g) of these factors. That is:

$$RC: \text{ Rule Bases } \longrightarrow \text{ Numerical Numbers}$$

$$RC(rule\ bases) = g(\pi, \rho, v) \tag{3}$$

From the formulations of π, ρ and v, we know that as these values increase, the RC will also increase. To reflect this characteristic, a formula in the following form is applied:

$$1.0 - \frac{1.0}{1.0 + \lambda(\omega)} \tag{4}$$

where,

$\omega = \pi, \rho, v$

$\lambda =$ a function of ω

λ is designed to reflect the influence of π, ρ and v on RC, and its value is designed to be greater or equal to 0, which makes the overall value of expression (4) greater than 0, but less than 1.

Based on the expression (4), the formula (3) is further refined as the follows empirically:

$$RC(R) = [W_1 * (1.0 - \frac{k_1}{k_1 + \sum_{i=1}^{|P|} \rho(r_i)}) + W_2 * (1.0 - \frac{k_2}{k_2 + \pi}) + W_3 * (1.0 - \frac{1.0 + k_3}{k_3 + v})]^n \quad (5)$$

where

$$\begin{aligned} R &= some\,rule\,base \\ P &= influenced\,node\,set \\ r_i &\in P \\ k_1,\ k_2,\ k_3 &= scale\,factors \\ n &= constant \\ W_1,\ W_2,\ W_3 &= weights \end{aligned}$$

The weights have been set in such a way that $W_1 + W_2 + W_3 = 1$ and they indicate the different contributions of the three components relating to ρ, π, and v. One possible setting[4] for the above parameters is:

$$W_1 = \frac{1}{3}$$
$$W_2 = \frac{1}{3}$$
$$W_3 = \frac{1}{3}$$

[4]It is based on our empirical data.

$$k_1 = 100$$

$$k_2 = 100$$

$$k_3 = 50$$

$$n = 2$$

The other rationales for expressing RC as formula (5) are summarized as follows:

- The complexity may be of a higher order than linear in terms of the three components.
- As ρ, π and v increase, the value of RC expression (formula (5)) also increases.
- Reasonable results can be achieved when formula (5) is applied to some sample rule bases.
- Only empirical studies and experiments may be used to deduce this kind of measurements.

Some simple basic structures among three rules and their corresponding RC values are shown in Figure 16 to illustrate the effectiveness of our measurement. The measuring results show that rule bases No. 1 and No. 3 have the highest complexity because of their relatively more complex connectors even though all of these four structures have almost the same degree of connections.

The RC metric (formula (4)) has some interesting properties listed below (suppose R represents a rule base):

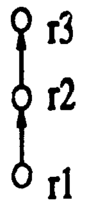
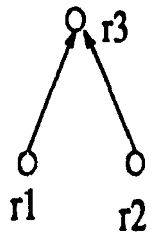
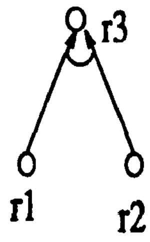
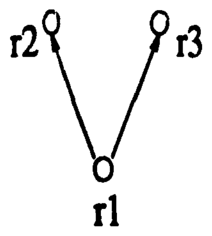
No.	Sample Rules	Dependency Digraph	RC(R)
1	r1: $a \rightarrow b$ r2: $b \rightarrow c$ r3: $c \rightarrow d$		0.0019
2	r1: $a \rightarrow b$ r2: $c \rightarrow b$ r3: $b \rightarrow d$		0.0016
3	r1: $a \rightarrow b$ r2: $c \rightarrow d$ r3: $b,d \rightarrow e$		0.0019
4	r1: $a \rightarrow b$ r2: $b \rightarrow c$ r3: $b \rightarrow d$		0.0011

Figure 16: Measurements of Basic Structures by *RC*

(a) $0 \leq RC(R) < 1$.

(b) Adding a new node (rule) or edge (connection) to R will increase RC(R) except that the added node is isolated.

(c) For any subset $\Theta \subseteq R$, $RC(\Theta) \leq RC(R)$, the complexity of each sub-rule bases is less than or equal to that of the global rule bases.

(d) $RC(R) = 0$ denotes that R contains only isolated nodes in which each node acts as a separate subgraph.

(e) $0.0003 < RC(R) < 0.4445$ if R forms only one search path.

(f) $RC(R) > 0.0011$ if there are more than one path in R.

The first and second properties may be proved from the approach and formulas described above. The third one may be deduced from the second property. The last three properties are based on the following observations:

- For isolated nodes:

$$\pi = 0$$

$$v = 1$$

$$P = \emptyset$$

$$\sum_{r_i \in P} \rho(r_i) = 0$$

So:

$$RC(R) = 0$$

- For nodes that constitute only one search path:

$$\pi \geq 2$$

$$v \geq 3$$

$$P = \emptyset$$

So:

$$W_2 * (1.0 - \frac{k_2}{k_2 + \pi}) \geq W_2 * (1.0 - \frac{k_2}{k_2 + 2})$$
$$W_3 * (1.0 - \frac{1.0 + k_3}{k_3 + v}) \geq W_3 * (1.0 - \frac{1.0 + k_3}{k_3 + 3})$$
$$W_1 * (1.0 - \frac{k_1}{k_1 + \sum_{r_i \in P} \rho(r_i)}) = 0$$
$$[W_2 * (1.0 - \frac{k_2}{k_2 + 2}) + W_3 * (1.0 - \frac{1.0 + k_3}{k_3 + 3})]^n \leq RC(R) < (W_2 + W_3)^n$$

that is: (suppose the values of the parameters are set as above)

$$0.0003 < RC(R) < 0.4445$$

- For nodes that form more than one search path:

$$\pi \geq 3$$

$$v \geq 5$$

$$|P| \geq 0$$

So:

$$W_2 * (1.0 - \frac{k_2}{k_2 + \pi}) \geq W_2 * (1.0 - \frac{k_2}{k_2 + 3})$$

$$W_3 * (1.0 - \frac{1.0 + k_3}{k_3 + v}) \geq W_3 * (1.0 - \frac{1.0 + k_3}{k_3 + 5})$$

$$1.0 - \frac{k_1}{k_1 + \sum_{r_i \in P} \rho(r_i)} \geq 0$$

$$RC(R) \geq [W_2 * (1 - \frac{k_2}{k_2 + 3}) + W_3 * (1.0 - \frac{1.0 + k_3}{k_3 + 5})]^n,$$

that is:

$$RC(R) > 0.0011$$

8.4.5 Other Considerations

The dependency AND/OR digraph for a rule base may consist of several separate subgraphs that are independent of each other. That is, there is no edge connecting these subgraphs and within each subgraph, each node is connected to some other nodes by edges. Such a subgraph represents a subset (group) of the rule bases, which is not related to others and could be obtained by applying some grouping algorithms. These features occur in many ES, especially those which are related to "classification" problems, e. g. fault diagnosis, disease diagnosis, animal identification, *etc.*

Because of this "independence" feature, the complexity measurement could be applied to these subgraphs in turn to reflect the different complexities of the different separate sub-rule bases. And the global complexity of the rule bases can be the maximum value of the complexities of these sub-rule bases, that is:

$$RC(R) = max(RC(R_1),\ RC(R_2) \ldots\ldots, RC(R_k))$$

R = a rule base to be measured.

$$R_i \subseteq R \ (1 \leq i \leq k).$$

The *RC* measurement may also be applied to a deliberately chosen subset of a rule base, that is, applied to some rules which are chosen to form a subset of the whole rule bases for a certain purpose. This application could be used in many cases, such as the distributions of efforts to be put into the different parts of a rule base in terms of the measurements in the maintenance process. It is also obvious that the *RC* value for a modular rule base is lower than that of an unstructured rule base, which is intuitively correct, since a modular rule base reduces the inter-relation among its rules.

8.5 Information Content-Based Metric — *ERC*

The principles of information-based measures used for measuring conventional software can also be applied to ES. Obviously, for rule bases, the information is contained in the items of each pattern in rules, which can be the predicates, the function calls, the mathematical operators and operands, *etc.* Therefore, our proposed metric (*ERC*) is based on the empirical distribution of these items within a rule base.

Suppose there are n items in a rule base, then the probability p_i for each item s_i is:

$$p_i : \text{Rule Bases} \longrightarrow \text{Numerical Numbers}$$

$$p_i = \frac{f_i}{N}$$

where f_i and N are the following measures:

$$f_i : (\mathcal{E}, A_1, P_1) \longrightarrow (\mathcal{N}, T_1, R_1)$$

where

$\mathcal{E}$: items in a rule base

A_1: occurrence of items

P_1: $(x, y) \in P_1$ if item x occurs more or equally often than item y.

$\mathcal{N}$: natural numbers

T_1: values

R_1: $(x, y) \in R_1$ if $x \geq y$

$f_i(\mathcal{E})$ = number of occurrences of the i-th item

and N is defined as the second layer metric:

$$N : \text{Rule Bases} \longrightarrow \text{Numerical Numbers}$$

$$N = \sum_{i=1}^{i=n} f_i$$

that is, N is the total number of occurrences of all items in a rule base.

Thus, the entropy (ERC) of the rule base is

$$ERC : \text{Rule Bases} \longrightarrow \text{Numerical Numbers}$$

$$ERC = -\sum_{i=1}^{i=n} p_i \log p_i$$

ERC is then computed and used for different ES as another ES complexity measure in addition to *RC*. Ard the hypothesis used is that a rule base with a higher *entropy*, on the whole, is more complicated than another rule base with a lower *entropy*.

The evaluation results presented in Chapter 13 will show that the performance of *ERC* is inferior to *RC*. The purpose for defining *ERC* is to compare the different metrics that are defined based on variou., viewpoints.

Chapter 9

Examples

9.1 Introduction

In order to illustrate the concepts and measures presented in the previous chapters, especially our proposed *RC* metric, this chapter selects two sample rule bases as measurement examples. They are: (a) the *animal identification rule base* [?] which is often used as a typical example in most expert system textbooks, and (b) the subset of *fault diagnosis rule base* [Rushby and Crow, 1990] from a real industrial application. The former is written in natural language using *if - then* format and the latter is written in CLIPS. Simple comparisons and analysis of the measuring results based on our intuitions are also conducted. A complete evaluation of the measures will be given in Chapter 13.

9.2 Animal Identification Rule Base

This rule base is designed to identify different animals. The rule base contains fifteen rules as shown below:

```
r1:

    if (animal has hair) then (animal is mammal)


r2:

    if (animal gives milk) then (animal is mammal)


r3:

    if (animal has feathers) then (animal is bird)


r4:

    if (animal can fly)

       (animal lays eggs)

    then (animal is bird)


r5:

    if (animal eats meat)  then (animal is carnivore)


r6:

    if (animal has pointed-teeth)

       (animal has claws)

       (animal has forward-eyes)

    then (animal is carnivore)
```

```
r7:

    if (animal is mammal)

       (animal has hoofs)

    then (animal is ungulate)


r8:

    if (animal is mammal)

       (animal chews cud)

    then (animal is ungulate)


r9:

    if (animal is mammal)

       (animal is carnivore)

       (animal has tawny-colour)

       (animal has dark-spots)

    then (animal is cheetah)


r10:

    if (animal is mammal)

       (animal is carnivore)

       (animal has tawny-colour)

       (animal has black-stripes)

    then (animal is tiger)
```

```
r11:
    if (animal is ungulate)
       (animal has long-neck)
       (animal has long-legs)
       (animal has dark-spots)
    then (animal is giraffe)

r12:
    if (animal is ungulate)
       (animal has black-stripes)
    then (animal is zebra)

r13:
    if (animal is bird)
       (animal does-not fly)
       (animal has long-neck)
       (animal has long-legs)
       (animal has black-and-white-colour)
    then (animal is ostrich)

r14:
    if (animal is bird)
```

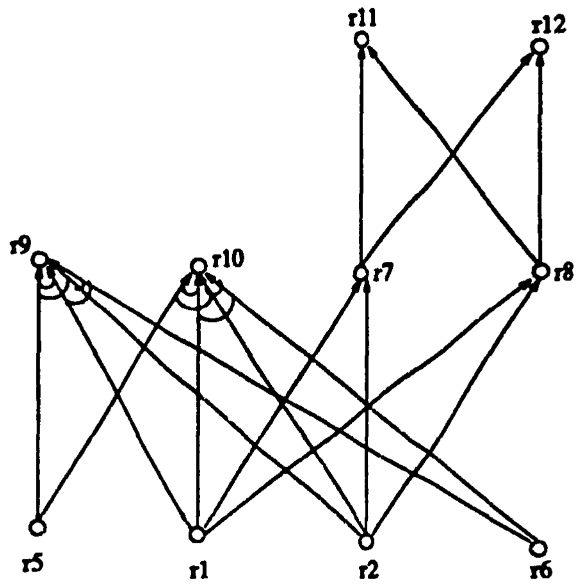


Figure 17: An AND/OR Digraph of the Animal Identification Rulebase

```
      (animal does-not fly)
      (animal does swim)
      (animal has black-and-white-colour)
   then (animal is penguin)


r15:
  if (animal is bird)
     (animal does fly-well)
  then (animal is albatross)
```

Two separate AND/OR digraphs deduced from the rule base are given in Figures 17 and 18 respectively, which correspond to two subsets of the rule base:

$$R_1 = \{r_1, r_2, r_5, r_6, r_7, r_8, r_9, r_{10}, r_{11}, r_{12},\}$$

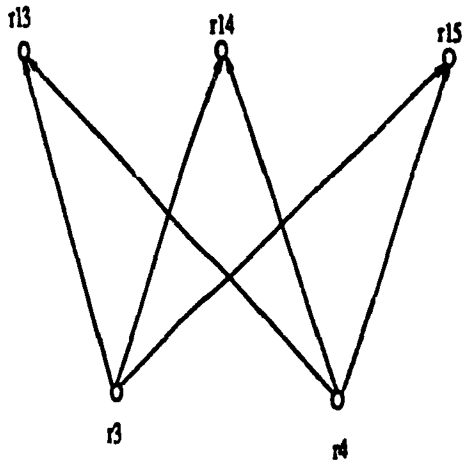


Figure 18: Another AND/OR Digraph of the Animal Identification Rulebase

$$R_2 = \{r_3, r_4, r_{13}, r_{14}, r_{15}\}$$

The paths contained in R_1 are:

1. $\{ \{r_5, r_1\}, \{r_9\} \}$
2. $\{ \{r_5, r_2\}, \{r_9\} \}$
3. $\{ \{r_1, r_6\}, \{r_9\} \}$
4. $\{ \{r_2, r_6\}, \{r_9\} \}$
5. $\{ \{r_5, r_1\}, \{r_{10}\} \}$
6. $\{ \{r_5, r_2\}, \{r_{10}\} \}$
7. $\{ \{r_2, r_6\}, \{r_{10}\} \}$
8. $\{ \{r_1, r_6\}, \{r_{10}\} \}$
9. $\{ \{r_1\}, \{r_7\}, \{r_{11}\} \}$

10. $\{ \{r_2\}, \{r_7\}, \{r_{11}\} \}$

11. $\{ \{r_1\}, \{r_7\}, \{r_{12}\} \}$

12. $\{ \{r_2\}, \{r_7\}, \{r_{12}\} \}$

13. $\{ \{r_1\}, \{r_8\}, \{r_{11}\} \}$

14. $\{ \{r_2\}, \{r_8\}, \{r_{11}\} \}$

15. $\{ \{r_1\}, \{r_8\}, \{r_{12}\} \}$

16. $\{ \{r_2\}, \{r_8\}, \{r_{12}\} \}$

And the paths in R_2 are:

1. $\{ \{r_3\}, \{r_{13}\} \}$

2. $\{ \{r_4\}, \{r_{13}\} \}$

3. $\{ \{r_3\}, \{r_{14}\} \}$

4. $\{ \{r_4\}, \{r_{14}\} \}$

5. $\{ \{r_3\}, \{r_{15}\} \}$

6. $\{ \{r_4\}, \{r_{15}\} \}$

Tables 3 and 4 show the derived values for the RC measurements, in which the qualifiers are (animal is), (animal has), (animal gives), *etc.* The values are (hair), (mammal), (bird), *etc.*

rules	$\rho(r_i)$
r_1	1
r_2	1
r_3	1
r_4	1
r_5	1
r_6	1
r_7	3
r_8	3
r_9	4
r_{10}	4
r_{11}	6
r_{12}	6
r_{13}	2
r_{14}	2
r_{15}	2

Table 3: Connectivity Measurement Value for Each Rule

items	π	v	RC
R_1	60.37	26	0.092
R_2	15.85	11	0.014

Table 4: Values Derived

Rule bases	ERC	NR	$ADSS$	$ABSS$	BC	NAC
R_1	2.938	10	1.50	2.67	4.36	102
R_2	2.935	5	1.00	2.00	2.00	57

Table 5: Results Measured by the metrics: ERC, NR, $ADSS$, $ABSS$, BC and NAC

Table 5 shows other metric measuring results on the two subsets of the sample rule base.

From the above results, we can see that

$$RC(R_1) > RC(R_2)$$

$$ERC(R_1) > ERC(R_2)$$

$$NR(R_1) > NR(R_2)$$

$$ADSS(R_1) > ADSS(R_2)$$

$$ABSS(R_1) > ABSS(R_2)$$

$$BC(R_1) > BC(R_2)$$

$$NAC(R_1) > NAC(R_2)$$

In addition to the "natural partition" of R which results in R_1 and R_2, we can also choose some rules from R_1 and R_2 to form some other artificial sub-rule bases. For example, the following "artificial partitions" could be made:

$$R_3 = \{r_1, r_2, r_7, r_8, r_{11}, r_{12}\}$$

items	π	v	RC
R_3	32	14	0.040
R_4	9.51	5	0.004

Table 6: Values derived

Rule bases	ERC	NR	$ADSS$	$ABSS$	BC	NAC
R_3	2.744	6	2.00	2.00	4.00	54
R_4	2.826	3	1.00	2.00	2.00	33

Table 7: Results Measured by the metrics: ERC, NR, $ADSS$, $ABSS$, BC and NAC

$$R_4 = \{r_3, r_4, r_{13}\}$$

The measuring results on R_3 and R_4 which are related to RC are shown in Table 6, and other measuring results are shown in Table 7.

Also, the conclusions about the rankings of R_3 and R_4 made by these measures can be obtained:

$$RC(R_3) > RC(R_4)$$

$$ERC(R_4) > ERC(R_3)$$

$$NR(R_3) > NR(R_4)$$

$$ADSS(R_3) > ADSS(R_4)$$

$$ABSS(R_3) = ABSS(R_4)$$

$$BC(R_3) > BC(R_4)$$

$$NAC(R_3) > NAC(R_4)$$

By partitioning a rule base into different sub-rule bases and applying the metric measurements on them, we can then compare the different parts of a rule base, and find out the key components in its formulation, which have the significant measuring values. This reveals one of metric applications which would help to maintain and test a rule base.

Summarizing and combining the above results, we have

$$RC(R_1) > RC(R_3) > RC(R_2) > RC(R_4)$$

$$ERC(R_1) > ERC(R_2) > ERC(R_4) > ERC(R_3)$$

$$NR(R_1) > NR(R_3) > NR(R_2) > NR(R_4)$$

$$ADSS(R_3) > ADSS(R_1) > ADSS(R_2), \quad ADSS(R_2) = ADSS(R_4)$$

$$ABSS(R_1) > ABSS(R_2), \quad ABSS(R_2) = ABSS(R_3) = ABSS(R_4)$$

$$BC(R_1) > BC(R_3) > BC(R_2), \quad BC(R_2) = BC(R_4)$$

$$NAC(R_1) > NAC(R_2) > NAC(R_3) > NAC(R_4)$$

These results show:

- Due to the different considerations (viewpoints) of the various measures, the rule bases are ranked differently in some cases, i. e. there are no unanimous rankings about these four rule bases, especially R_2 and R_3.
- R_1 has the highest value in most measures except $ADSS$, which meets our intuition. $ADSS$ measure does not rank R_1 high, because, in general, it is possible that the average depth of a rule base (in this cases, R_1) may be less than that of its sub-rule base (R_3).
- R_4 has the lowest value in most measures except ERC, which is reasonable. ERC is defined in terms of the occurrences of all items in a rule base, it does not relate to the overall structure of the rule base or search space directly, so, in some cases, its results are not so satisfactory. One modification over ERC measure is that not all the items in a rule base are counted for occurrence, only those which are in the matching patterns are considered, since the matching patterns (items) are generally believed to contribute significantly to rule bases.
- When applying the above measures to characterize some attributes of the rule bases, their validity and performance could be quite different. A further comparison and assessment of these measures will be conducted in Chapter 13.

9.3 Fault Detection, Isolation and Recovery Rule Base

The MMU FDIR system is a rule-based ES for the task of fault detection, isolation, and recovery of the Manned Maneuvering Unit (MMU) apparatus used in aerospace.

One part of its rule base is summarized and modified as follows (complete detail is described by Rushby [Rushby and Crow, 1990]):

```
Rule A:

 (defrule cea-a-test-prime-mode

    (check status of MMU)

    (side a on)

    (side b on)

 => (assert (failure cea))

    (assert (suspect a)))


Rule B:

  (defrule cea-a-test-backup-mode

    (check status of MMU)

    (side a on)

    (side b off)

 => (assert (failure cea))

    (assert (suspect a)))


Rule C:

  (defrule cea-b-test-backup-mode

    (check status of MMU)

    (side a off)

    (side b on)
 => (assert (failure cea))
    (assert (suspect b)))
```

Rule D:

```
  (defrule test-failure-cea-suspect-a
    ?a<- (failure cea)
    (suspect a)
    (side a on)
    ?b <- (side b on)
 => (retract ?a ?b)
    (assert (side b off)))
```

Rule E:

```
  (defrule test-failure-cea-a-good
    (not (failure cea))
    ?x <- (suspect a)
    ?a <- (side b off)
    ?b <- (side a on)
 => (retract ?a ?b ?x)
    (assert (side b on))
    (assert (side a off))
    (assert (cea-a-good)))
```

Rule F:

```
  (defrule test-a-cea-side-b-good
     (not (failure cea))
     (side b on)
     (side a off)
     (cea-a-good)
 => (assert (failure cea-coupled)))
```

Rule G:

```
  (defrule print-failure-cea-a
     (not (failure cea))
     (side a off)
     (side b on)
     (failure cea-a)
 => (assert (done)))
```

Rule H:

```
  (defrule print-failure-cea-b
     (not (failure cea))
     (side b off)
     (side a on)
     (failure cea-b)
 => (assert (done)))
```

Rule I:

```
(defrule test-failure-cea-a-bad
  ?a <- (failure cea)
  (suspect a)
  ?b <- (side b off)
  ?c <- (side a on)
  (not (failure cea-b))
 => (retract ?a ?b ?c)
  (assert (failure cea-a)))
```

Rule J:

```
(defrule test-failure-cea-b-bad
  ?a <- (failure cea)
  (suspect b)
  ?b <- (side a off)
  ?c <- (side b on)
  (not (failure cea-a))
 => (retract ?a ?b ?c)
  (assert (side a on))
  (assert (side b off))
  (assert (failure cea-b)))
```

```
Rule K:

  (defrule test-a-cea-side-a-and-b

    (failure cea)

    (failure cea-a)

    ?x <- (side b on)

    (side a off)

 => (retract ?x)

    (assert (failure cea-a-b))

    (assert (side b off)))


Rule L:

  (defrule test-b-cea-side-a-and-b

    (failure cea)

    (failure cea-b)

    ?x <- (side a on)

    (side b off)

 => (retract ?x)

    (assert (failure cea-a-b))

    (assert (side a off)))
```

Based on an analysis of the above rule base, the corresponding AND/OR digraph is constructed (Figure 19), which shows the dependency among the rules. From the digraph, we can derive the following five paths:

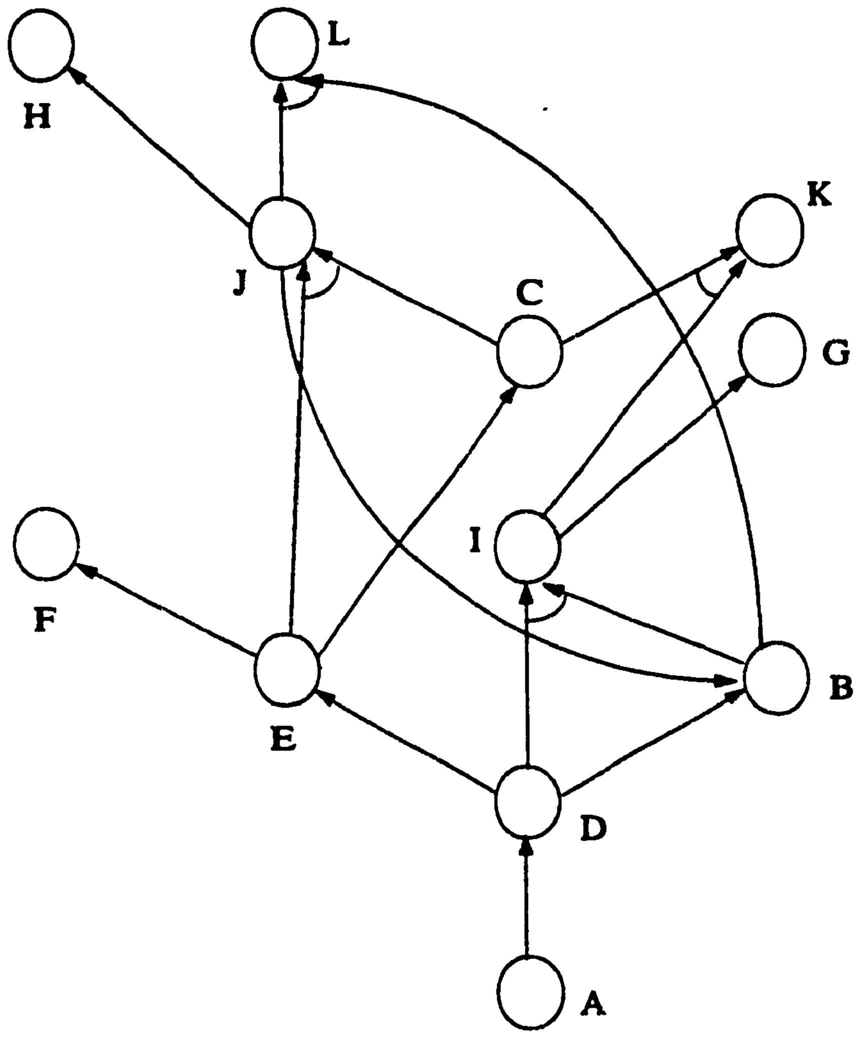


Figure 19: Rule Dependency of the Sample Rule Base

rules	$\rho(r_i)$
Rule A	1
Rule B	12
Rule C	6
Rule D	3
Rule E	5
Rule F	5
Rule G	13
Rule H	8
Rule I	13
Rule J	8
Rule K	13
Rule L	12

Table 8: Connectivity Measurement Value for Each Rule

Rule Base	π	v	RC
MMU FDIR	65.01	26	0.17

Table 9: Values derived

1. { {A}, {D}, {E}, {F} }

2. { {A}, {D}, {E}, {C, E}, {J}, {B, J}, {L} }

3. { {A}, {D}, {E}, {C, E}, {J}, {H} }

4. { {A}, {D}, {B, D}, {I}, {G} }

5. { {A}, {D}, {B, D}, {I}, {I, C} {K} }

The measuring results are listed in Tables 8, 9, 10.

It is obvious that this rule base is more complicated than the first sample rule base, the measuring results reflect this intuition by assigning the second sample rule

NR	ADSS	ABSS	NAC	BC	RC	ERC
12	4.6	1.55	100	7.51	0.17	4.01

Table 10: Several Measures on the Sample Rulebase

base higher values than those of the first rule base.

Chapter 10

Implementing Measuring Tool

10.1 Introduction

To support the measuring task, an automatic measuring tool (system) is needed. In this thesis, such a tool is implemented using high level programming languages, which will ease the measures of rule bases and analysis of the measuring results. Different rule bases to be measured form the input data to the tool. In the following sections, we will briefly introduce this tool and address several implementation issues such as the system architecture, design, rule base class (main data structure), and implemented functions.

10.2 Overall System Architecture

The structure of the measuring system (tool) built to support our presented metric measurements is shown in Figure 21. It consists of three major parts.

- FILTER, which is designed to parse the rule bases to be measured. These rule bases are written in different rule base description languages, and they

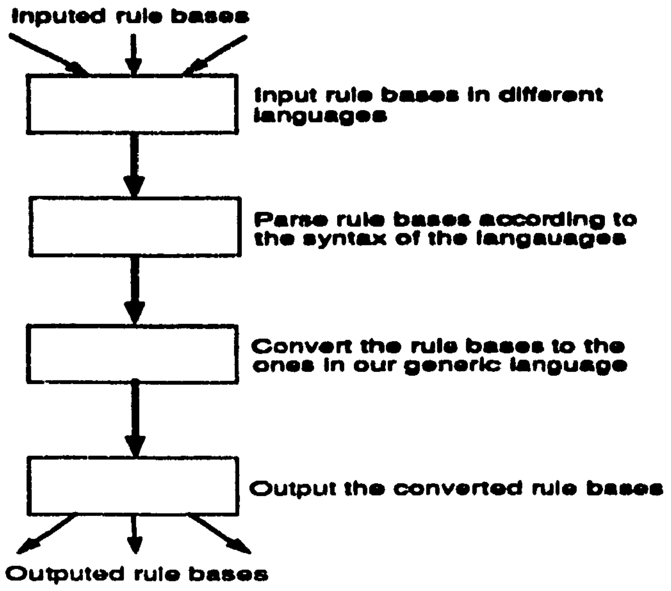

Inputed rule bases
Input rule bases in different languages
Parse rule bases according to the syntax of the langauages
Convert the rule bases to the ones in our generic language
Output the converted rule bases
Outputed rule bases

Figure 20: Processing Steps of FILTER

will be converted by FILTER into those described by our rule base description language. FILTER is written in PROLOG which is generally believed to be a good tool for writing the parser. Figure 20 summarizes the processing steps involved in FILTER.

- MEASURES, which is the central component of the tool. According to the formulation of different metrics, it performs the measures on the rule bases converted by FILTER, and outputs the desired ES measuring values. It is written in C/C++, the prevalent tool, and the detailed implementation will be shown in the following sections.

- ANALYSIS, which will further analyze the measuring results given by MEASURES. It compares the metrics, performs some statistical analysis, and produces further results such as the regression line between some metric and the anomaly rate, which can be used for the prediction. Some existing statistical analysis packages will be utilized, like the ones for correlation and regression analysis implemented in Matlab. New features, like the non-parametric statistical comparison will be added by writing new functions in C/C++.

10.3 Implementation

The language tools — PROLOG, C/C++ and Matlab are chosen for implementing the measuring system. The main considerations for using these tools are:

- PROLOG is believed to be a good tool for implementing a parser, because PROLOG's DCG (definite clause grammars) facility provides a convenient notation for implementing formal grammars in PROLOG, defining and expressing a language, and parsing strings to be evaluated according to the stated grammars which is directly executable by PROLOG as a syntax analyzer.

  Example 10.1 To recognize and parse the CLIP rules, the following grammar rules in DCG are constructed and used:

```
defrule(header(Name, Comment), Antecedents, Consequents) -->
   ['('], ['defrule'], symbol(Name), string(Comment),
   antecedents(Antecedents), ['=>'], consequents(Consequents), [')'].
```

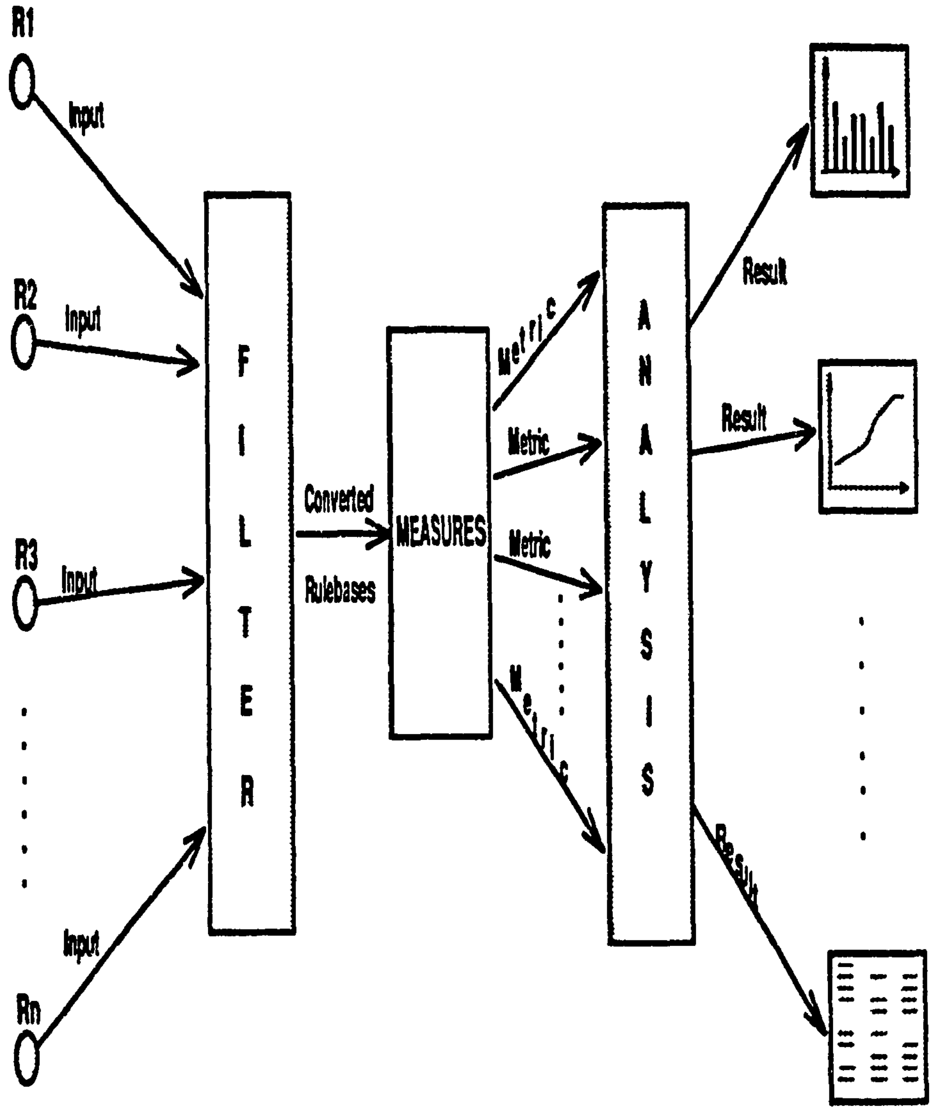


Figure 21: Overall Measuring Structure

```
defrule(header(Name, '""'), Antecedents, Consequents) -->
    ['('], ['defrule'], symbol(Name), antecedents(Antecedents),
    ['=>'], consequents(Consequents), [')'].
```

Based on the DCG grammars and some other descriptions, the rules in target rule bases can be parsed, analyzed and converted to the required formats. For other kinds of rule base languages, a similar set of rules, like those presented in example 10.1, can be built for the analysis. Hence, PROLOG is ideal for the implementation of FILTER.

- The popularity of C language is well-known. In addition, as the extension of C, C++ entails the application of object-oriented paradigm in the design and implementation of systems, it adds more desired features to the implemented software. One of them is the improved *expendability* that could facilitate the additions of new functions, and is an important factor to be considered in designing our measuring system, because of the possible addition of new measures in the future. C/C++ are then chosen as the tool to implement MEASURES.

- Matlab. Matlab is a tool designed for the mathematical analysis. The functions in it include the correlation and regression analysis which will be used for calculating the correlation coefficient between two variables, and the coefficients of the regression model, and plotting the results. These functions plus other statistical functions implemented by us constitute ANALYSIS.

10.4 Data Structure

Class has been proven to be an effective and useful data structure in the design of software systems. Many features that object-oriented paradigm brings come from the introduction of classes. Since a rule base, the main focus of the system, is an abstract data object, it is defined as a class in the system, which contains the rules, other attributes of the rules, and the methods for measuring the rules and rule bases.

The attributes associated with a rule base class include:

- Rules, which can be defined as the classes themselves. The relationship between the rule base class and rule class is the *containership* which indicates that the rule class is a data member of rule base class. The attributes of a rule class can be the *name, antecedents* and *consequents.* All the rules in a rule base class are linked together and form a list of objects, which are indicated by the *head pointer* and *tail pointer* (Figure 22).
- Head pointer, which points to the head of the rule list in a rule base.
- Tail pointer, which points to the tail of the rule list in a rule base.
- Data RC, ERC, NR, $ADSS$, $ABSS$, BC and NAC which store the values of different rule base measures, defined in the previous chapters.

The main methods (functions) associated with a rule base class are:

- "*loadrule*", which reads in the rules from the file and forms the linked list of rules.

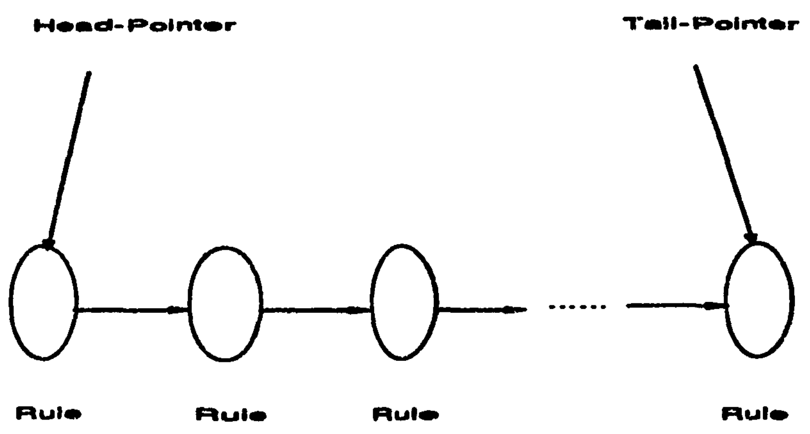


Figure 22: Linked List of Rc' s

- *"dependency"*, which analyzes the interrelation among the rules and stores the relationships.
- *RC_CAL*, *ERC_CAL*, *NR_CAL*, *ADSS_CAL*, *ABSS_CAL*, *BC_CAL* and *NAC_CAL*, which are the *methods* for calculating the different metrics and storing the values in the corresponding attributes.

Representing the rule bases in the above structure makes the further addition of new measures convenient, which will be the addition of new *methods* based on the existing data members and methods. Figure 23 illustrates the structure of the rule base class, and its attributes as well as methods.

10.5 Measuring Library

The set of measuring functions (methods) for the presented metrics form the measuring library which can be extended by adding more functions for more measures. Table3 11 to Table 17 list the functions which were implemented in the system.

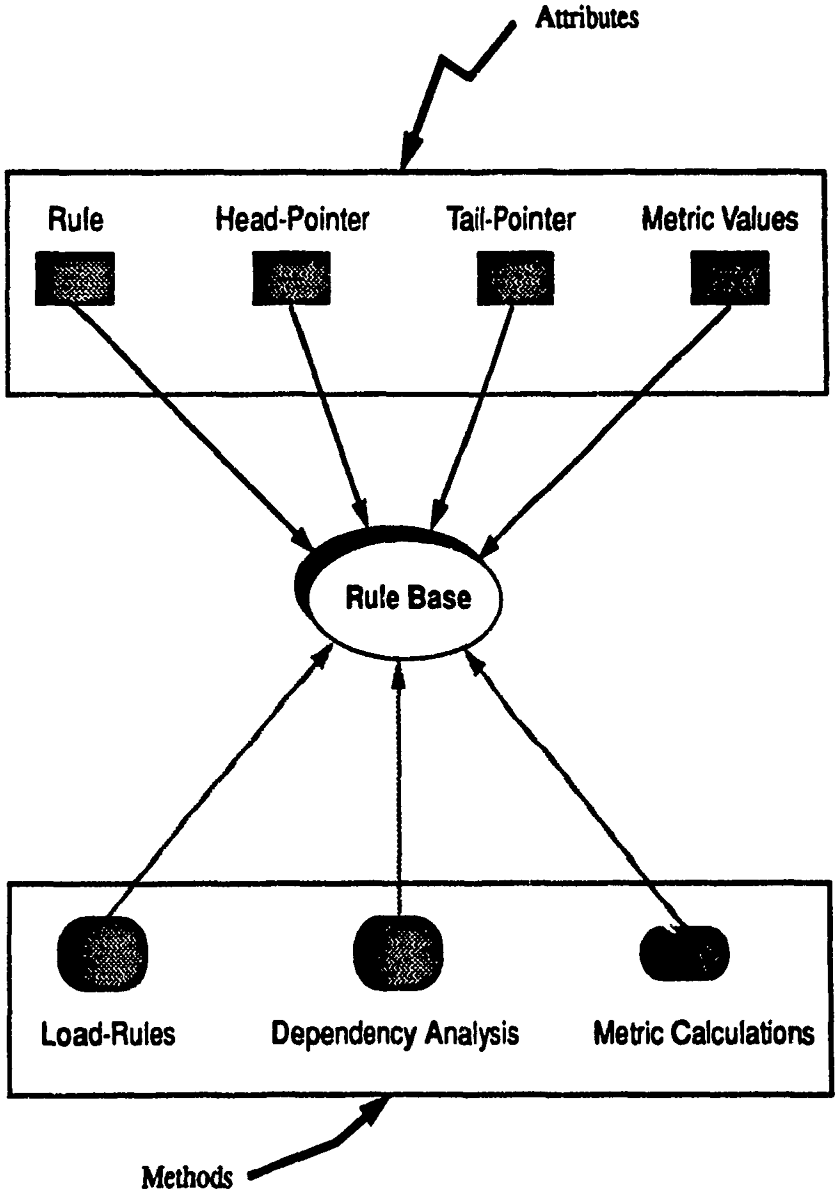


Figure 23: A Rule Base Class

Function	*RC_CAL*
Description	Applying to a rule base and measuring it according to *RC* formalism
Input	A rule base whose rules are represented in a list structure
Output	*RC* value

Table 11: Description of *RC_CAL* Function

Function	*ERC_CAL*
Description	Applying to a rule base and measuring it according to *ERC* formalism
Input	A rule base whose rules are represented in a list structure
Output	*ERC* value

Table 12: Description of *ERC_CAL* Function

Function	*NR_CAL*
Description	Applying to a rule base and measuring it according to *NR* formalism
Input	A rule base whose rules are represented in a list structure
Output	*NR* value

Table 13: Description of *NR_CAL* Function

Function	*ADSS_CAL*
Description	Applying to a rule base and measuring it according to *ADSS* formalism
Input	A rule base whose rules are represented in a list structure
Output	*ADSS* value

Table 14: Description of *ADSS_CAL* Function

Function	*ABSS_CAL*
Description	Applying to a rule base and measuring it according to *ABSS* formalism
Input	A rule base whose rules are represented in a list structure
Output	*ABSS* value

Table 15: Description of *ABSS_CAL* Function

Function	*BC_CAL*
Description	Applying to a rule base and measuring it according to *BC* formalism
Input	A rule base whose rules are represented in a list structure
Output	*BC* value

Table 16: Description of *BC_CAL* Function

Function	NAC_CAL
Description	Applying to a rule base and measuring it according to NAC formalism
Input	A rule base whose rules are represented in a list structure
Output	NAC value

Table 17: Description of NAC_CAL Function

Chapter 11

Measuring Results

11.1 Introduction

To test the effectiveness of the presented metrics, we apply them to measure different rule bases. 76 rule bases are collected as test data. In addition, the anomaly rates found in these rule bases are also counted and presented. The metric measuring results plus the anomaly rates provide the basis for further evaluating the performance of these metrics.

11.2 Data Collected

Two sources of data were collected for the measures. The first set of data came from the students' projects and another set from real expert systems. Table 18 lists the main application domains of the test data.

Application Domains	
First Set of Data	Second Set of Data
Software Selection	Telecommunication Line Diagnosis
Mutual Fund Investment	Fault Detection, Isolation and Recovery
Medical Diagnosis	Aerospace Apparatus Diagnosis
Library Management	Tape Selection
Course Selection	Medical Diagnosis
Automobile Fault Diagnosis	
Chess	
Hospital Management	
Software Evaluation	
Phone Line Fault Detection	
University Selection	
Steel Wire Rope Selection	
Tree Identification	
Animal Identification	
Trip Route Finding	
Computer Purchasing	

Table 18: Application Domains of the Test Data

11.3 Results

Figures 24, 25, 26, 27, 28, 29 and 30 show the results of applying the metrics to the first set of data[1] and Figure 31 shows the anomaly rates of this set of data. Figures 32, 33, 34, 35, 36, 37 and 38 show the results of applying the metrics to the second set of data and Figure 39 shows the anomaly rates of this set of data. In these figures, the horizontal axis represents different rule bases, and the vertical axis denotes the metric measuring results of these rule bases.

[1]In measuring, for comparison reasons, the rules in these rule bases that mainly perform "input" and "output", or do not connect with other rules are not taken into account.

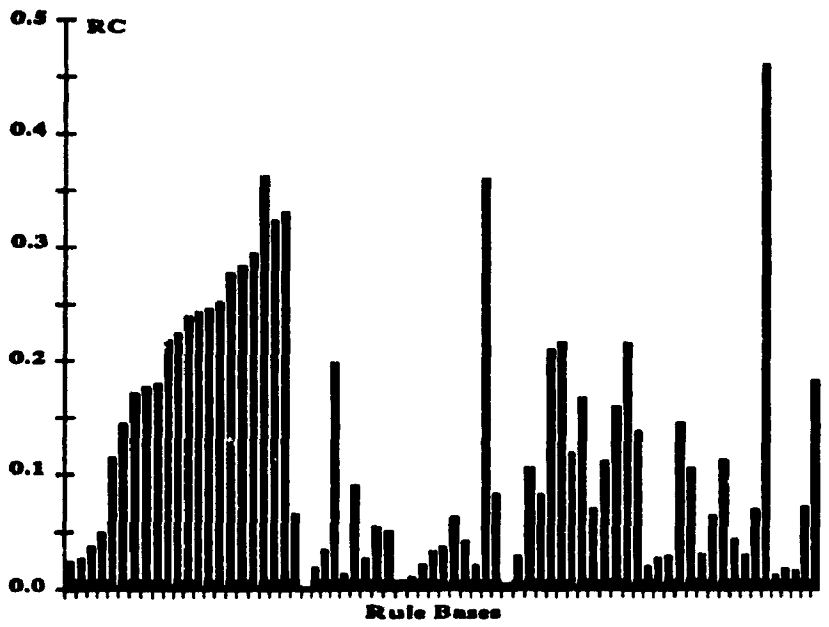


Figure 24: Results of Applying *RC* to the Test Data

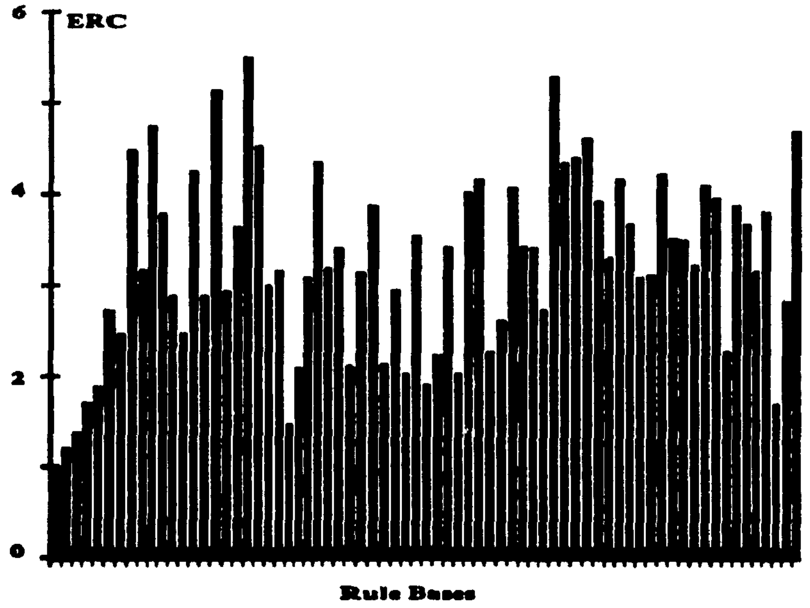


Figure 25: Results of Applying *ERC* to the Test Data

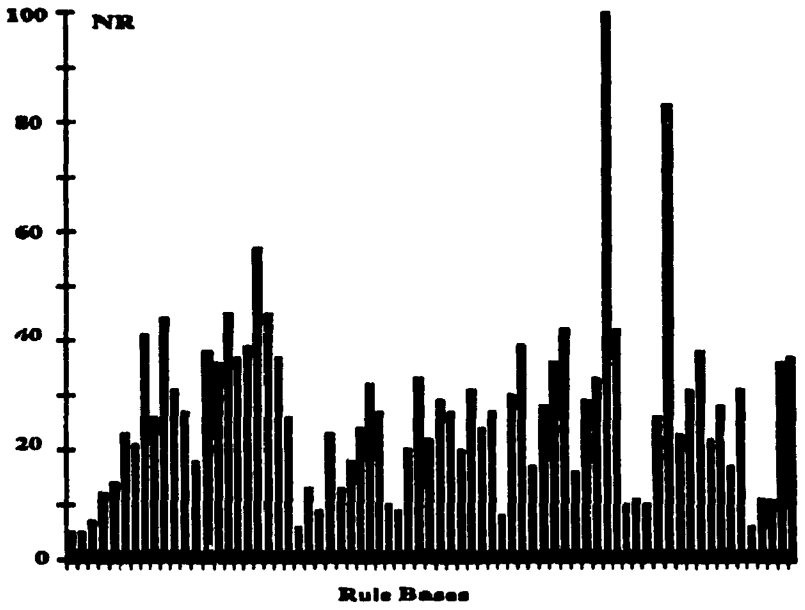


Figure 26: Results of Applying *NR* to the Test Data

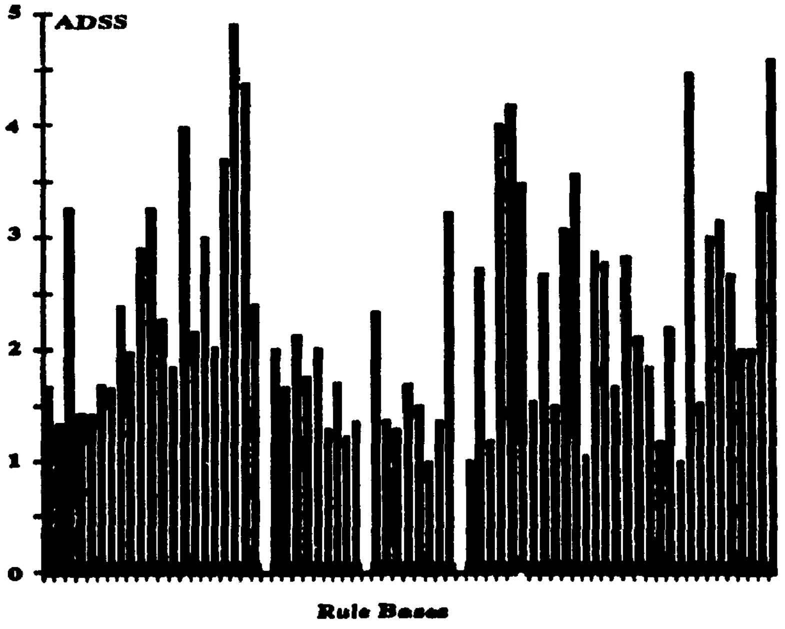


Figure 27: Results of Applying *ADSS* to Test Data

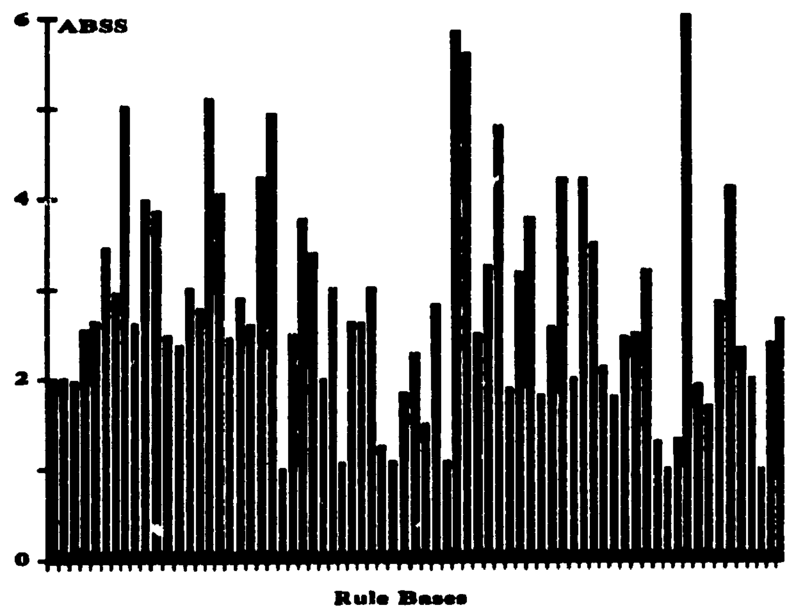


Figure 28: Results of Applying *ABSS* to the Test Data

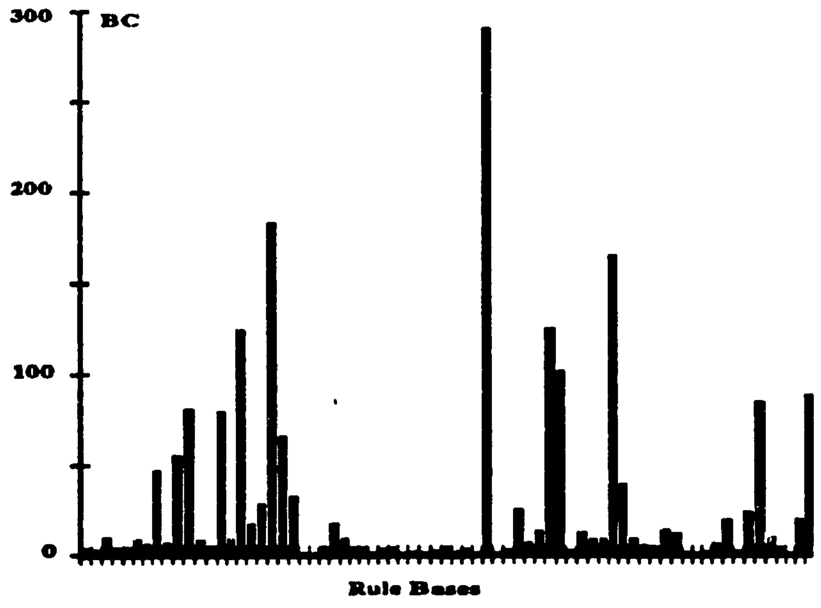


Figure 29: Results of Applying *BC* to the Test Data

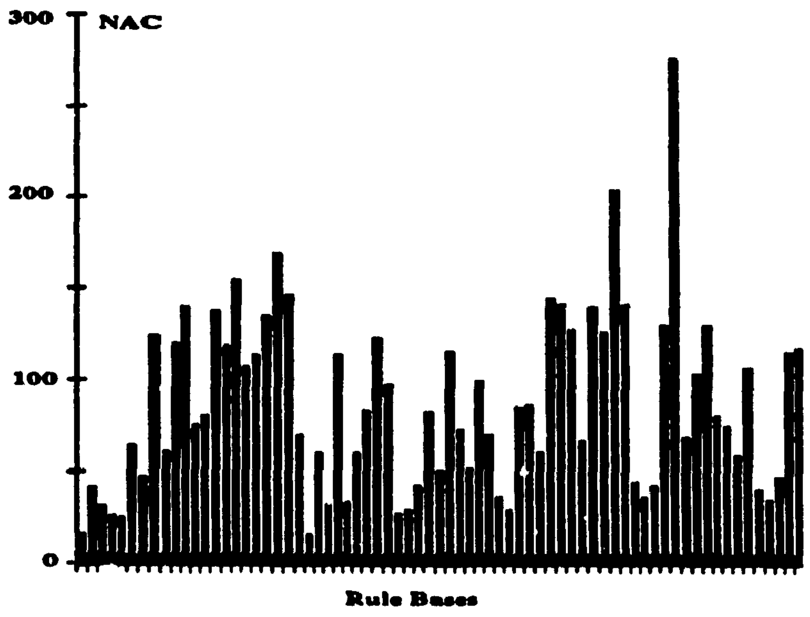


Figure 30: Results of Applying *NAC* to the Test Data

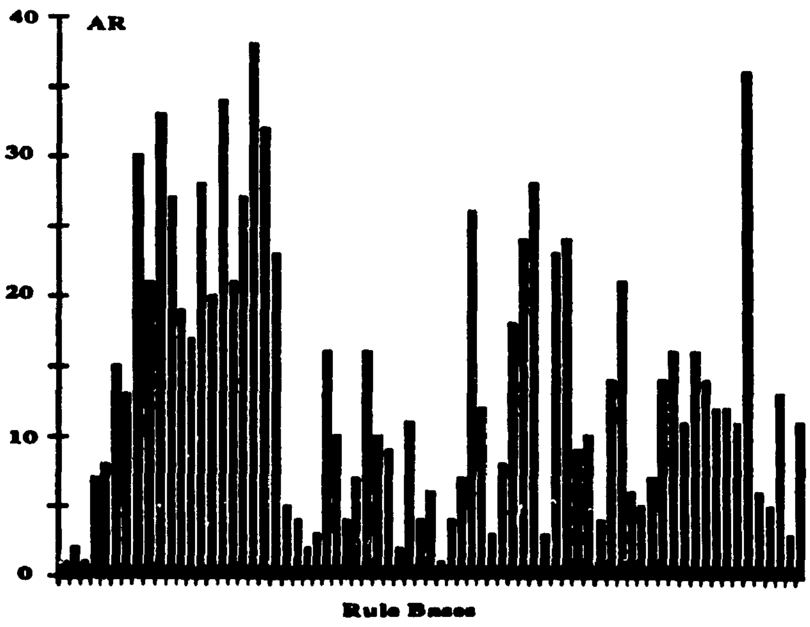


Figure 31: Anomaly Rates of the Test Data

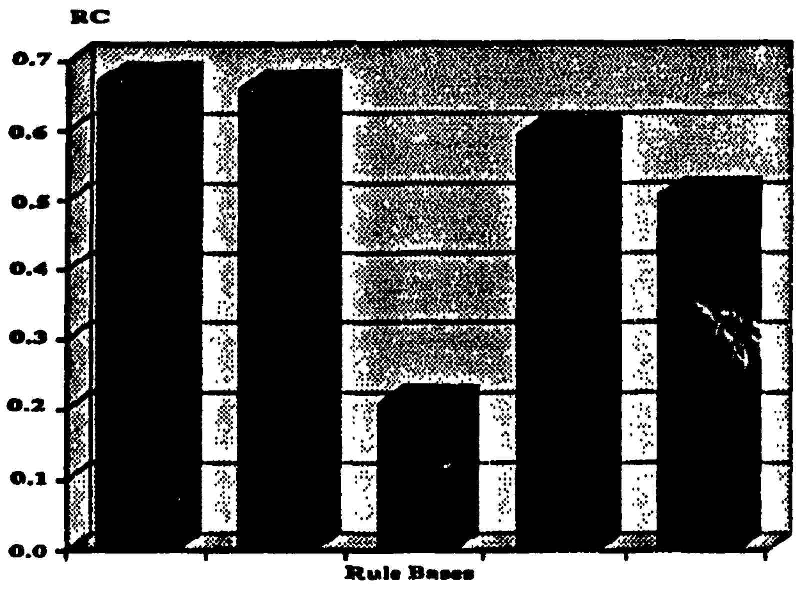


Figure 32: Results of Applying *RC* to the Test Data

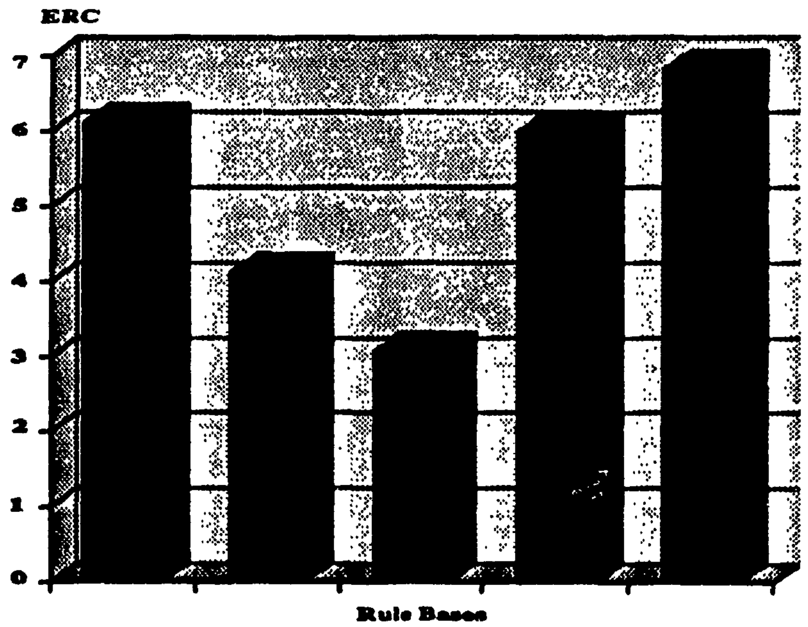


Figure 33: Results of Applying *ERC* to the Test Data

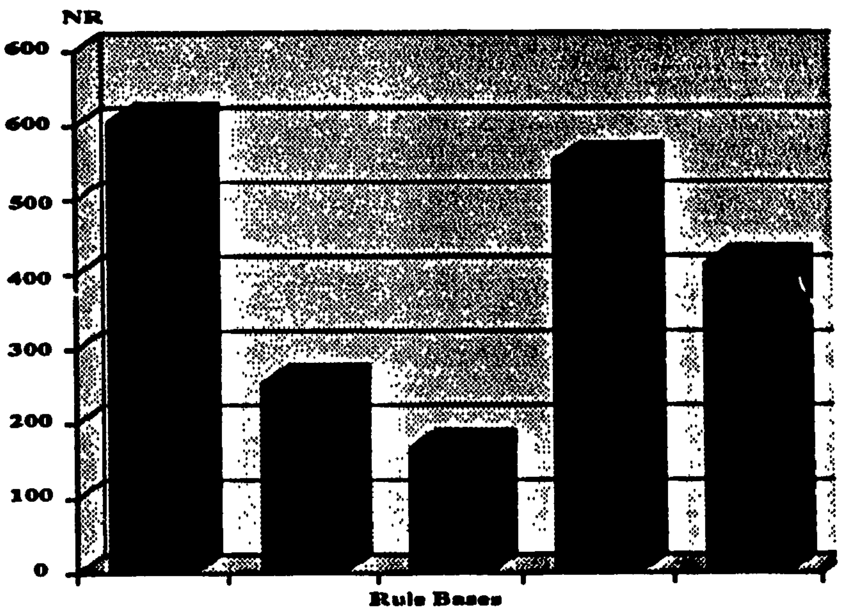


Figure 34: Results of Applying *NR* to the Test Data

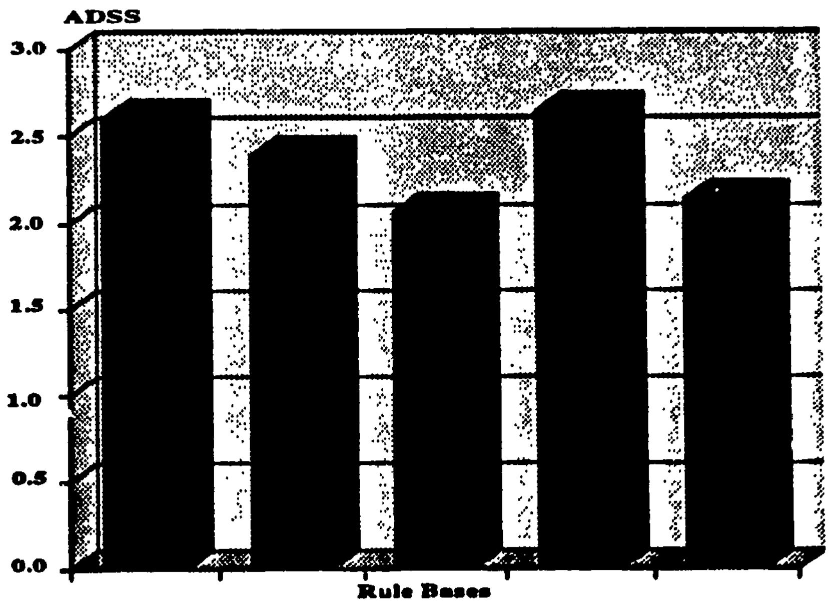


Figure 35: Results of Applying *ADSS* to Test Data

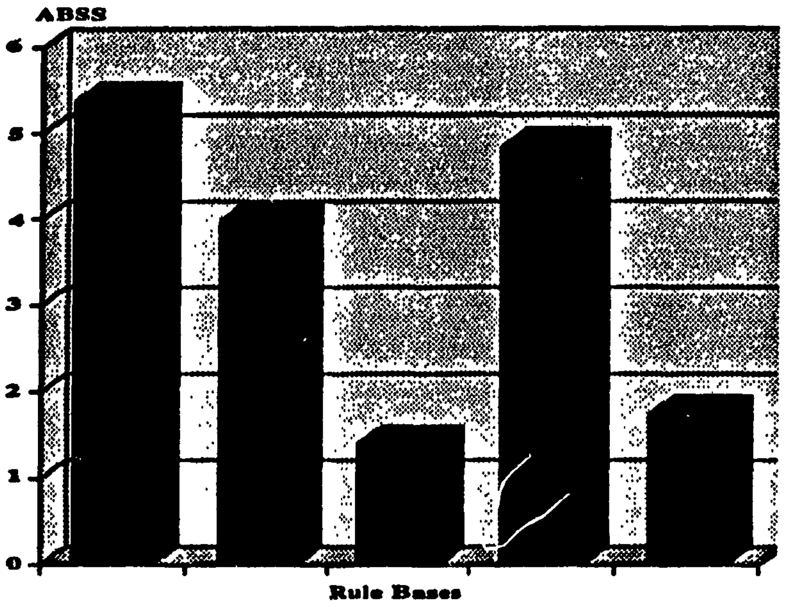


Figure 36: Results of Applying *ABSS* to the Test Data

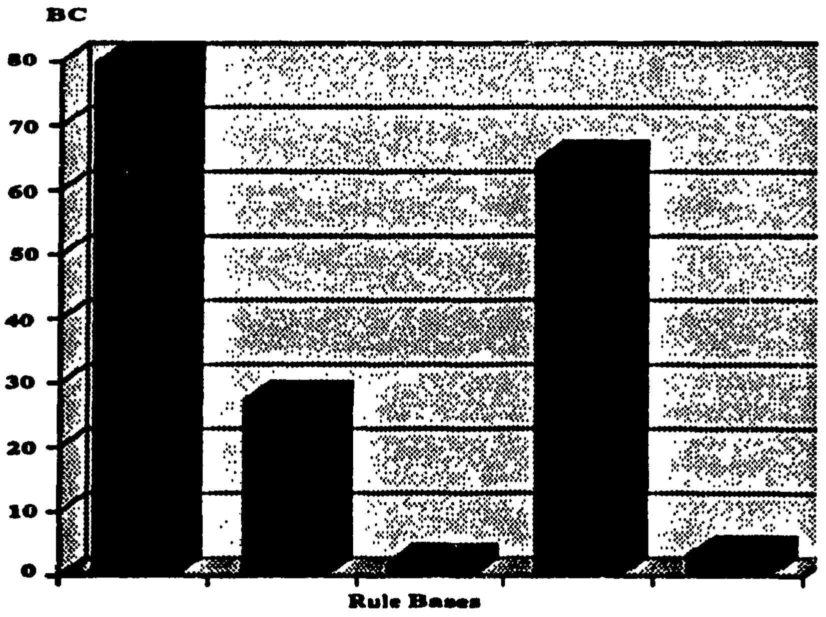


Figure 37: Results of Applying *BC* to the Test Data

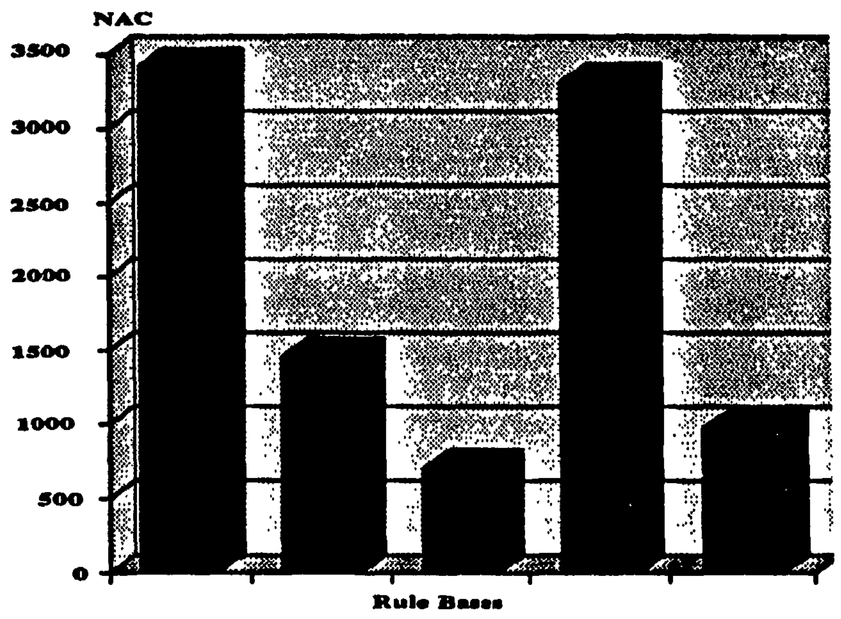


Figure 38: Results of Applying *NAC* to the Test Data

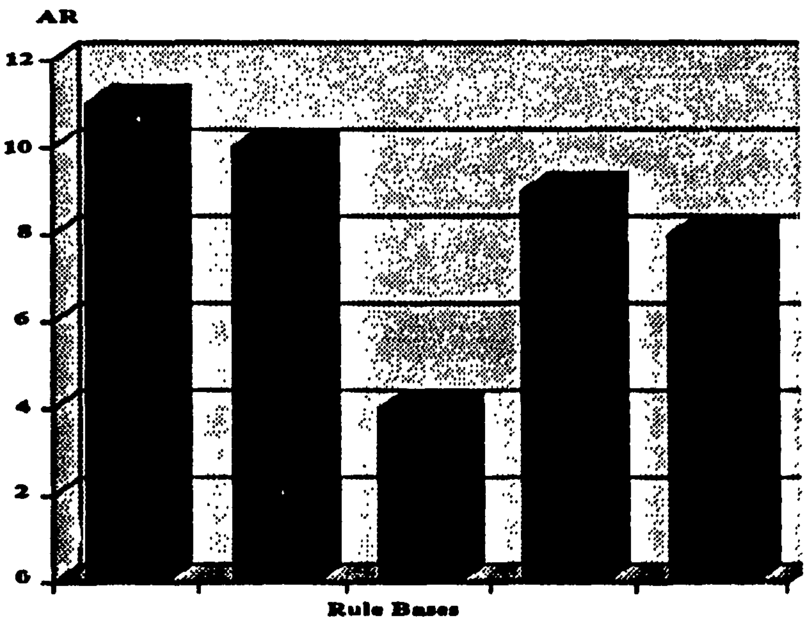


Figure 39: Anomaly Rates of the Test Data

From the results indicated in the figures, we can see that

- The measures of the second set of test data are in general higher than the first. This is due to the nature of the real expert systems, i.e. their complex domains and knowledge;
- The ranges and distributions of the different metric values vary significantly. In two typical cases, RC value varies from 0 to 0.7. However, for ERC, its values range from 0 to 3500.
- There exist some normal ranges in which the measures on most rule bases fall. This normal ranges differ as the applications vary.

Further comparisons and applications of the metrics and their measuring values will be discussed in the next section and Chapter 13.

11.4 Applying Metrics

The measuring results presented in the previous section show the distributions of the metric measuring values, which can be further examined to determine the normal range of the metric values in order to identify abnormal rule bases whose metric values exceed the normal range. Figures 40, 41, 42, 43, 44, 45 and 46 show the calculated percentages of rule bases whose metric values fall within different ranges.

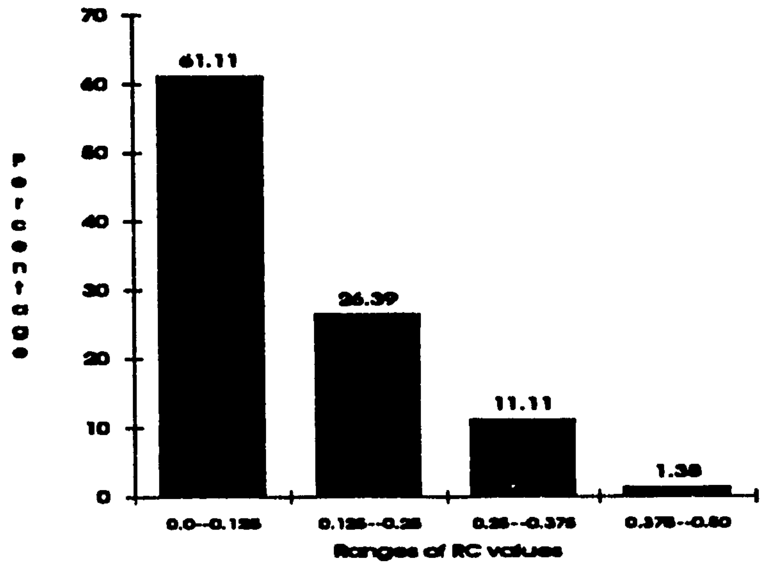


Figure 40: Percentage Profile for Expert Systems

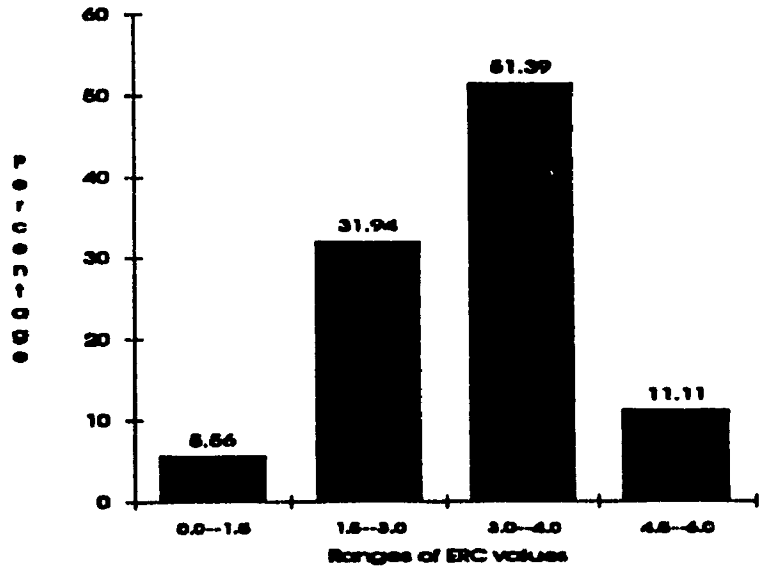


Figure 41: Percentage Profile for Expert Systems

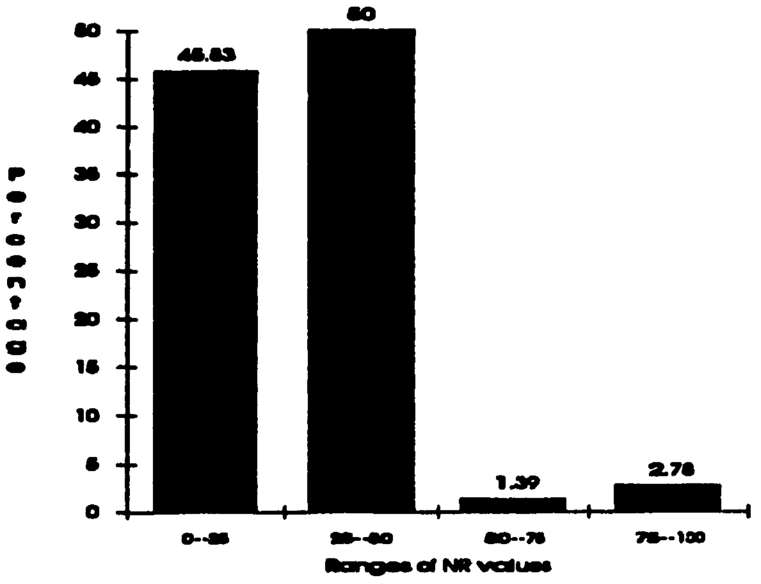


Figure 42: Percentage Profile for Expert Systems

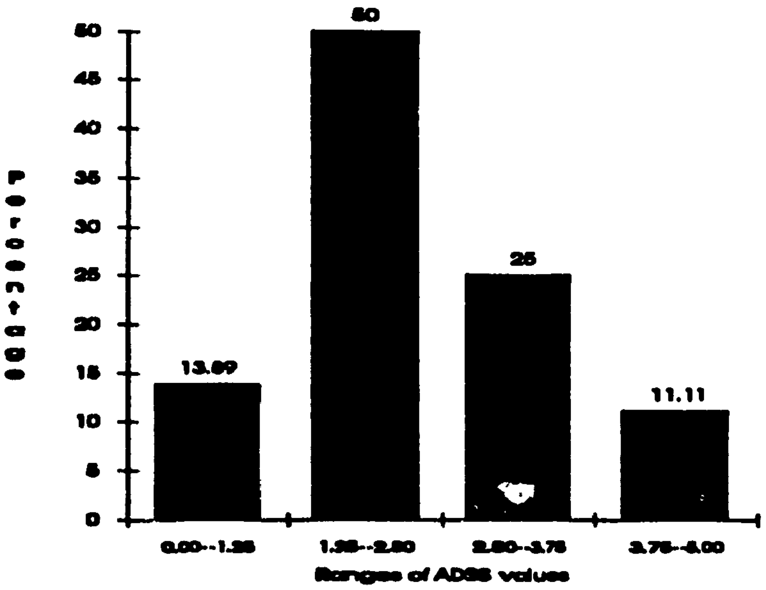


Figure 43: Percentage Profile for Expert Systems

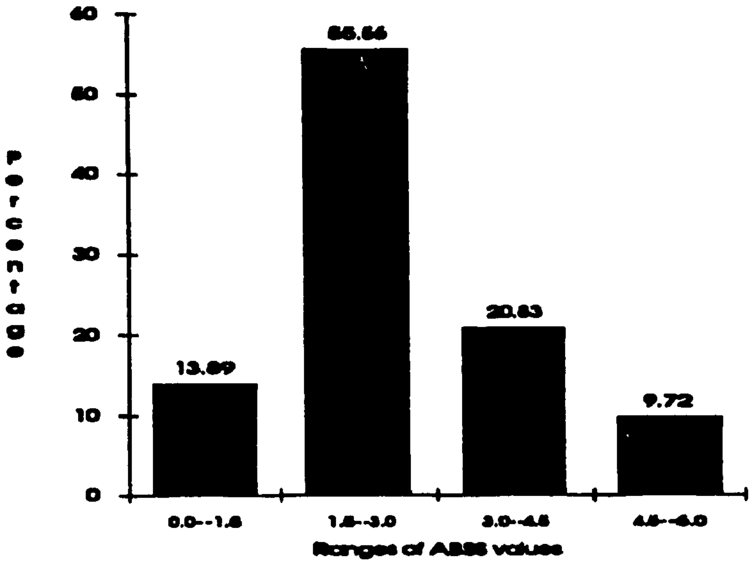


Figure 44: Percentage Profile for Expert Systems

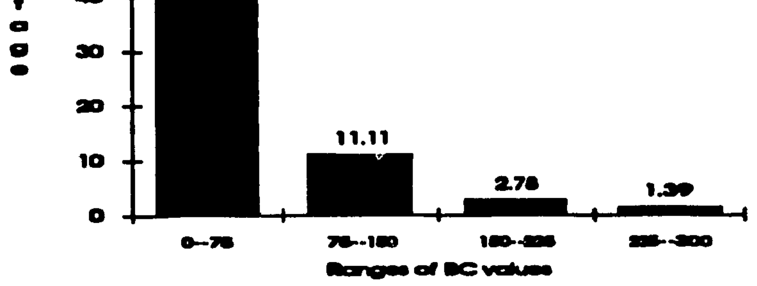


Figure 45: Percentage Profile for Expert Systems

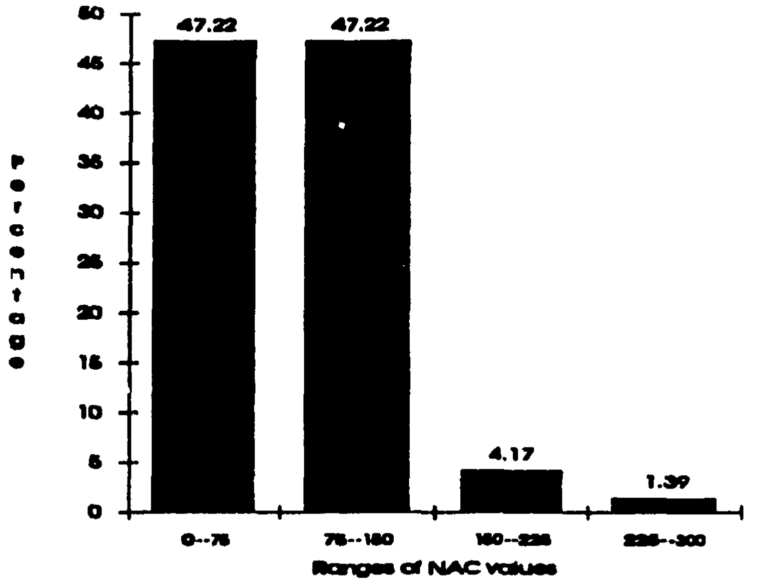


Figure 46: Percentage Profile for Expert Systems

We can further refine the above ranges (percentages) and obtain the normal ranges where most rule bases fall in. Rule bases that have abnormal ranges are then identified as abnormal rule bases. There may exist many possible reasons for the abnormal values, such as the complicated knowledge, inappropriate representation of knowledge, and anomalies in the knowledge. The project development team and experts need to work together to find out the the most likely reasons or explanations. The normal ranges of our presented metrics for the student's projects are summarized in Table 19 which can be used to

- Identify existing abnormal rule bases;
- Guide the design of future rule bases;
- Evaluate the performance of students and the student classes.

Another important application of the metrics, as indicated in the previous chapters, is the use of their measuring values to assess the quality model presented in Chapter 5. One way to perform the assessment is to profile the quality by its quality factors which are further profiled by the quality criteria, and finally the criteria are

Metrics	Normal Range	Percentage
RC	0.0 — 0.3	93.06%
ERC	1.2 — 5.1	93.06%
NR	6 — 46	93.06%
$ADSS$	1.0 — 4.5	91.67%
$ABSS$	1.0 — 5.0	93.06%
BC	1.0 — 110.0	93.06%
NAC	20 — 155	93.06%

Table 19: Usual Ranges for the Metrics

evaluated by using the metric values. In this way, the quality profile is represented as a histogram that identifies the degree of quality factors achieved, which can be based on a scale of 0 to 100. The following steps describe the procedure of our profile method:

1. Select and compose the metrics for a quality factor, suppose there are m_1, m_2, ..., m_n;
2. Decide the normal range for m_i $(1 \leq i \leq n)$;
3. Assign a weight w_i (percentage) for each metric m_i such that $w_1+w_2+......+w_n =$ 100%;
4. Obtain the measuring value v_i for each metric m_i $(1 \leq i \leq n)$;
5. Calculate the deviation σ_i of each metric value v_i from its normal range;
6. Calculate the percentage profile for the presented quality factor by using the formula:

$$100\% - \sigma_1 \times w_1 - \sigma_2 \times w_2 - \cdots\cdots - \sigma_n \times w_n$$

7. Output the profile.

Example 11.1 To evaluating the *testability*, we may group and use the metrics of *NR, ADSS, ABSS* and *RC*, that is, these four metrics are designed to compose the *testability*. Based on the normal ranges of these metrics, the testability of different rule bases is assessed. The rule bases which meet all the ranges of the above four metrics that compose this quality factor will be characterized by 100% testability. Others are evaluated in terms of their variation from each normal range, and will have a less than 100% testability. Figure 47 shows a rule base with the different metric values, and its profile for the quality factor — *testability.* Other quality factors can also be evaluated in the same way with a different composition of metrics. The overall profiles of quality factors then demonstrate the quality of an ES. How to compose the metrics for profiling different quality factors depends on the managers, expert, development team, users, and applications of the ES, which is beyond the scope of this research.

Other applications of metrics include their use as predictors which will be discussed later.

Metrics	Values
RC	*0.4*
ADSS	*4.0*
ABSS	*5.5*
NR	*4.5*

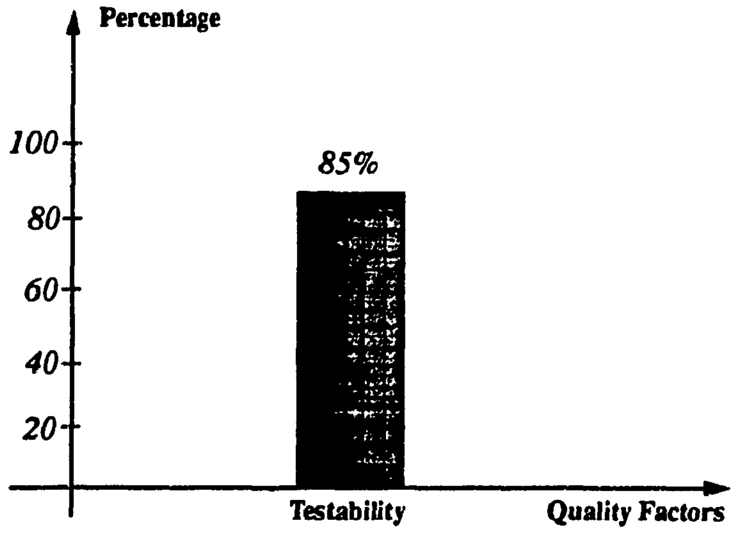


Figure 47: An Example of Profiling Quality Factor

Chapter 12

Statistical Analysis and Testing

12.1 Introduction

Having observed the metric measuring results presented in the last chapter, we now proceed to evaluate and assess the validity and performance of these metrics, and to compare and select appropriate metric(s) for various applications. Statistical analysis and testing provide a means to help us to achieve these objectives. This chapter presents and discusses several statistical techniques applied to the analysis of the metrics described in this thesis.

12.2 Correlation Between Two Variables

Correlation analysis examines whether two variables are inter-related, and give a quantitative value for describing the degree of relatedness. In the following discussion, we will use X to represent one variable with a set of measuring values $\{x_1, x_2, \cdots x_i \cdots, x_n\}$, Y to represent another variable with another set of measuring values $\{y_1, y_2, \cdots y_i \cdots, y_n\}$, and $n > 1$. To analyze the relation between X

and Y, we usually first examine the scatterplot diagrams between X and Y.

12.2.1 Plotted Diagrams

By plotting one variable X against another Y using x_i and y_i values ($1 \leq i \leq n$), we can obtain a scatter diagram which demonstrates the relationship between X and Y. This is useful when at first we don't have any knowledge about the relationship between X and Y. A plotted diagram can give us some indications such as the existence of a po itive relation between X and Y, even though we cannot precisely specify the relations at this moment. For example, Figure 48 suggests that there is a relation between X and Y, and that relation may be described by a linear model.

12.2.2 Correlation Coefficient

To further obtain the precise and quantitative information about the relation between X and Y, the correlation coefficient r which shows the degree of correlation can be calculated by the following formulas:

$$r = \frac{\sum_{j=1}^{j=n}(x_j - \overline{x})(y_j - \overline{y})}{\sqrt{\sum_{j=1}^{j=n}(x_j - \overline{x})^2 \sum_{j=1}^{j=n}(y_j - \overline{y})^2}}$$

where

$$\overline{x} = \frac{1}{n}\sum_{j=1}^{j=n} x_i$$

$$\overline{y} = \frac{1}{n}\sum_{j=1}^{j=n} y_i$$

It can be shown that r varies from -1 to $+1$.

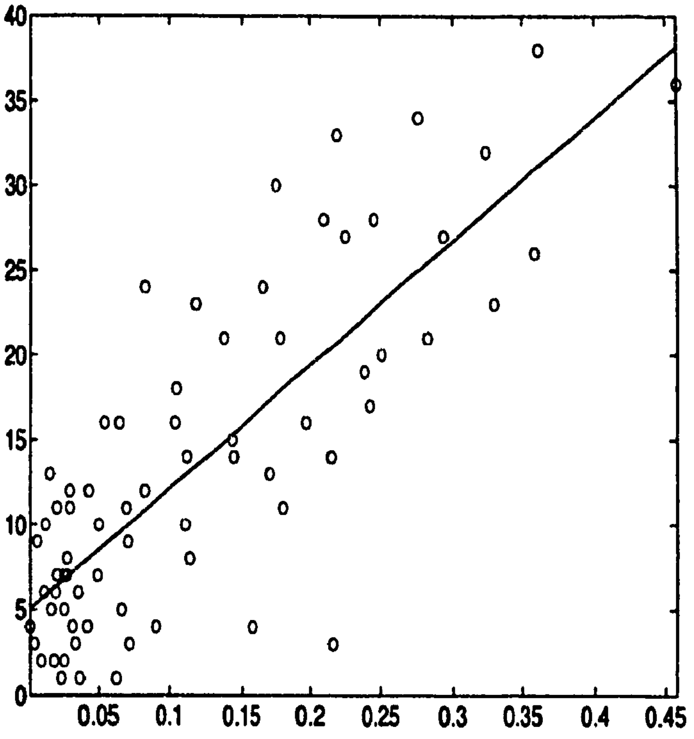


Figure 48: An Example of Plotted Diagram

In addition, when r is near $+1$, it indicates a strong positive correlation, when r approaches -1, it indicates a strong negative correlation, and when r is near zero, it indicates no linear relation between X and Y. The correlation coefficient r is usually referred to as the *product-moment* correlation coefficient, to distinguish it from the rank order correlation coefficient which will be discussed in the next section.

12.3 Rank Order Correlation

In this analysis, X and Y are the ranks of n objects, which are assigned by two groups of "judges" and treated as discrete variables. The point is to determine whether the two ranks are related or show any agreement.

It can be shown in this case that

$$\sum_{j=1}^{j=n} x_j^2 = \sum_{j=1}^{j=n} y_j^2 = \frac{1}{6}n(n+1)(2n+1)$$

$$\sum_{j=1}^{j=n}(x_j-\bar{x})^2 = \sum_{j=1}^{j=n}(y_j-\bar{y})^2 = \frac{n(n^2-1)}{12}$$

and the correlation coefficient between the ranks X and Y is:

$$r_s = 1 - \frac{6\sum_{i=1}^{i=n} d_i^2}{n(n^2-1)}$$

where

$$\sum_{i=1}^{i=n} d_i^2 = \sum_{i=1}^{i=n}(x_i-y_i)^2 = \frac{1}{3}n(n+1)(2n+1) - 2\sum_{i=1}^{i=n} x_i y_i$$

The r_s then shows how well the ranks X and Y are correlated.

Product-moment correlation coefficients and rank order correlation coefficients can be further tested to determine their significance.

12.4 Tests of Significance

The word "significance" denotes how strong the evidence we have against a null hypothesis about some observations (correlation coefficients, comparison of values, *etc.*). *Test of significance* means testing a hypothesis for the significance.

12.4.1 Hypothesis and Alternative Hypothesis

A statistical hypothesis is usually a statement about a statistical population or the values of some parameters of the population [Ostle and Malone, 1988]. To reject or accept a hypothesis, we must conduct an experiment based on an underlying model and observe the outcomes of the result. To perform the experiment, a basic hypothesis to be tested must be first specified, which is called the *null hypothesis.* The *null hypothesis* is the basis for specifying different parameters of the model in the testing. Another hypothesis should also be set up, called *alternative hypothesis,* which states what will be accepted if the *null hypothesis* is rejected. Both *null hypothesis* and *alternative hypothesis* must be specified before the experiment.

12.4.2 Testing A Hypothesis

In addition to the specifications of the ***null hypothesis*** and ***alternative hypothesis***, a statistic is needed to be identified, called the ***test statistic***, the value of which will be used to decide whether to accept or reject the ***null hypothesis***. The ***test statistic*** is related to the sample size and the testing can be performed based on the decision-making rules which can be a subjective matter to some extent. A region in which the outcomes of the experimental results (values of the *test statistic*) under the specified model are unlikely is called *critical region*, for which we will reject the *null hypothesis*, because the *null hypothesis* can not give a reasonable explanation of the observed experimental results. If the observed values of the *test statistic* is not in the critical region, we do not reject the hypothesis. When the critical region contains the values at both ends of the possible range of the *test statistic*, we call it a *two-tail test* of the null hypothesis. Figure 49 gives the graphical illustration of a possible rejection region. The total probability of all the outcomes of the *test statistic* falling in the critical region is defined as the *significance level*, and an observation (outcome) contained in the critical region is *significant at that level*, which provides *significance evidence to reject the null hypothesis, in favour of the alternative hypothesis.*

The following steps summarize the test procedure:

1. Specify a *null hypothesis.*

2. Specify an *alternative hypothesis.*

3. Identify the underlying model and *test statistic.*

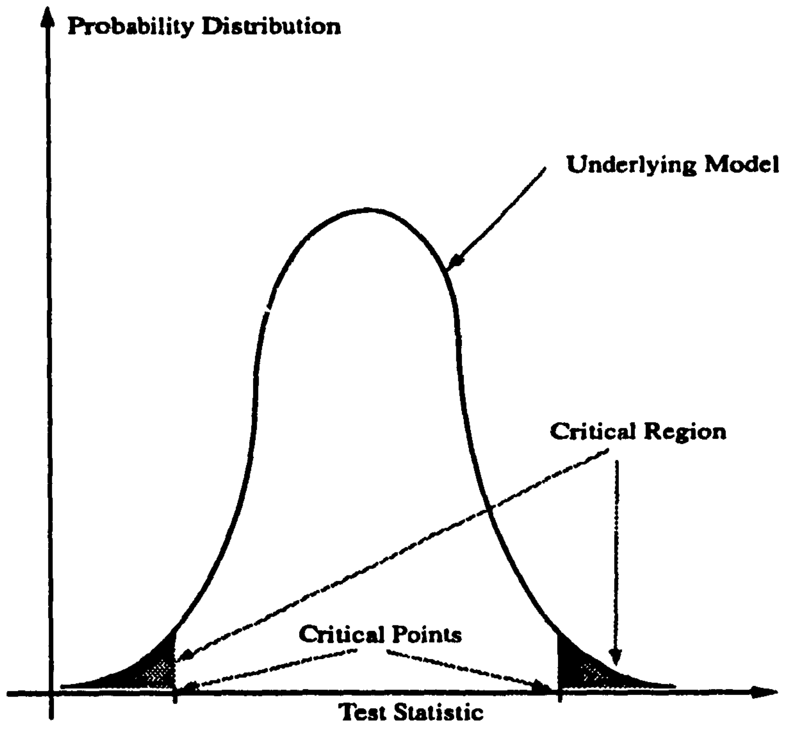


Figure 49: Graphical Illustration of a Critical Region

4. State a decision rule, i.e., all possible values of the *test statistic* must be assigned to the established critical region or the rest region.

5. Obtain the outcome (value) of the *test statistic* that will allow us to validate the hypotheses based on the weighting of the probability.

Usually, the hypothesis that (1) permits precise specification of the probabilities of all outcomes and parameters of the experiment (model), and (2) expresses scepticism of the claim being made or of the result we would like to be true, is assumed as the *null hypothesis.*

In testing, when the probability for a statistic based on a sample is unknown, tests of significance are commonly made by transferring that into another statistic for which the probability is known under the *null hypothesis.* The t distribution and standard normal distribution Z are commonly used, into which other statistics are converted. So, the experimental underlying models are the Z and t, and the test statistics are the Z and t values. The critical region is

$$|t| \geq t_o$$

or

$$|Z| \geq Z_o$$

where t_o or Z_o are the critical points that lie on the boundary points between the critical region and the rest region. Their values are calculated from the problems or specified critical regions. The probability of the test statistic falling into the critical

	True Situation	
Decision	Null Hypothesis is true	Null Hypothesis is false
Accept the null hypothesis	No error	Type II er ror
Reject the null hypothesis	Type I error	No error

Table 20: Types of Error Associated with Tests of Null Hypothesis

region, that is, the significant level, can be obtained by calculating the area under the critical region using the t or Z distribution function. By computing the outcome of the *test statistics* and observing if the value (outcome) falls into the critical region, we can hence judge the hypotheses. Figure 50 shows that testing procedure based on the transformations.

12.4.3 Two Types of Error

When we make a decision to accept or reject a hypothesis based on the outcome of the test statistic, it is possible to commit two types of error. A Type I error occurs when a null hypothesis is in fact true, but the value of test statistic leads us to reject the null hypothesis. The probability of Type I error is exactly the probability of all the outcomes contained in the critical region. Type II occurs when the null hypothesis is in fact false, but the value of test statistic does not make us reject the null hypothesis. Table 20 summarizes the types of error.

12.4.4 Tests of Significance for the Correlation Coefficients

Having calculated the correlation coefficient r between variables X and Y, we need to examine the value of r. If r is very close to 0, we can be satisfied that no relation exists between X and Y. On the other hand, if r is close to ± 1, we are happy to

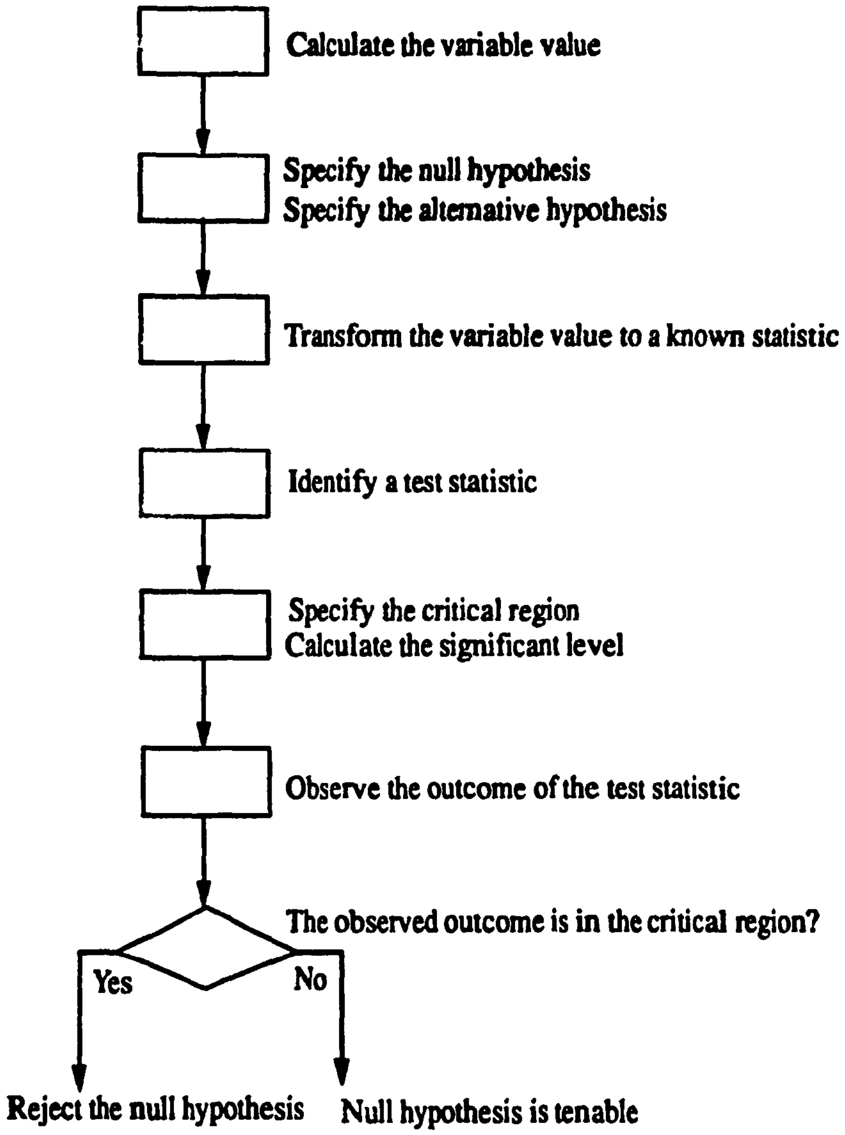

Calculate the variable value
Specify the null hypothesis
Specify the alternative hypothesis
Transform the variable value to a known statistic
Identify a test statistic
Specify the critical region
Calculate the significant level
Observe the outcome of the test statistic
The observed outcome is in the critical region?
Yes
No
Reject the null hypothesis
Null hypothesis is tenable

Figure 50: Transformation-Based Test Procedure for Significance

claim that there is a correlation. But, if the value of r is intermediate, say 0.350, we then need an objective way to assess and test the variable r.

It has been shown that when the sample size n is 10 or more, a function of r,

$$t = r\sqrt{\frac{n-2}{1-r^2}} \qquad (6)$$

under the null hypothesis that no relation exists between X and Y, can be approximated by the t-distribution with $(n-2)$ degrees of freedom [Clarke and Cooke, 1983, Edwards, 1976]. Hence, based on formula (6), the outcome of the test can be obtained by substituting the r value, and a two-tailed test on r can be performed (under the experimental model). Table 29 in the Appendix lists different t values with the degree of freedom at various significance levels. By using this table and the calculated outcome of t that corresponds to r from formula (6), the significance level where r lies can be determined. Hence, the decision of rejection or acceptance of the hypothesis can be made. By substituting the various values of n and the tabled values of t at different significance levels to solve for the value of r in the formula (6), a Table about the r values can be given, which is Table 30 in the Appendix. So, the calculated r value can be directly compared with this Table, and the significance levels can be determined.

It should be noted that the t distribution is affected by the sample size (degrees of freedom). If r is based on a small number of observations, even a relatively large observed value of r may not be significant enough to reject the null hypothesis from the calculated probability (significant level). On the other hand, when n is increased, a relatively small value of r may be significant enough to reject a null hypothesis.

12.4.5 Test of Significance of the Difference Between Two Coefficients

Another question about the correlation coefficients is the significance of the difference between two values of obtained correlation coefficients, r_1 and r_2. It is needed to investigate if the values r_1, say 0.875, and r_2, say 0.756, are significantly different, or they may come from a common population and are both the estimates of correlation coefficient of the same population.

There is a Z transformation which can be applied to transfer any value of correlation coefficient to a new variable, Z, that is,

$$Z = \frac{1}{2}[\log_e(1 + r_i) - \log_e(1 - r_i)]$$

where, r_i is some observed value of the correlation coefficient.

The expression of Z is approximately normal distributed with variance

$$\sigma_z^2 = 1/(n-3)$$

[n is the sample size, i.e., the number of pairs of observations from which r_i was calculated].

To make the test of significance of difference between the coefficients r_1 and r_2, we transform both r_1 and r_2 by Z transformation, say Z_1 and Z_2, and the variance of $Z_1 - Z_2$ is given by:

$$Var[Z_1 - Z_2] = Var[Z_1] + Var[Z_2] = \sigma_{z_1}^2 + \sigma_{z_2}^2$$

Then, the difference between Z_1 and Z_2 divided by the square root of $Var[Z_1 - Z_2]$ results in

$$Z = \frac{Z_1 - Z_2}{\sqrt{Var[Z_1 - Z_2]}} \tag{7}$$

Under the null hypothesis that r_1 and r_2 both are the estimate of the same population, expression (7) follows a standard normal distribution $\mathcal{N}(0,1)$. This model is then used as the experimental model from which the significance level is calculated. In this case, the $Z_1 - Z_2$ value which corresponds to $r_1 - r_2$ is the test statistic. The $\mathcal{N}(0,1)$ model can be further tabled. Table 31 shows the Z values and the cumulative probability for the area up to z, that is, $Z \leq z$, on which the probability for the area $|Z| \geq z$ can be calculated, which represents the significance level. If the significant level is quite small, the null hypothesis of no difference between r_1 and r_2 is rejected, the alternative hypothesis that r_1 and r_2 are significantly different is accepted.

12.5 Regression Analysis

In many applications, it is necessary to seek a way of expressing the functional relationship between two variables X and Y in terms of their their values x_i and y_i (i=1, 2 ......, n). This can be solved by applying regression analysis techniques.

A *regression model*, that denotes the functional relationship between X and Y, can be described as:

$$\hat{Y} = \varphi(X, \theta_0, ..., \theta_q)$$

where $\hat{Y}$ is the ***response variable*** which represents the estimation of Y, φ is a mathematical function that is usually called the ***regression function***, X is the ***independent variable*** or ***predictor***, and θ_0, ..., θ_q are q unknown parameters. An example of a regression model is the linear model which involves the linear estimation of Y based on X. It is postulated as:

$$\hat{Y} = \theta_1 X + \theta_0$$

The parameters θ_0 and θ_1 are estimated from observation data by applying the *least-squares estimates*, whose principle is to minimize the sum of the deviation between estimated values and and observed values of Y. The deviation indicates the residual of the model, and the square of the residual, that are to be minimized, can be expressed as

$$Q = \sum (y_i - \hat{y}_i)^2 = \sum (y_i - \theta_1 x_i - \theta_0)^2$$

where the x_i and y_i are the experimental observations of X and Y.

By minimizing the value of Q, we get:

$$\theta_1 = \frac{S_{xy}}{S_{xx}}$$

$$\theta_0 = \overline{y} - \theta_1 \overline{x}$$

where

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$\overline{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

$$S_{xx} = \sum_{i=1}^{n}(x_i - \overline{x})^2 = \sum_{i=1}^{n} x_i^2 - n\overline{x}^2$$

$$S_{xy} = \sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y}) = \sum_{i=1}^{n} x_i y_i - n\overline{x}\overline{y}$$

The estimated variances of θ_1 and θ_0 are:

$$S_{\theta_1}^2 = S_E^2 / \sum_{i=1}^{n}(x_i - \overline{x})^2$$

$$S_{\theta_0}^2 = S_E^2[(1/n) + \overline{x}^2 / \sum_{i=1}^{n}(x_i - \overline{x})^2]$$

where

$$S_E^2 = (S_{yy} - \theta_1 S_{xy})/(n-2)$$

$$S_{yy} = \sum_{i=1} n(y_i - \overline{y})^2$$

It has been shown that minimizing Q results in a sample regression line that best fits the observational data.

The above discussion provides the basis for the next chapter in which X and Y are replaced by the different correlation coefficients and other parameters.

Chapter 13

Evaluation of Expert System Metrics

13.1 Introduction

Evaluation is an important and necessary step for accepting and using existing metrics. The questions to be addressed are: (1) the effectiveness and quality of the metrics; and (2) the selection of appropriate metric(s), in the sense that they are all used as *indicators* or *predictors* of some ES characteristics for certain applications.

In this chapter, we are going to investigate the validity and performance of the metrics, that is, the extent to which a metric in fact assesses or predicts the quality characteristics [Itzfeldt, 1990]. So far, there are no established criteria for validating a software metric, and the most often used approach is by comparing a metric of interest with

1. Other metrics measuring the same characteristics.

   The internal consistency among the metrics is validated, that is, if different metrics are designed to measure the same characteristics, then they must be consistent. The evaluation is based on the assertion that the closer the metric correlates with other metrics, the more they are internally consistent with respect to measuring the same quality characteristics, if the definitions of the different metrics are based on different considerations (viewpoints) which are reasonable and necessary. Also, the overall high correlation of a metric with all other metrics indicates that the formulation of that metric reflects those considerations made by other metrics, so it is more sound and well-defined.

2. Some phenomena affected by the quality factors and criteria to be measured by the metrics. The empirical quality data, such as the anomaly rate, could be used as the phenomena. The evaluation basis is that the closer the metrics correlates with the phenomena, the more valid they are with respect to the quality characteristics affecting the phenomena. In addition, the high correlation of a metric with the phenomena shows that the metric performs well when used to assess or predict the quality characteristics.

Therefore, the metrics are tested by evaluating their (a) validity and (b) performance. The former indicates if a metrics is appropriate as the measures on some quality characteristics, that is, if it is well formulated for measuring the characteristics, like complexity. The latter shows which metric is most valid in assessing or predicting the quality characteristics.

The above analytical approach represents the empirical evaluation of metrics. For this type of evaluation, the statistical analysis and testing is the technique most frequently applied. It should also be noted that the results of the empirical evaluation are meaningful only under the circumstances that all the metrics are used to reflect the same characteristic of the quality. In addition to this empirical evaluation, metrics can also be evaluated in an abstract way (theoretical ground), that is, they can be evaluated against some general criteria and properties that produce the desired and expected performance of the metrics. The following sections detail the evaluation techniques presented in this thesis.

13.2 Empirical Evaluation

One application of the ES metrics is to use them as indicators or predictors of some ES characteristics, among them is the complexity, that is, their values are all used to represent the measures of some ES characteristics. We then need to evaluate and compare them for their validity. The techniques presented in the last Chapter are applied to the analysis.

13.2.1 Intercorrelations of Metrics

Here, the metrics are compared with each other by plotting each pair of metrics and calculating their correlation coefficients. If all these metrics measure some common ES characteristics, their measuring values should show a positive correlation among them. The scatterplot diagrams of the metric pairs are presented in Figures 51 to 71, and the calculated correlation coefficients are listed in Table 21. Moreover, the significance of these coefficients is tested by using two-tailed test under the t distribution. The null hypothesis is that no correlation exists between a pair of metrics, and the alternative hypothesis is that there is a positive relation between a pair of metrics. Table 22 shows the corresponding significance levels for the coefficients between each pair of metrics.

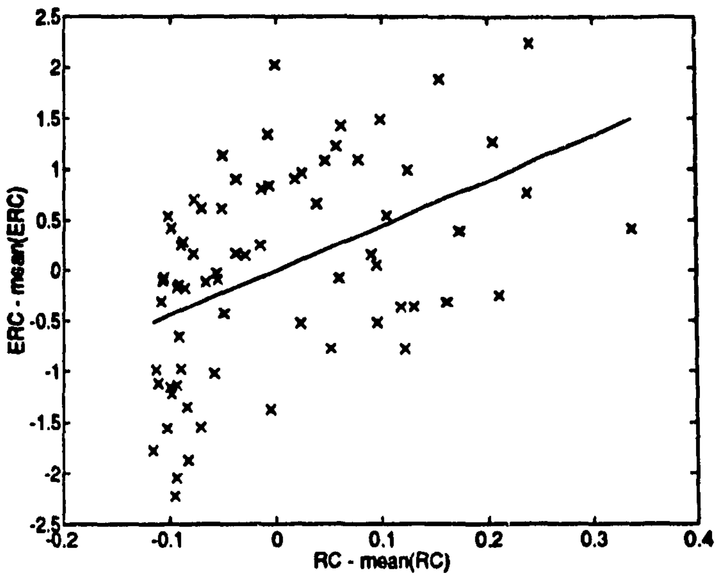


Figure 51: *RC* versus *ERC*

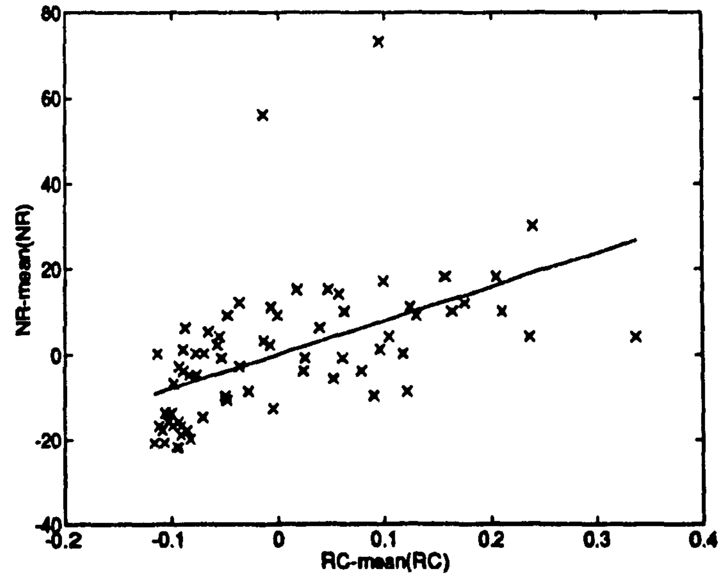


Figure 52: *RC* versus *NR*

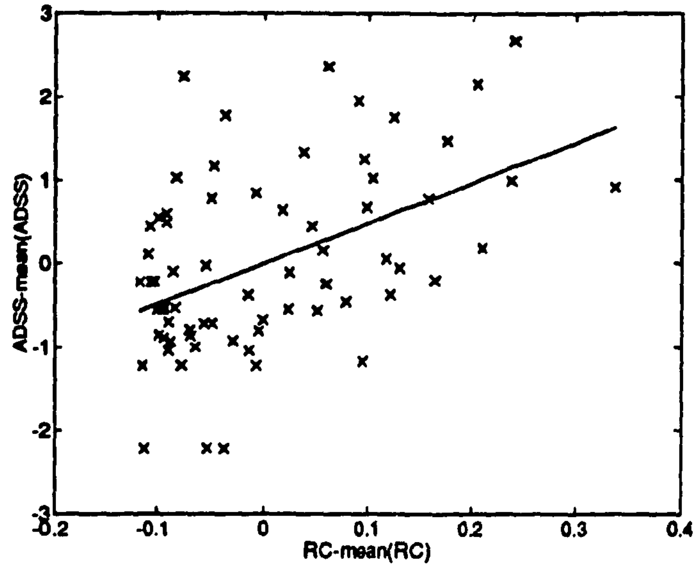


Figure 53: *RC* versus *ADSS*

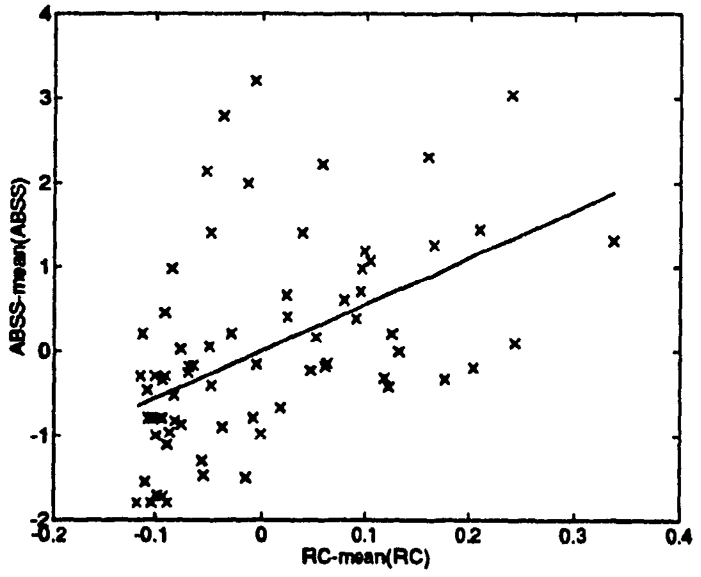


Figure 54: *RC* versus *ABSS*

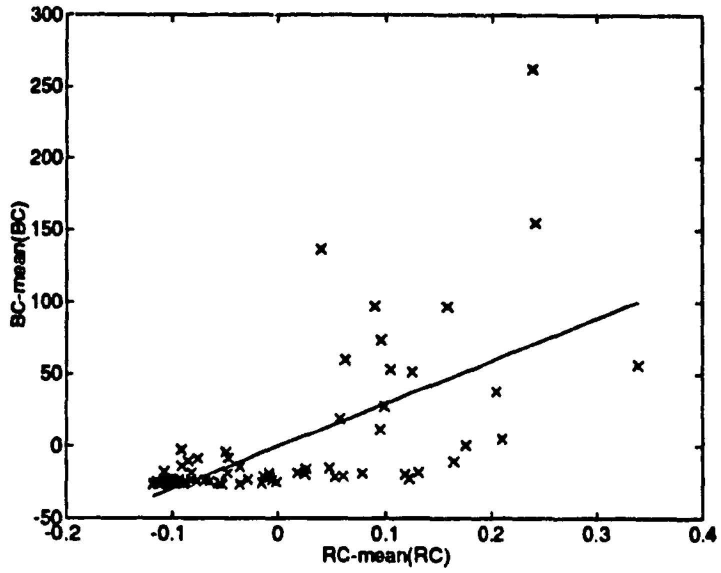


Figure 55: *RC* versus *BC*

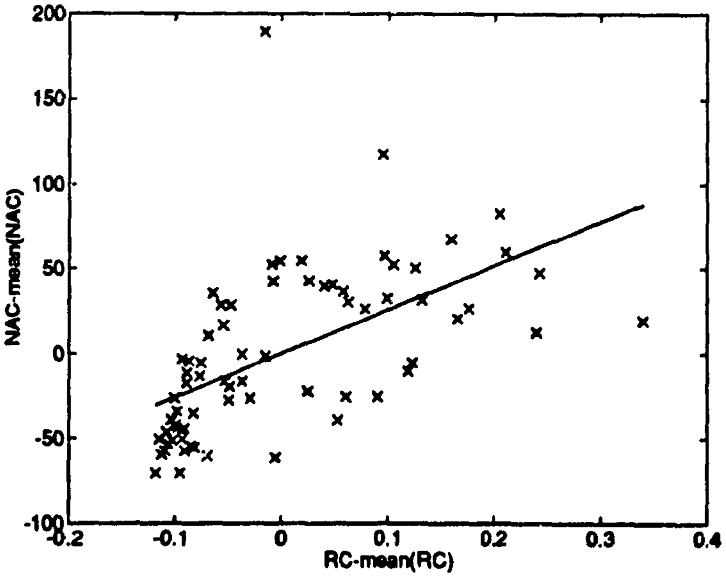


Figure 56: *RC* versus *NAC*

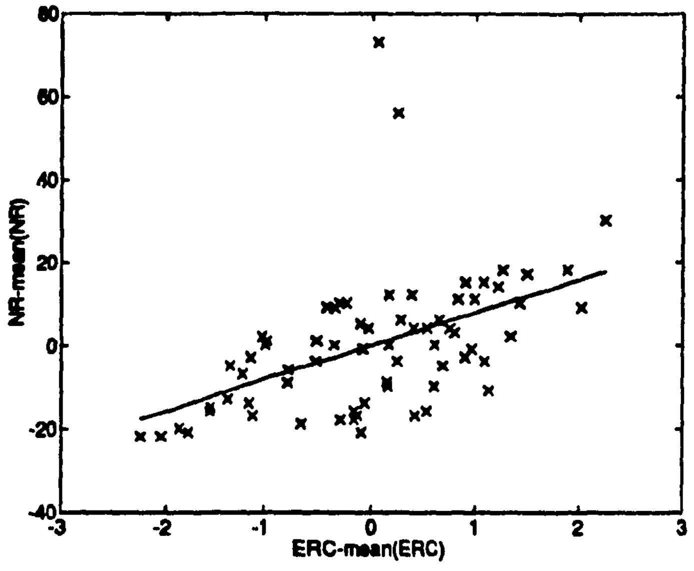


Figure 57: *ERC* versus *NR*

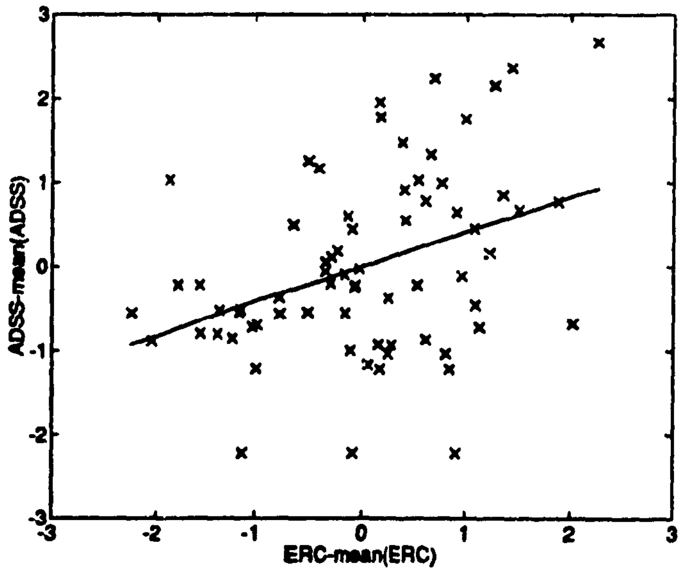


Figure 58: *ERC* versus *ADSS*

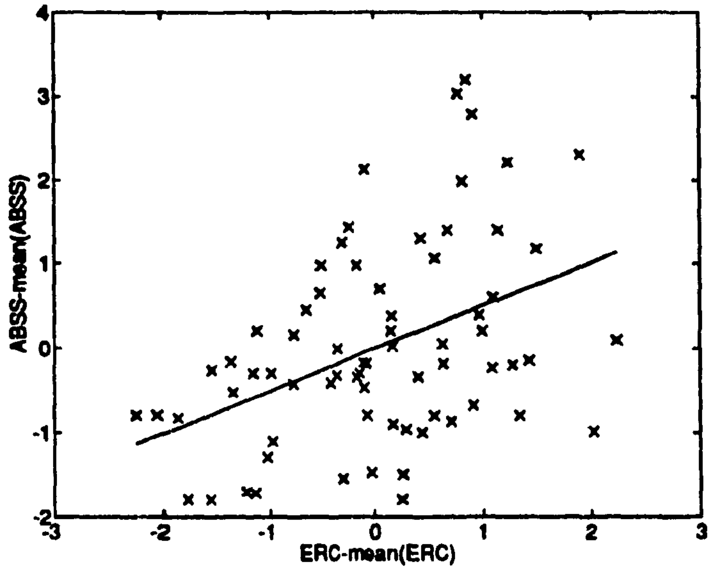


Figure 59: *ERC* versus *ABSS*

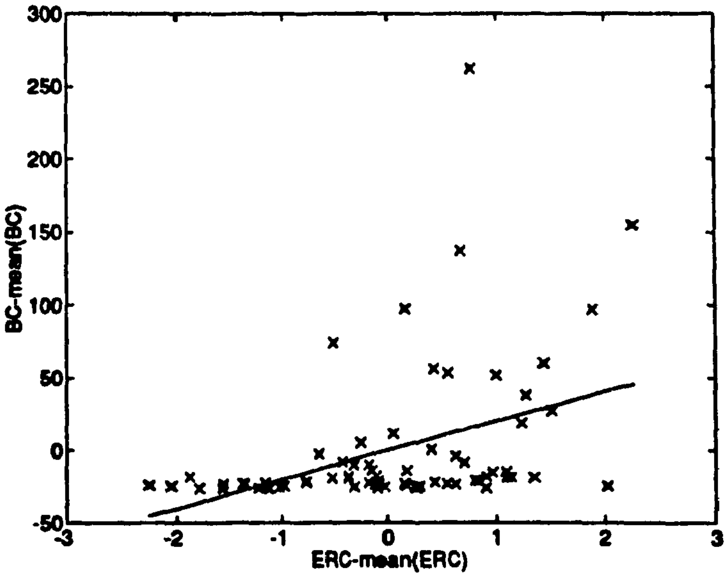


Figure 60: *ERC* versus *BC*

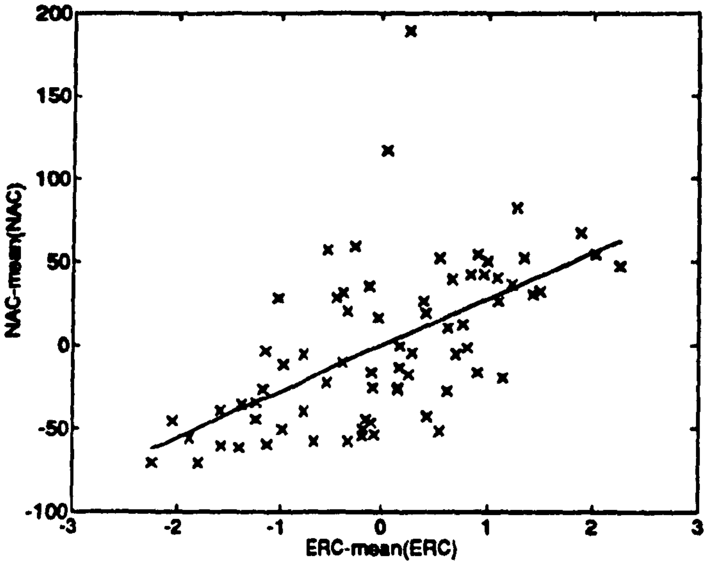


Figure 61: *ERC* versus *NAC*

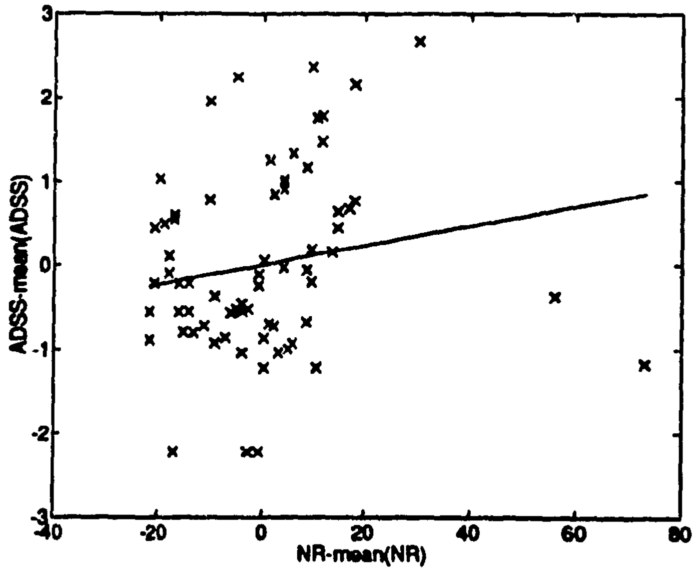


Figure 62: *NR* versus *ADSS*

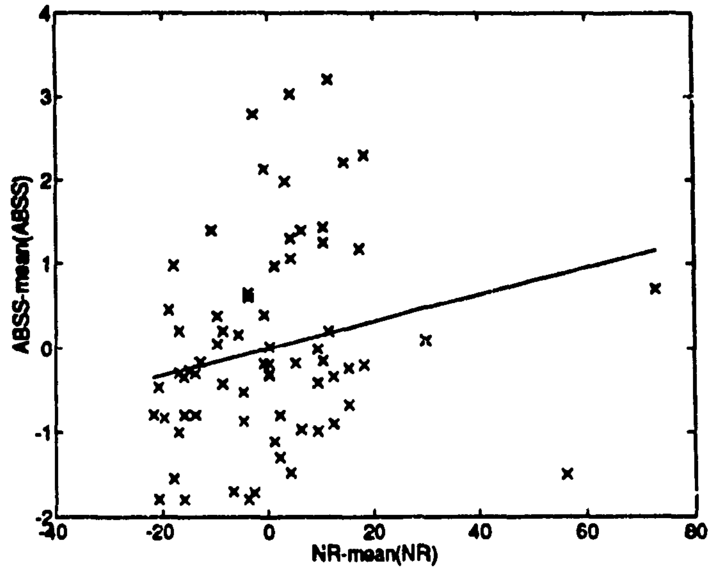


Figure 63: NR versus $ABSS$

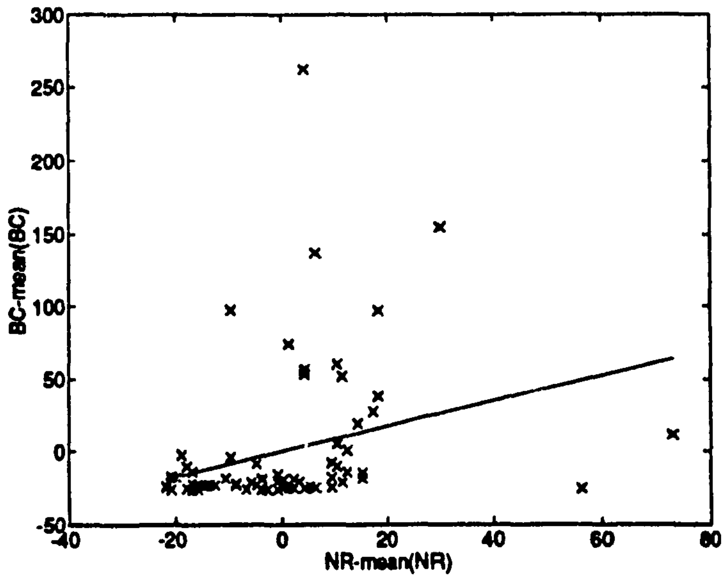


Figure 64: NR versus BC

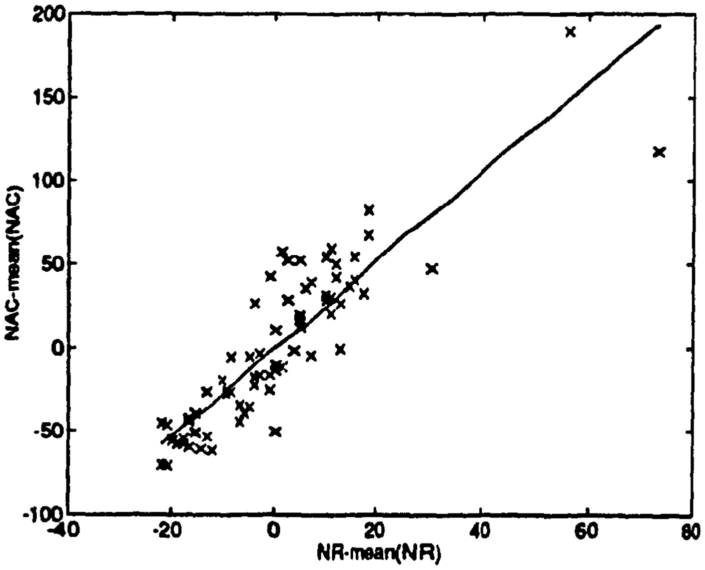


Figure 65: NR versus NAC

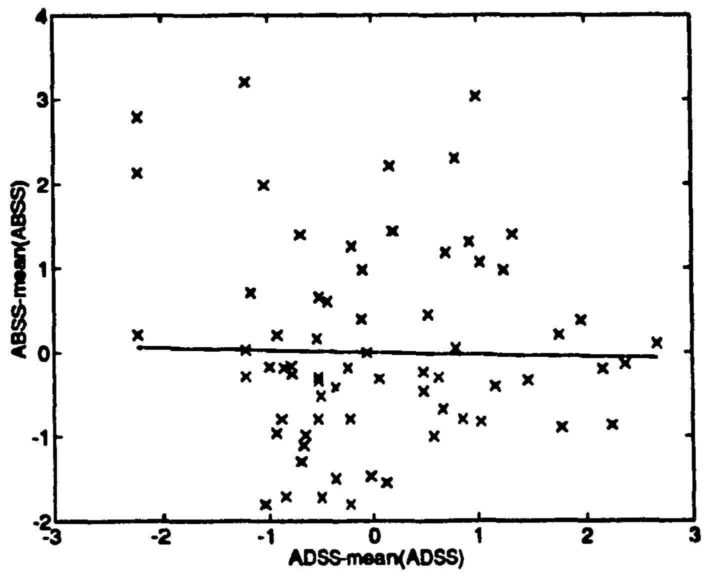


Figure 66: *ADSS* versus *ABSS*

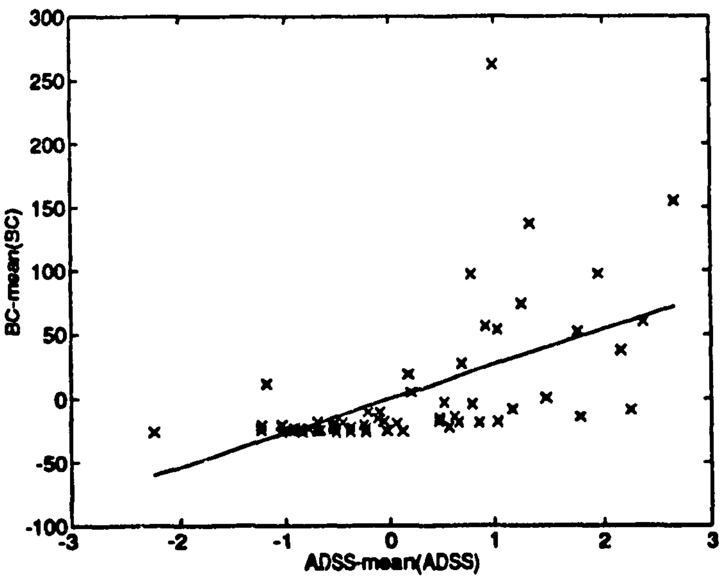


Figure 67: *ADSS* versus *BC*

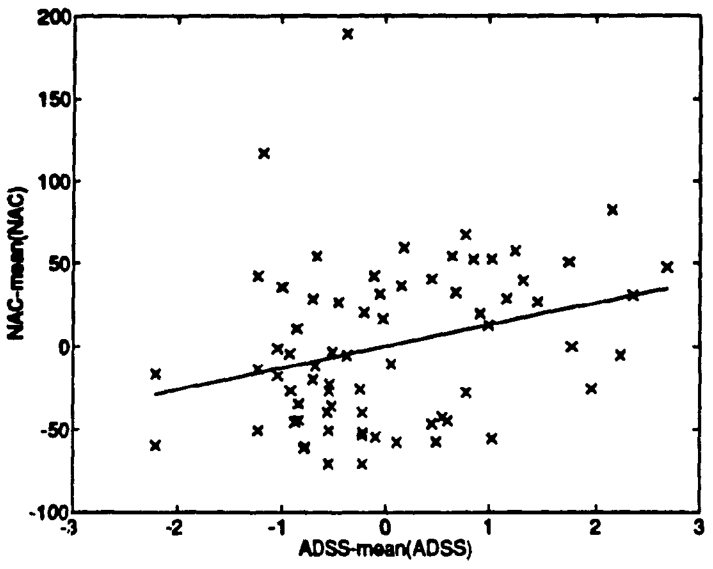


Figure 68: *ADSS* versus *NAC*

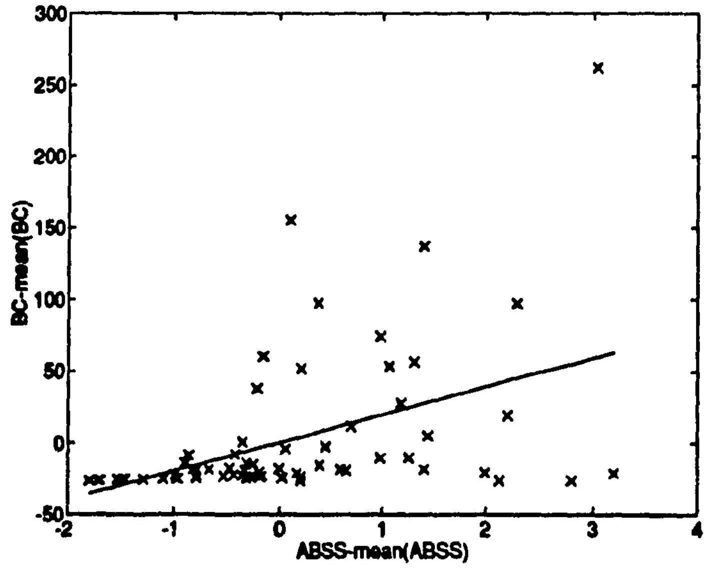


Figure 69: *ABSS* versus *BC*

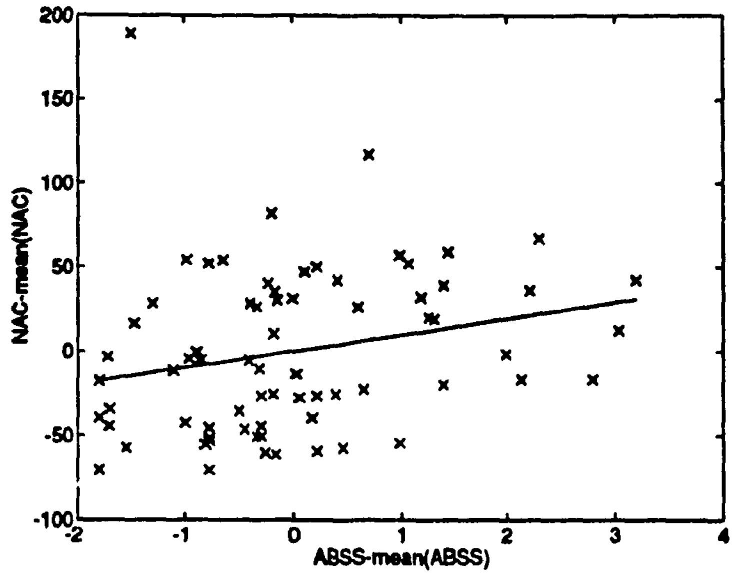


Figure 70: *ABSS* versus *NAC*

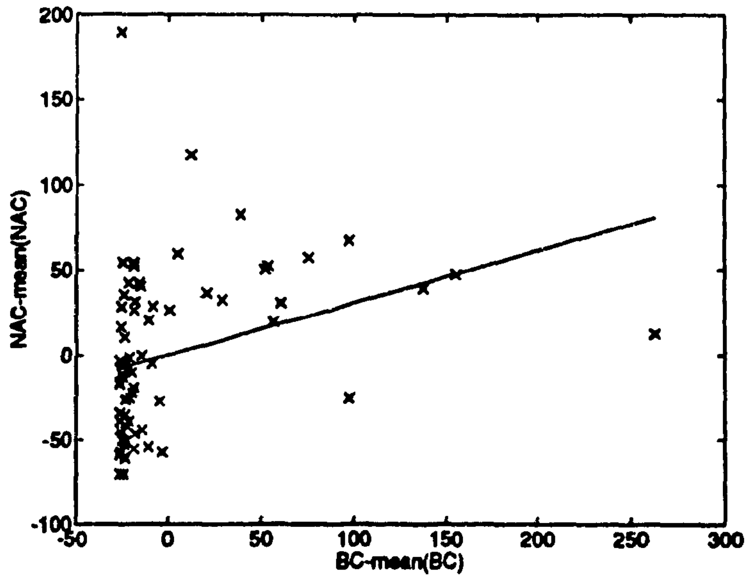


Figure 71: *BC* versus *NAC*

From the above evaluation results, we can see

- *RC* metric has the best overall correlation with all other metrics which are based on different viewpoints. Hence, it means that the formulation of *RC* reflects the different considerations or information of other metrics. So, this indicates that our proposed *RC* metric is well-defined and can be used as an appropriate indicator or predictor of some ES characteristics, like complexity.

- *NR* highly correlates with *NAC*, this is attributed to the nature of their formulations, that is, they both measure the length of rule bases. The very high correlation also implies that they can be replaced by each other in some situations. In this sense, it is redundant to have both metrics in one application.

- The null hypothesis can be rejected for all pairs of metrics, except three: (*ADSS*, *ABSS*), (*NR*, *ADSS*) and (*NR*, *ABSS*). This means that all metric pairs except these three are correlated, i.e. every metric is correlated or consistent with all or most of other metrics, this explains why sometimes they, to some extent, were in fact used to measure the same ES quality characteristic, say complexity. However, our further evaluation of the metrics, which will be

	ERC	*NR*	*ADSS*	*ABSS*	*BC*	*NAC*
RC	0.4701	0.5231	0.4789	0.4987	0.6338	0.5726
ERC		0.4905	0.3867	0.4264	0.4066	0.5769
NR			0.1760	0.2170	0.2823	0.8843
ADSS				-0.0229	0.5824	0.2864
ABSS					0.4677	0.2401
BC						0.3176

Table 21: Correlation Coefficients Among The Metrics

	ERC	*NR*	*ADSS*	*ABSS*	*BC*	*NAC*
RC	0.001	0.001	0.001	0.001	0.001	0.001
ERC		0.001	0.010	0.001	0.001	0.001
NR			0.100	0.100	0.050	0.001
ADSS				0.700	0.001	0.050
ABSS					0.001	0.050
BC						0.010

Table 22: Significance Levels for Each Coefficient

presented in a later section, shows that their performance can be quite different.

13.2.2 Correlations of Metrics with Anomaly Rates

One phenomenon reflecting ES quality is the number of anomalies (anomaly rate (AR)). The presented metrics are further validated by examining the relationship between these metrics and the anomaly rates.

Our basic hypothesis is that a rule base with a higher metric value (*RC*, *ERC*, *NR*, *ADSS*, *ABSS*, *BC*, *NAC*), would usually have a higher anomaly rate than a rule base with a lower metric value. Based on this assumption, it is expected that all the metric measuring results would exhibit this characteristic. Therefore, the correlation analysis technique was applied to examine this relationship. Figures 72, 73, 74, 75, 76, 77 and 78 illustrated such a relationship. They indicate that there are indeed positive relations between these metrics and the anomaly rate. A linear model may be appropriate for describing such relations. Furthermore, to quantitatively describe the relationship, the *correlation coefficients* between the measuring results and anomaly rates are calculated. The significance of these coefficients is tested by using a two-tailed test with the null hypothesis that no relation exists between the metrics and

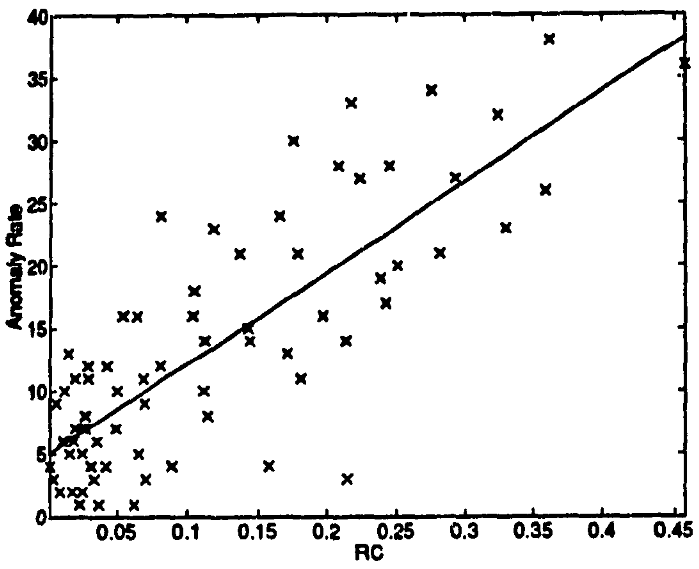


Figure 72: *RC* versus *AR*

the anomaly rate. The results (Table 23) show that there is sufficient evidence for us to reject the null hypothesis in favour of an alternative hypothesis that there is a correlation.

Another method of validating the metrics is based on the analysis of rank order comparison. The metric measuring results, as the ***estimators*** or ***indicators*** of the

Metric Pairs	Correlation Coefficients	Significance Levels
(*RC*, *AR*)	0.8026	0.001
(*ERC*, *AR*)	0.6003	0.001
(*NR*, *AR*)	0.5294	0.001
(*ADSS*, *AR*)	0.4553	0.001
(*ABSS*, *AR*)	0.3574	0.010
(*BC*, *AR*)	0.5798	0.001
(*NAC*, *AR*)	0.5344	0.001

Table 23: Correlation Coefficients Between the Metrics and the Anomaly Rate

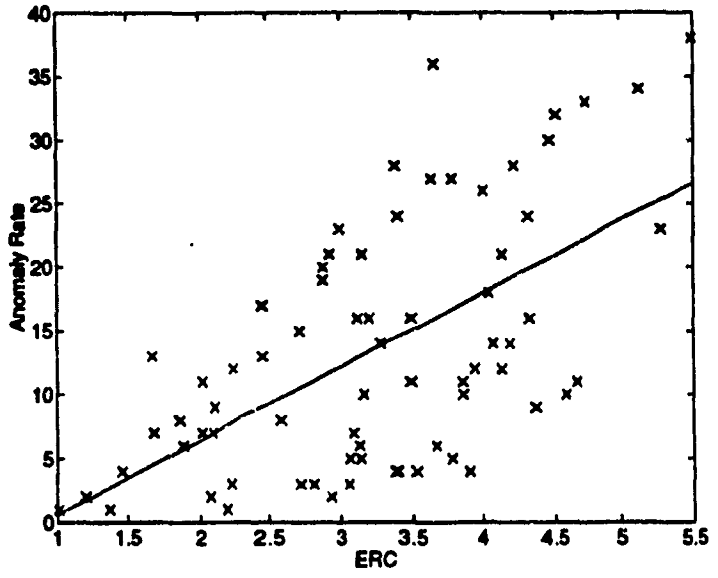


Figure 73: *ERC* versus *AR*

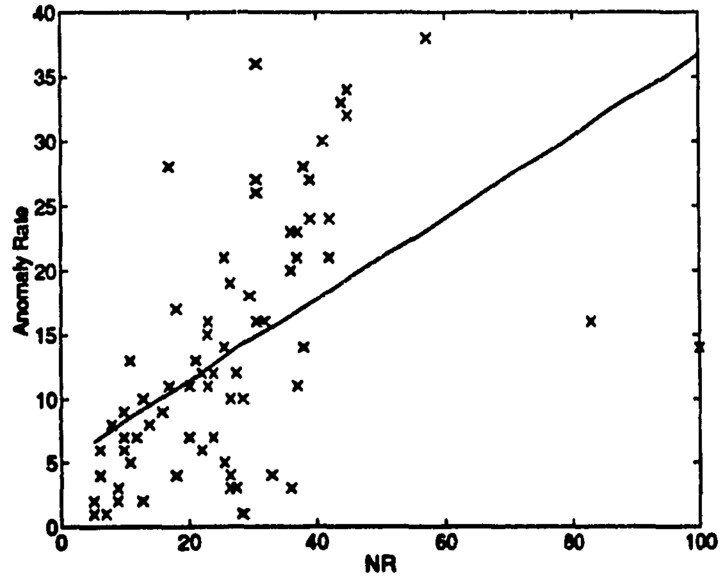


Figure 74: *NR* versus *AR*

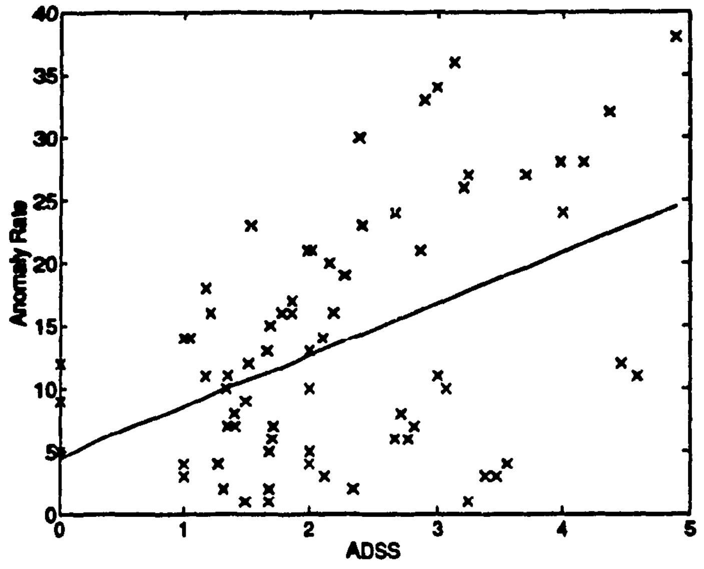


Figure 75: *ADSS* versus *AR*

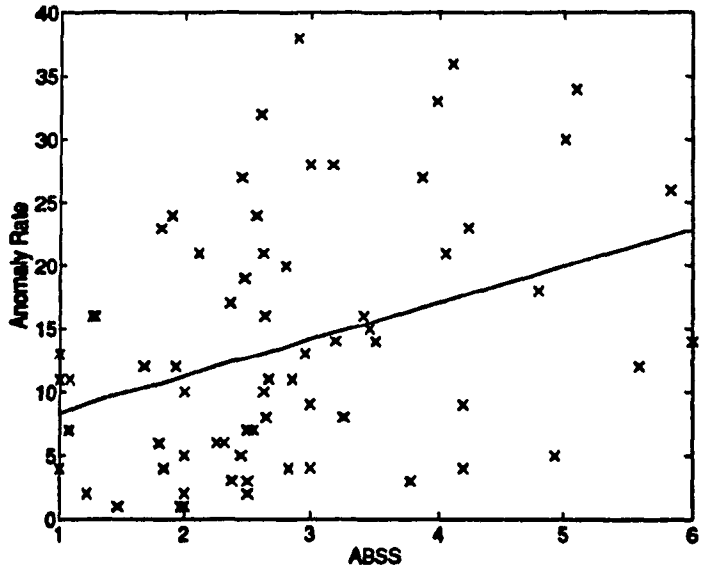


Figure 76: *ABSS* versus *AR*

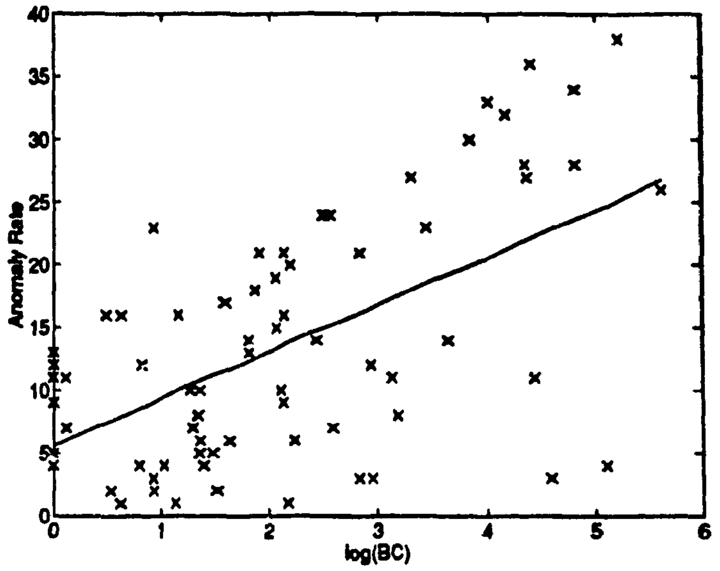


Figure 77: $log(BC)$ versus AR

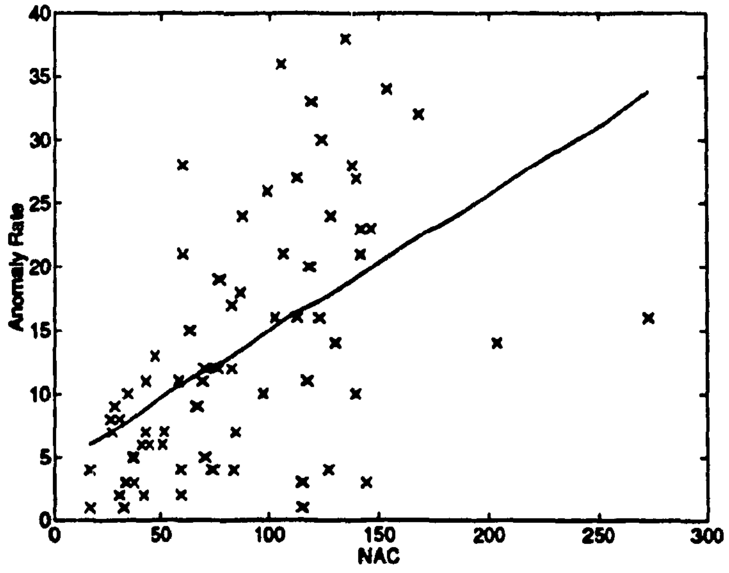


Figure 78: NAC versus AR

Metric Pairs	Rank Correlation Coefficients	Significance Levels
(RC, AR)	0.7422	0.001
(ERC, AR)	0.5703	0.001
(NR, AR)	0.6530	0.001
$(ADSS, AR)$	0.4647	0.001
$(ABSS, AR)$	0.3485	0.010
(BC, AR)	0.4748	0.001
(NAC, AR)	0.6030	0.001

Table 24: Rank Correlation Coefficients Between the Metrics and the Anomaly Rate

ES's quality characteristics, are used as one "judge" to give the ordinal rank of the ES. This is treated as one variate and is tested with another variate (rank) given by the anomaly rates (another "judge") with the null hypothesis that the two sets of ranks are independent of each other. Suppose n rule bases are to be ranked: x_i is the ranking in terms of the metric measuring results, y_i the ranking according to the anomaly rates. Then, the *correlation coefficient* r_S between the ranks x_i and y_i can be calculated. Also, based on the t transformation, and the null hypothesis, a two-tailed test on r_s can be performed and the significance levels of r_s can be determined. Table 24 gives the r_s values and the corresponding significance levels, which leads us to reject the null hypothesis and accept the alternative hypothesis that there exists a relationship between x_i and y_i. Since anomaly rate relates to the quality of the ES, the above results show that all these metrics could be used for the estimation (assessment) of the ES quality.

13.3 Comparison of Metrics

Even though all the presented metrics could be used for assessing or predicting the ES quality characteristics, their performance would be quite different. Consider the case in which ali these metrics are used to act as the complexity measures. The problem then is: which metric is more reliable and precise? Since all the metrics are correlated with the anomaly rate, this leads us to the investigation and comparison of the values of the correlation coefficients between the metrics and anomaly rate.

The significance of the difference between two correlation coefficients r_1 and r_2 obtained from two different observations (correlations between the metrics and anomaly rate) can be tested by using the z_r transformation under the null hypothesis that no significant difference exists between r_1 and r_2, and the alternative hypothesis that one coefficient is higher than another one. Hence, one can compare the different metrics and select the one with the best performance, i.e. the highest correlation coefficients, for the purpose of assessment or prediction.

Comparative results appear in Table 25, in which *metric1* > *metric2* means that there is evidence against the null hypothesis. Hence there is a significant difference between *metric1* and *metric2*, and *metric1* has a much better correlation with the anomaly rate than *metric2*. Meanwhile *metric1* $\simeq$ *metric2* indicates that the null hypothesis is tenable and the difference between *metric1* and *metric2* is not sufficiently great, i.e., they may come from the same data population and have equal correlation with the anomaly rate. From these results, we can see that the metrics can be

Metric Comparison
$(RC > ERC)$
$(RC > NR)$
$(RC > ADSS)$
$(RC > ABSS)$
$(RC > BC)$
$(RC > NAC)$
$(ERC \simeq NR)$
$(ERC > ADSS)$
$(ERC > ABSS)$
$(ERC \simeq BC)$
$(ERC \simeq NAC)$
$(NR \simeq ADSS)$
$(NR > ABSS)$
$(NR \simeq BC)$
$(NR \simeq NAC)$
$(ADSS \simeq ABSS)$
$(ADSS \simeq BC)$
$(ADSS \simeq NAC)$
$(BC > ABSS)$
$(NAC > ABSS)$
$(NAC \simeq BC)$

Table 25: Results of the Metric Comparisons

ranked (from high to low) in the following sequence: RC, ERC, BC, NAC, NR, $ADSS$, $ABSS$. RC correlates best with AR. In other words, RC is the best candidate for the assessment of ES quality. This is consistent with the empirical analysis and study presented below.

- The high correlation with AR is an inherent property of the RC measure, determined by our consideration in its formulation, which takes into account the three important characteristics of ES: *content, size* and *search space*. Hence, RC can indeed act as a good means for assessing the ES quality.

- In some applications, the rule bases, especially those with a large number of rules, can be divided into different subsets of rules without any connection to each other. As a result, their complexity is reduced considerably, and it may not look as high as the NR and NAC may indicate. Simply counting the number of rules or the number of patterns in the antecedents and consequents of rules without consideration of their internal relations could give the wrong message and lead to a contradictory decision.

- The contents of the matching patterns in each rule are also an important aspect which should not be ignored when comparing different rule bases. However, $ABSS$, $ADSS$, BC, NR and NAC reflect the overall size of the rule base or the abstract search space only; they do not account for the contents.

- In fact, $ABSS$ and $ADSS$ measure only a portion of the search space; they are incomplete measures. So, their performance is even worse than that of the others.

Criteria	Meaningful	Reasonable	Reliable	Cost-effective
RC	Yes	Yes	Yes	Yes
ERC	Yes	No	No	Yes
NR	Yes	Yes	Yes	Yes
BC	Yes	No	Yes	Yes
ADSS	Yes	No	Yes	Yes
ABSS	Yes	No	Yes	Yes
NAC	Yes	Yes	No	Yes

Table 26: Metric Evaluation Against The Criteria

- As mentioned before, the *dependency* among the rules is an important characteristic of ES, which should not be ignored for all measures. However, *ERC*, *NAC* and *NR* do not consider this critical part, so their performance is affected by this deficiency.

The above different empirical evaluation results which are all based on the different methods (viewpoints) show that *RC* has the best performance as the *indicator* or *predictor* of ES quality characteristics.

13.4 Theoretical Evaluation

Metrics can also be evaluated in an abstract way. First, as a review, we will briefly evaluate the metrics based on the criteria defined in Chapter 6.

Table 26 shows the results of the evaluation.

In addition to the criteria, some general properties are also desired and expected regarding the performance and behavior of ES metrics. In software engineering, a set of abstract properties was already presented by Weyuker [Weyuker, 1988] for formally comparing and evaluating the conventional software metrics applied. The purpose for defining such general properties is to assess the suitability of existing metrics from a more subjective grcund. A similar but different set of properties can also be developed for the compar;son and evaluation of ES metrics. We propose the following properties to be used for ES me'.rics.

Suppose in the following discussions, the capital letters denote different ES, $\theta(\mathcal{R})$ represents the metric θ measured on an ES $\mathcal{R}$. Then we have:

Property 1: $\exists \mathcal{P}\, \exists \mathcal{R} \cdot (\theta(\mathcal{P}) \neq \theta(\mathcal{R}))$.

This property requires that the metric be able to scale and compare different ES.

Obviously, a measure must satisfy this property in order to be useful.

Property 2: $\forall \mathcal{P}\, \forall \mathcal{R} \cdot (\mathcal{P} \subset \mathcal{R} \rightarrow \theta(\mathcal{P}) < \theta(\mathcal{R}))$.

This property means that the measure on a sub-ES should be less than that of the ES as a whole.

In general, for the metrics whose values increase as the ES become more complicated, this property should stand. Th'~s kind of metrics includes size ~netrics, complexity metrics and search space metrics.

Property 3: For a nonnegative constant c, there are only a finite number of different

ES[1] with the measure c.

This property denotes that if there are infinite ES in which there exist some differences among them, that have the same value of the measure, then the measure is weak in the sense that it cannot distinguish these ES, that is, the measure is not sensitive to the differences.

Property 4: $\exists \mathcal{P}\ \exists \mathcal{R} \cdot (\mathcal{P} /= \mathcal{Q} \wedge \theta(\mathcal{P}) = \theta(\mathcal{R}))$.

"/= " means the unequal dependency structures of different ES.

This property indicates that ES with different structures may have the same measure.

Implication behind this property is that a sound metric should not consider only the rule dependency structure in an ES, different attributes such as the content of matching patterns should also be taken into account. This can also be seen from software engineering where the metrics that are purely defined in terms of control structure have been criticized to lack concerns about the amount of computations contained in a program.

Property 5: $\exists \mathcal{P}\ \exists \mathcal{R} \cdot (\mathcal{P} <> \mathcal{Q} \wedge \theta(\mathcal{P}) = \theta(\mathcal{R}))$.

"<> " means the unequal functions of different ES.

This property indicates that ES with different functions may have the same

[1]Two ES are considered to be the same if one is just a renaming of the second one.

measure.

Because of the different implementations of the ES functions, it is possible that in some cases, the measure on these implementations may have the same value, even though the functions that ES perform are different. This also shows that it is the detail of implementation that determines the measure.

Property 6: $\exists \mathcal{P}\, \exists \mathcal{R} \cdot (\mathcal{P} \equiv \mathcal{Q} \wedge \theta(\mathcal{P}) \neq \theta(\mathcal{R}))$.

" $\equiv$" means the ES with the same function.

This property indicates that ES having the same behavior may not have the same measure.

Still, this reflects the different implementations of functions from another point of view.

Property 7: $\exists \mathcal{P}\, \exists \mathcal{R} \cdot (\mathcal{P} \neq \mathcal{Q} \wedge \theta(\mathcal{P}) = \theta(\mathcal{R}) \wedge \exists \mathcal{Q} \cdot (\theta(\mathcal{P} \cup \mathcal{Q}) \neq \theta(\mathcal{R} \cup \mathcal{Q})))$

This property means that the conjunction of ES may have different effects.

Since an ES, more precisely, a rule base, may potentially have different interactions with the existing rule bases, the conjunctions of that rule base with the existing rule bases may change the measuring results differently, even though they have the same value for the measure before conjunctions. One extreme case is that P is just the renaming of R, and Q has no item that is common with P, but Q contains several consequents that match some antecedents of R. A simple case is:

$$P: \; A1 \longrightarrow A2$$

$$R: \; B1 \longrightarrow B2$$

$$Q: \; C1 \longrightarrow B1$$

Property 8: If $\mathcal{P}$ is a renaming of $\mathcal{R}$, that is, if there exists a sequence $\mathcal{R}= P_1, P_2, \ldots\ldots,$ $P_n = \mathcal{P}$, P_i is obtained by replacing all the instances of an identifier x in P_{i-1} by y where y does not appear in $\mathcal{P}_{i-1}$, then for the metric θ, which is based on the syntactic structures of ES, $\theta(\mathcal{P}) = \theta(\mathcal{R})$.

This property points out that, in general, renaming will not change the measure.

Property 9: $\exists\mathcal{P}\ \exists\mathcal{R} \cdot (\theta(\mathcal{P}) + \theta(\mathcal{R}) \geq \theta(\mathcal{P} \cup \mathcal{R}))$

This property asserts that, in some cases, the conjunction of ES decreases the measure.

Due to the unknown contents in different rule bases, it is reasonable to think that in some situations the conjunction or combination of two rule bases will not increase the value for a measure. Consider the case where two rule bases are identical, then the conjunction of them actually increases nothing but some redundant rules.

Property 10: $\exists\mathcal{P}\,\exists\mathcal{R}\cdot(\theta(\mathcal{P})+\theta(\mathcal{R})<\theta(\mathcal{P}\cup\mathcal{R}))$

This property asserts that, in some cases, the conjunction of ES increases the measures.

Clearly, in some cases, the conjunction of rule bases, however, will increase the value for a measure, because of the addition of extra interaction between them.

Property 11: $\exists\mathcal{P}\,\exists\mathcal{R}\cdot(\mathcal{P}\Leftrightarrow\mathcal{R}\wedge\theta(\mathcal{P})\neq\theta(\mathcal{R}))$

"$\Leftrightarrow$" sign means the equal dependency graphs of the ES.

This property indicates that for some ES, even though they have the same dependency structures, the measure on them could produce different results.

The reason for having the above property is based on the considerations that in some cases, the types and contents of dependencies contained in rule bases may be different and these are not reflected in the dependency graph, however, the measure should be sensitive to these differences because of their importance. So, measuring results on them should be different.

Based on the properties introduced above, we evaluate the presented metrics. The results are listed in Table 27, from which we can see that the performance of the three metrics: BC, $ADSS$ and NAC are the same. This occurs because they are all based on the abstract search space and study the search paths in the space. So, they have the same effect. Also, NR and NAC, as measures of the size, play a similar role. The performance of ERC is a little better. All the above measures lack the complete

Property No.	1	2	3	4	5	6	7	8	9	10	11
RC	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
ERC	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
NR	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	No	No	No
BC	Yes	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
ADSS	Yes	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
ABSS	Yes	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
NAC	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	No	No	Yes

Table 27: Evaluation of Different Metrics

Metrics	Performance
RC	Good Performance
ERC, *BC*, *NR*, *NAC*	Moderate Performance
ADSS ABSS	Weak Performance

Table 28: The Metrics Grouped According to Their Performance

insight into the ES. This is why *RC* metric performs best among all these measures. *RC* takes into account both the ***search space*** and ***size***, as well as the ***content of matching patterns***. This result indicates again that the hybrid metric can usually give more reliable and meaningful results.

According to the evaluation results in this section, the presented metrics can be ranked as shown in Table 28. The evaluation results are consistent with those obtained from empirical evaluation and comparison as presented in sections 13.2 and 13.3.

13.5 Using Metrics as Predictors

One ultimate goal of our metric study is to formulate appropriate metrics to predict and to identify the various factors that have an effect on the quality of ES. From the above analysis, we have already shown that *RC* is most effective and has the best correlation with the anomaly rate which is synonymous with ES quality from narrower viewpoint, or some other quality characteristics. So, *RC* is useful in predicting the quality or some quality characteristics. To further quantitatively assess the relationship between the anomaly rate and *RC*, the regression analysis technique can be applied. The diagram plotted in Figure 72 and the high correlation coefficient in Table 23 indicate that a linear line (linear model) may fit well into the data points in the *RC* and *AR* relation plane. Based on the existing data, and the use of least squares method, we get the following equation:

$$AR = 72.677 * RC + 4.891 \tag{8}$$

where the estimated variances $(S^2_{\theta_1})$ and $S^2_{\theta_0})$ for the slope (72.677) and intercept (4.981) are

$$S^2_{\theta_1} = 42.276$$

$$S^2_{\theta_0} = 1.085$$

In formula (8), *RC* is an ***independent variable (predictor)*** which represents the measure and *AR* is a ***dependent variable.*** Equation (8) can be applied to similar ES to compare their quality. However, for different kinds of ES with various characteristics (different application domains, environments, *etc.*), the coefficients of the formula may be different, which will be reflected in the differences of the values of the slope and intercept of linear regression lines in the plane. Furthermore, to improve the accuracy of prediction using regression equations, the validity of applying them must be further assessed for different usages, since they are also affected by the characteristics of the applications. This study will be conducted in our future research.

Chapter 14

Concluding Remarks

Software metrics are valuable because they provide quantitative information for assessment and prediction. Some of their applications include the identification of unusual software, profile and predict software quality and characteristics. The practices of applying software metrics and their benefits can be seen from many successful Metrics Programs in different business organizations, where the quantitative metrics have been used as a powerful tool, e.g. (1) improving product quality; (2) increasing development team productivity; (3) better project planning and estimation; and (4) better project management and so on. So, metrics play a key role and are essential in software development.

One of the special types of software is ES whose applications have dramatically increased during the past years. However, due to the special features of ES such as the complex *dependency* among the rules, the behavior of such systems is sometimes hard to predict. This subjects ES to more quality problems and increases the risk of applying them. Hence, there is an urgent need to formulate certain measures on

ES. Like the metrics for conventional software, the development of ES metrics will definitely help to achieve the goal of assuring the ES quality.

The formulation of ES metrics involves several steps, one of them is the study of the interesting characteristics of ES. Designing ES metrics further requires the understanding of the essence of ES, especially the rule bases that possess many special characteristics which distinguish them from conventional software. This thesis aims to investigate the formal metrics for ES, which is currently lacking. Several problems are addressed, including the formal definition and description of ES metrics, and an evaluation of the presented metrics.

The basis for defining appropriate software metrics is provided by the measurement theory. So to make the metrics reasonable and meaningful, we have to meet some conditions required by the theory, such as *presentation condition* which preserves the empirical properties of the software in formal systems. The basic components of the measurement theory and its application to metric measurements are presented and addressed in this thesis with the aim to provide a rationale for metric study.

Since ES possess both the features that are common with conventional software, and the distinctive features of their own, it is reasonable to first conduct a comparative study between conventional software and ES, then (1) adapt the existing mature measuring approaches for conventional software to measure the common features, and (2) develop new techniques to measure those special features. For rule-based ES, the common features are the conventional *operators* and *operands*, so some widely accepted measures, such as the Halstead's software science can be used to measure them. The *rule* and *dependency* are the distinctive features, which require new formalism to be defined. To meet this end, this thesis presents the AND/OR digraph for formally describing these special features, and based on this AND/OR digraph and the contents of the rules, metric measuring methods are formally proposed and presented. Seven ES metrics are presented and examined in this thesis, including our proposed RC for the ES complexity measure. It should be noted that little effort has been devoted to this subject before, even though the complexity measures have been playing a critical and important role in quality evaluation of conventional software. The above ES metrics have been developed after taking different viewpoints into consideration, and they are the basic metrics in the quality model presented in this thesis, that is, their values form the basis for further assessing or predicting such quality characteristics as *maintainability, reliability* and *testability* which are important to ES. However, when the metrics are used as the *indicator* or *predictor* of some quality characteristics, it is possible that their performance may be quite different. Some are more effective than others. This leads us to the study of metric evaluation, a subject which is essential for accepting and using existing metrics. This thesis proposes several metric evaluation techniques based on statistical testing and analysis. These evaluation techniques are based on the empirical ground in the sense that the metric values are compared with either other metric values or some phenomena affected by ES quality characteristics. Metrics can also be validated from the theoretical ground, that is, in an abstract way. A set of general properties and criteria which represent the desired performance of the metrics are therefore proposed in this thesis. Metrics are further evaluated against them as well. The above evaluation techniques can help us to judiciously validate and select the existing metrics, and guide the design of new metrics that would be proposed in the future.

Assessing the validity of the presented metrics by using the above evaluation techniques reveals that *RC* is most effective and has the best performance among these metrics in evaluating ES. This indicates that *RC* metric, defined as a hybrid metric that takes into account three important characteristics of rule bases: the *matching contents, size* and *search space*, is well formulated, and it can be a useful tool to assess and predict the quality of ES.

This thesis also discusses and presents several important issues relating to the ES metrics. They include: the metrics organization, metrics factors, general methodology of metric formulation, automatic measuring tools, and so on. These issues deal with the different aspects of ES metrics, that should not be ignored.

Currently, this research is limited to the measures of the static characteristics of rule-based ES. Our future work may include:

- Extending the metrics to different kinds of expert systems, such as frame-based ES, semantic-based ES, and those which contain uncertainty;
- Investigating the dynamic characteristics of ES and their measurements;
- Comparing the measurement models against subjective expert opinions.

Similar to the development of conventional software, the development of a good quality ES needs the proper formulation and application of their metrics, and this research is a challenging area and yet it is seldomly addressed. We believe that the study conducted in this thesis is important and hope that it will lay a foundation for our future research to reach the long-term goal of building a wider spectrum of metrics for assessing and predicting the quality of products developed in knowledge engineering.

At last, it is worth mentioning that some results of this research have already been accepted for publications [Chen and Suen, 1993a, Chen and Suen, 1993b, Chen and Suen, 1994a, Chen and Suen, 1994b, Chen and Suen, 1994c, Chen et al., 1994d], where different issues concerning the quantitative assessment of ES are presented and discussed.

References

[Arthur, 1985] Arthur, L. J. (1985). *Measuring Programming Productivity and Software Quality.* John Wiley & Sons, New York.

[Barrett, 1990] Barrett, B. W. (1990). A software quality specification methodology for knowledge-based systems. In Culbert, C., editor, *AAAI-90 Workshop on Knowledge Based Systems Verification, Validation and Testing.* AAAI. Unpublished Workshop Notes.

[Basili, 1980] Basili, V. (1980). Tutorial on models and metrics for software management and engineering. In *Proc. of Fourth International Conference on Computer Software & Applications Conference*, pages 288–293.

[Behrendt et al., 1991] Behrendt, W., Lambert, S. C., Ringland, G. A., Hughes, P., and Poulter, K. (1991). Gateway: Metrics for knowledge based systems. In Liebowitz, J., editor, *Proceedings of the World Congress on Expert Systems*, volume 2, pages 1056–1067, New York. Pergamon Press.

[Buchanan, 1987] Buchanan, B. G. (1987). Artificial intelligence as an experimental science. Technical Report KSL 87-03, Knowledge Systems Laboratory, Stanford University, Stanford, CA.

[Card and Glass, 1981] Card, D. N. and Glass, R. L. (1981). *Measuring Software Design Quality.* Prentice-Hall, Englewood Cliffs, New Jersey 07632.

[Cardeñosa, 1995] Cardeñosa, J. (1995). VALID: An environment for validation of KBS. *Expert Systems with Applications*, 8(3). In press.

[Carrico et al., 1989] Carrico, M. A., Girard, J. E., and Jones, J. P. (1989). *Building Knowledge Systems.* McGraw-Hill Book Company, New York.

[Channon, 1974] Channon, R. N. (1974). *On A Measure of Program Structure.* Ph.D. Dissertation, Carnegie-Mellon University, Pittsburgh, PA.

[Chen and Suen, 1993a] Chen, Z. and Suen, C. Y. (1993). Application of metric measures: From conventional software to expert systems. In *Proc. of the AAAI Workshop on Verification, Validation and Testing of Knowledge-Based Systems*, pages 44–51.

[Chen and Suen, 1993b] Chen, Z. and Suen, C. Y. (1993). Evaluating expert systems by formal metrics. In *Proc. of the Canadian Conference on Electrical and Computer Engineering*, pages 763–766.

[Chen and Suen, 1994a] Chen, Z. and Suen, C. Y. (1994a). Measuring the complexity of rule-based expert systems. *Expert Systems with Applications*, 7(4). In press.

[Chen and Suen, 1994b] Chen, Z. and Suen, C. Y. (1994b). Complexity metrics for rule-based expert systems. In *Proc. of the IEEE International Conference on Software Maintenance.* In press.

[Chen et al., 1994c] Chen, Z., Grogono, P., and Suen, C. Y. (1994). Quantitative evaluation of expert systems. In *Proc. of the IEEE International Conference on Systems, Man and Cybernetics*. In press.

[Chen and Suen, 1994d] Chen, Z. and Suen, C. Y. (1994d). Metrics for assessing the quality of rule-based systems. *IEEE Trans. on Knowledge and Data Engineering*. Under review.

[Clarke and Cooke, 1983] Clarke, G. M. and Cooke, D. (1983). *A Basic Course in Statistics*. Edward Arnold, Caulfield East, Victoria.

[Deutsh and Willis, 1988] Deutsh, M. S. and Willis, R. R. (1988). *Software Quality Engineering*. Prentice Hall, Englewood Cliffs, NJ.

[Edwards, 1976] Edwards, A. L. (1976). *An Introduction to Linear Regression and Correlation*. W. H. Freeman and Company, San Francisco.

[Elshoff, 1976] Elshoff, J. (1976). An analysis of some commercial PL/1 programs. *IEEE Trans. on Software Engineering*, SE-2(5):113–120.

[Elshoff, 1984] Elshoff, J. L. (1984). Characteristics of program complexity measurement. In *Proc. of the Int. Conference on Software Engineering*, pages 288–293.

[Fenton, 1991] Fenton, N. E. (1991). *Software Metrics: A Rigorous Approach*. Chapman & Hall, London.

[Forsyth, 1984] Forsyth, R. (1984). The architecture of expert systems. In Forsyth, R., editor, *Expert Systems: Principles and Case Studies*, pages 9–17. Chapman and Hall, London.

[Gevarter, 1982] Gevarter, W. B. (1982). An overview of expert systems. Technical Report Natural Bureau of Standards Report No. NBSIR 82-2505.

[Grady and Caswell, 1987] Grady, R. B. and Caswell, D. L. (1987). *Software Metrics: Establishing a Company-wide Program.* Prentice Hall, New York.

[Grogono et al., 1991] Grogono, P., Batarekh, A., Preece, A., Shinghal, R., and Suen, C. Y. (1991). Expert system evaluation techniques: a selected bibliography. *Expert Systems*, 8(Issue 4):227–240.

[Halstead, 1977] Halstead, M. (1977). *Elements of Software Science.* Elesevier North-Holland, New York.

[Harrison, 1984] Harrison, W. (1984). Applying McCabe's complexity measure to multiple exit programs. *Software-Practice Experience*, pages 1004–1007.

[Harrison, 1992] Harrison, W. (1992). Software measurement and metrics. *Encyclopedia of Computer Science and Technology*, 26(Supplement 11):363-372.

[Harrison and Magel, 1981a] Harrison, W. and Magel, K. (1981a). A complexity measure based on nesting level. *ACM SIGPLAN Notices*, 16(3):63–74.

[Harrison and Magel, 1981b] Harrison, W. and Magel, K. (1981b). A topological analysis of computer programs with less than three binary branches. *ACM SIGPLAN Notices*, 16(4):51–63.

[Harrison et al., 1982] Harrison, W., Magel, K., and Raymond Kluczny, A. D. (1982). Applying software complexity metrics to program maintenance. *Computer*, 15:65–79.

[Henry and Kafura, 1981] Henry, S. and Kafura, D. (1981). Software structure metrics based on information flow. *IEEE Trans. on Software Engineering*, SE-7(5):510–518.

[IEEE, 1983] IEEE (1983). *IEEE Standard Glossary of Software Engineering Terminology*. Software Engineering Technical Committee of the IEEE Computer Society, IEEE, New York.

[IEEE, 1989] IEEE (1989). *Standard Dictionary of Measures to Produce Reliable Software*. IEEE Standard Board, IEEE, New York.

[Ingels, 1971] Ingels, F. M. (1971). *Information and Coding Theory*. Intext Educational Publishers, San Francisco.

[Inglis, 1985] Inglis, J. (1985). Standard software quality metrics. *AT & T Technical Journal*, 65(2):113–118.

[Itzfeldt, 1990] Itzfeldt, W. D. (1990). Quality metrics for software management and engineering. In Mitchell, R., editor, *Managing Complexity in Software Engineering*, pages 127–151. Peter Peregrinus Ltd., London.

[Jacob and Froscher, 1990] Jacob, R. J. K. and Froscher, J. N. (1990). A software engineering methodology for rule-based systems. *IEEE Transactions on Knowledge and Data Engineering (US)*, 2(2):173–189.

[Kaisler, 1986] Kaisler, S. H. (1986). Expert system metrics. In *Proc. 1986 IEEE International Conference on Systems, Man, and Cybernetics*, volume 1, pages 114–120. IEEE.

[Kiper, 1992] Kiper, J. D. (1992). Structural testing of rule-based expert systems. *ACM Transaction on Software Engineering and Methodology*, 1(2):168–187.

[Landauer, 1990] Landauer, C. (1990). Correctness principles for rule-based expert systems. *Expert Systems with Applications (US)*, 1(3):291–316.

[Li and Cheung, 1987] Li, H. F. and Cheung, W. (1987). An empirical study of software metrics. *IEEE Trans. on Software Engineering*, SE-13(6):697–708.

[Liebowitz, 1990] Liebowitz, J. (1990). Issues in verifying and validation of expert systems. In *Proc. of Fourth International Symposium on Knowledge Engineering*, pages 105–115.

[Liebowitz, 1991] Liebowitz, J. (1991). *Operational Expert System Applications in the United States.* Pergamon Press, New York.

[Manns and Coleman, 1988] Manns, T. and Coleman, M. (1988). *Software Quality Assurance.* Macmillan Education, London.

[McCabe, 1976] McCabe, T. J. (1976). A complexity measurement. *IEEE Trans. on Software Engineering*, SE-2(4):308–320.

[Möller and Pauliosh, 1993] Möller, K. H. and Pauliosh, D. J. (1993). *Software Metrics*. Chapman & Hall Computing, New York.

[Mehrotra, 1991] Mehrotra, M. (1991). Rule grouping: A software engineering approach towards verification of expert system. NASA Contract Report 4372, Vigyan Inc., NASA Langley, Hampton VA.

[Miller, 1990] Miller, L. A. (1990). Dynamic testing of knowledge bases using the heuristic testing approach. *Expert Systems with Applications (US)*, 1(3):249–269.

[Nazareth and Kennedy, 1990] Nazareth, D. L. and Kennedy, M. H. (1990). Verification of rule-based knowledge using directed graphs. Technical report, School of Business Administration, University of Wisconsin-Milwaukee, Milwaukee WI 53201.

[Ostle and Malone, 1988] Ostle, B. and Malone, L. C. (1988). *Statistics in Research*. Iowa State University Press, Ames, Iowa 50010.

[Pedersen, 1991] Pedersen, K. (1991). Well-structured knowledge bases. In Gupta, U. G., editor, *Validating and Verifying Knowledge-Based Systems*, pages 365–386. IEEE Computer Society Press, Los Alamitos, California.

[Philip, 1993] Philip, G. C. (1993). Guidelines on improving the maintainability and consultation of rule-based expert systems. *Expert Systems with Applications*, 7(2):169–179.

[Plant, 1991a] Plant, R. T. (1991a). Factors in software quality for knowledge-based systems. *Information and Software Technology (UK)*, 33(7):527-536.

[Plant, 1991b] Plant, R. T. (1991b). Rigorous approach to the development of knowledge-based systems. *Knowledge Based Systems*, 4(4):186-196.

[Preece, 1990] Preece, A. D. (1990). The role of specifications in expert system evaluation. In Culbert, C., editor, *AAAI-90 Workshop on Knowledge Based Systems Verification, Validation and Testing.* AAAI. Unpublished Workshop Notes.

[Preece and Shinghal, 1991] Preece, A. D. and Shinghal, R. (1991). Practical approach to knowledge base verification. In *Proc. of the Conference on Applications of Artificial Intelligence 9*, pages 608-619.

[Prerau et al., 1990] Prerau, D. S., Gunderson, A. S., Reinke, R. E., and Adler, M. R. (1990). Maintainability techniques in developing large expert systems. *IEEE Expert*, pages 71-80.

[Ramamoorthy et al., 1985] Ramamoorthy, C. V., Tsai, W.-T., Yamaura, T., and Blide, A. (1985). Metrics guided methodology. In *Proc. of IEEE Computer Society's Ninth International Computer Software & Application Conference*, pages 111-120.

[Rushby, 1988] Rushby, J. (1988). Quality measures and assurance for AI software. NASA Contractor Report CR-4187, SRI International, Menlo Park, CA. 137 pages.

[Rushby and Crow, 1990] Rushby, J. and Crow, J. (1990). Evaluation of an expert system for fault detection, isolation, and recovery in the manned maneuvering unit. NASA Contract Report CR-187466, SRI international, Meno Park, CA.

[Schneiderman, 1980] Schneiderman, B. (1980). ***Software Psychology: Human Factors in Computer and Information Systems.*** Winthrop Publishers, Inc., Cambridge, Massachusett.

[Schulmeyer, 1987] Schulmeyer, G. G. (1987). Software quality assurance–coming to terms. In Schulmeyer, G. G. and McManus, J. I., editors, ***Handbook of Software Quality Assurance***, pages 1–13. Van Nostrand Reinhold Company, New York.

[Sheppard, 1988] Sheppard, M. (1988). An evaluation of software product metrics. *Information and Software Technologies*, 30(3):177–188.

[Shinghal, 1992] Shinghal, R. (1992). *Formal Concepts in Artificial Intelligence.* Chapman & Hall, New York.

[Soloway, 1987] Soloway, E. (1987). Assessing the maintainability of xcon-in-rime: Coping with the problems of a very large rule-base. In *Proceedings of AAAI-87 Workshop on Validation and Verification of Expert Systems*, pages 824–849.

[Suen et al., 1989] Suen, C. Y., Grogono, P. D., and Shinghal, R. (1989). Applicability of software engineering techniques to expert systems. Report for Bell Canada, Concordia University, Department of Computer Science, 1455 de Maisonneuve Blvd. West, Montréal QC, Canada H3G 1M8.

[Suen et al., 1990] Suen, C. Y., Grogono, P. D., Shinghal, R., and Coallier, F. (1990). Verifying, validating, and measuring the performance of expert systems. *Expert Systems with Applications (US)*, 1(2):93–102.

[Suen and Shinghal, 1991] Suen, C. Y. and Shinghal, R. (1991). ***Operational Expert System Applications in Canada.*** Pergamon Press, New York.

[Tai, 1984] Tai, K.-C. (1984). A program complexity metric based on data flow information in control graphs. In ***Processing of 7th International Conference on Software Engineering***, pages 239–248.

[Tajima, 1981] Tajima, D. M. (1981). The computer software industry in japan. *Computer*, 14(5):89–96.

[Weyuker, 1988] Weyuker, E. J. (1988). Evaluating software complexity measures. *IEEE Tran. on Software Engineering*, SE-14(9):1357–1365.

[Williams, 1986] Williams, C. (1986). Expert systems, knowledge engineering, and ai tools— an overview. *IEEE Expert*, winter:66–70.

[Zuse, 1991] Zuse, H. (1991). *Software Complexity: Measures and Methods.* Walter de Gruyter, D-1000 Berlin.

[Zuse and Bollmann, 1989] Zuse, H. and Bollmann, P. (1989). Software metrics: Using measurement theory to describe the properties and scales of static software complexity metrics. *SIGPLAN Notices*, 24(8):510–518.

Appendix A

t Values

This appendix presents tabled t distribution in a two-tail test.

The probabilities (significant levels) are given by the column headings, and the first column shows the degree of freedom. For example, with 60 d.f., we have

$$P(t \geq 2.660) + P(t \leq -2.660) = 0.01$$

That is the significant level for the critical region: $t \geq 2.660 \wedge t \leq$ is 2.660.

	Significance Levels					
d.f.	p=0.1	p= 0.05	p=0.02	p=0.01	0.002	p=0.001
1	6.314	12.706	31.821	63.657	318.31	636.62
2	2.920	4.303	6.965	9.925	22.327	31.598
3	2.353	3.182	4.541	5.841	10.214	12 924
4	2.132	2.776	3.747	4.604	7.173	8.610
5	2.015	2.571	3.365	4.302	5.893	6.869
6	1.943	2.447	3.143	3.707	5.208	5.959
7	1.895	2.365	2.998	3.499	4.785	5.408
8	1.860	2.306	2.896	3.355	4.501	5.041
9	1.833	2.262	2.821	3.250	4.297	4.781
10	1.812	2.228	2.764	3.169	4.144	4.387
11	1.796	2.201	2.718	3.106	4.025	4.437
12	1.782	2.179	2.681	3.055	3.930	4.318
13	1.771	2.160	2.650	3.012	3.852	4.221
14	1.761	2.145	2.624	2.977	3.787	4.140
15	1.753	2.131	2.602	2.947	3.733	4.073
16	1.746	2.120	2.583	2.921	3.686	4.015
17	1.740	2.110	2.567	2.898	3.646	3.965
18	1.734	2.101	2.552	2.878	3.610	3.922
19	1.729	2.093	2.539	2.861	3.579	3.883
20	1.725	2.086	2.528	2.845	3.552	3.850
21	1.721	2.080	2.518	2.831	3.527	3.819
22	1.717	2.074	2.508	2.819	3.505	3.792
23	1.714	2.069	2.500	2.807	3.485	3.767
24	1.711	2.064	2.492	2.797	3.467	3.745
25	1.708	2.060	2.485	2.787	3.450	3.725
26	1.706	2.056	2.479	2.779	3.435	3.707
27	1.703	2.052	2.473	2.771	3.421	3.690
28	1.701	2.048	2.467	2.763	3.408	3.674
29	1.699	2.045	2.462	2.756	3.396	3.659
30	1.697	2.042	2.457	2.750	3.385	3.646
40	1.684	2.021	2.423	2.704	3.307	3.551
60	1.671	2.000	2.390	2.660	3.232	3.460
120	1.658	1.980	2.358	2.617	3.160	3.373
∞	1.645	1.960	2.326	2.576	3.090	3.291

Table 29: Tabled *t* Values

Appendix B

r Values

This appendix shows tabled r values in two-tail test. It is obtained by solving r from the relation

$$t = r\sqrt{\frac{n-2}{1-r}}$$

where, t represents the t distribution.

The probabilities (significant levels) are given by the column headings, and the first column shows the degree of freedom. For example, with 62 d.f., we have

$$P(r \geq 0.325) + P(r \leq -0.325) = 0.01$$

That is, the significant level for the critical region $r \geq 0.325 \wedge r \leq -0.325$ is 0.01.

		Significance Levels		
n	d.f.	0.05	0.01	0.001
3	1	0.997	0.999	1.000
4	2	0.950	0.990	0.999
5	3	0.878	0.959	0.991
6	4	0.811	0.917	0.974
7	5	0.754	0.875	0.951
8	6	0.707	0.834	0.925
9	7	0.666	0.798	0.898
10	8	0.632	0.765	0.872
11	9	0.602	0.735	0.847
12	10	0.576	0.708	0.823
13	11	0.553	0.684	0.810
14	12	0.532	0.661	0.780
15	13	0.514	0.641	0.760
16	14	0.497	0.623	0.742
17	15	0.482	0.606	0.725
18	17	0.456	0.575	0.693
20	18	0.444	0.561	0.679
21	19	0.433	0.549	0.665
22	20	0.423	0.537	0.652
27	25	0.381	0.487	0.597
32	30	0.349	0.449	0.554
37	35	0.325	0.418	0.519
42	40	0.304	0.393	0.490
47	45	0.288	0.372	0.465
52	50	0.273	0.354	0.443
62	60	0.250	0.325	0.408
72	70	0.232	0.302	0.380
82	80	0.217	0.283	0.357
92	90	0.205	0.267	0.338
102	100	0.195	0.254	0.321

Table 30: Tabled r Values

Appendix C

Z Values

This appendix presents the values of Z, the standard normal variable, from 0.0 to 3.9, showing the cumulative probability up to z, that is, the table entry represents $P(Z \leq z)$. The probability for the area $(Z \geq z) \wedge (Z \leq -z)$ (significant level) in two-tail test can hence be obtained by:

$$P(Z \geq z) + P(Z \leq -z) = 2 \times (1.0 - P(Z \leq z))$$

For example, when $z = 2.24$, from the table, we get

$$P(Z \leq 2.24) = 0.9838$$

Hence,

$$P(Z \geq 2.24) + P(Z \leq -2.24) = 2 \times (1 - 0.9838) = 0.0324$$

z	0.00	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
0.0	0.5000	0.5040	0.5080	0.5120	0.5160	0.5199	0.5239	0.5279	0.5319	0.5359
0.2	0.5793	0.5832	0.5871	0.5910	0.5948	0.5987	0.6026	0.6064	0.6103	0.6141
0.4	0.6554	0.6591	0.6628	0.6664	0.6700	0.6736	0.6772	0.6808	0.6844	0.6879
0.6	0.7257	0.7291	0.7324	0.7357	0.7389	0.7422	0.7454	0.7486	0.7517	0.7549
0.8	0.7881	0.7910	0.7939	0.7967	0.7995	0.8023	0.8051	0.8078	0.8106	0.8133
1.0	0.8413	0.8438	0.8461	0.8485	0.8508	0.8531	0.8554	0.8577	0.8599	0.8621
1.2	0.8849	0.8869	0.8888	0.8907	0.8925	0.8944	0.8962	0.8980	0.8997	0.9015
1.4	0.9192	0.9207	0.9222	0.9236	0.9251	0.9265	0.9279	0.9292	0.9306	0.9319
1.6	0.9452	0.9463	0.9474	0.9484	0.9495	0.9505	0.9515	0.9525	0.9535	0.9545
1.8	0.9641	0.9649	0.9656	0.9664	0.9671	0.9678	0.9686	0.9693	0.9699	0.9606
2.0	0.9772	0.9778	0.9783	0.9788	0.9793	0.9798	0.9803	0.9808	0.9812	0.9817
2.2	0.9861	0.9864	0.9868	0.9871	0.9875	0.9878	0.9881	0.9884	0.9887	0.9890
2.4	0.9918	0.9920	0.9922	0.9925	0.9927	0.9929	0.9931	0.9932	0.9934	0.9936
2.6	0.9953	0.9955	0.9956	0.9957	0.9959	0.9960	0.9961	0.9962	0.9963	0.9964
2.8	0.9974	0.9975	0.9976	0.9977	0.9977	0.9978	0.9979	0.9979	0.9980	0.9981
3.0	0.9987	0.9987	0.9987	0.9988	0.9988	0.9989	0.9989	0.9989	0.9990	0.9990
3.2	0.9993	0.9993	0.9993	0.9994	0.9994	0.9994	0.9994	0.9995	0.9995	0.9995
3.4	0.9997	0.9997	0.9997	0.9997	0.9997	0.9997	0.9997	0.9997	0.9997	0.9998
3.6	0.9998	0.9998	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999
3.8	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999
3.9	1.0000									

Table 31: Tabled Z Values