STATISTICAL TECHNIQUES

AND

PROJECT MONITORING

by

Lesley Margaret Pickard

PhD

City University

Computer Science Department

March 1994

Contents Page

Illustrations

Tables

Acknowledgements

I would like to thank my supervisor Dr. Barbara Kitchenham for all her help, advice and encouragement during the work and write-up of my thesis.

I would also like to acknowledge the help and discussions of my colleagues on the REQUEST project, especially Steve Linkman. The work for my thesis was funded by ESPRIT as part of the Reliability and Quality for European Software Technology (REQUEST) project. I wish to thank STL/ICL for their support.

I would like to thank Eric Barber for his help with the implementation of the expert system rules and Colin Meredith and Max Turner for their involvement in the Verification and Validation exercise.

In addition, I would like to thank Barbara Kitchenham for the collection of the datasets and Sharon Aid for the complexity assessment for dataset 3. I would also like to thank Sue Linkman for allowing me to use her boxplot illustrations for dataset 4 which are in Appendix C.

Declaration

Abstract

The aim of this thesis is to identify statistical techniques which are appropriate for the analysis of software development metrics and to investigate how they might be useful to support quality management procedures.

The initial approach was to investigate the use of statistical techniques to identify consistent relationships between measures collected during the development and fault or change-proneness of the final product. No common relationships were identified between the datasets when module relationships were considered. Therefore, there is little hope of identifying any general relationships between module attributes and product quality attributes.

However, some techniques were good at identifying outlier/anomalous components irrespective of the particular attributes. For univariate outlier detection a modification of the boxplot technique was found to be useful. This is described in the document. For bi-variate outliers, scatterplots were found to be useful. This thesis describes how the scatterplot technique can be automated to objectively identify outliers. It describes a set of rules which were implemented into a prototype. The objective was to produce a technique which most consistently identified the anomalies that had been identified subjectively by an expert consultant.

The thesis describes how summary statistics can be useful at the project level. It identifies a sub-set of useful information to enable a project manager to control his/her project. A target value, where appropriate is suggested for each measurement. Monitoring is based on the principle that when an actual attribute value exceeds the target value then it is likely to be a potential problem in the development.

A survey highlighted that for automatic anomaly detection to be of any significant benefit to a project manager, some interpretation is required to identify the likely cause of the anomaly and its effect on the project. The thesis shows how the cause of an anomaly can be diagnosed with the help of a simple expert system which looks at a combination of attribute values for diagnosis.

1. Introduction

The aim of this thesis is to identify statistical techniques which are appropriate for the analysis of software development metrics and to investigate how they might be used to support quality management procedures.

The author's specific objectives originated from a European collaborative project called REQUEST. The part of REQUEST the author was involved in was the development of a COnstructive QUality MOdel (COQUAMO). It was intended that COQUAMO would use measurements collected during the development to predict final product quality. The REQUEST project, and in particular, the COQUAMO model is described in section 1.1. The author was only involved in the monitoring mode of the model. Her initial objectives were to:

- Identify consistent/general relationships between software measurements collected during the development and fault or change-proneness of the product;

- Identify measurements and methods for detecting atypical software components;

- Construct stable, predictive models.

The initial approach was to investigate the use of statistical techniques required to meet the above objectives and to analyse software data sets to validate the proposed techniques. The author also had a requirement to use the statistical techniques to automate the use of the predictive models. As a result of the author's initial work (section 3, "Preliminary Analysis") the view of the project goals changed. This altered the author's goals to:

- identify methods to automate anomaly detection;

- set up an advice system to help diagnose problems, that is, automate the interpretation of the detected anomalies.

Chapter 2 discusses in general why measurement of software attributes is important and how it can help project managers control their projects during development.

The REQUEST project team envisaged that project managers would wish to predict quality throughout the development of a product and to detect problems before the end of the project. In chapter 3, the author describes the investigations and analyses she undertook to identify consistent/general relationships and atypical values. This analysis revealed that the nature of software attribute data affects the use of classical statistical techniques and shows how some of the problems could be overcome.

Before the author could choose a statistical technique which automates the detection of atypical values, she investigated type of anomalies an experienced project manager would detect as anomalous. Chapter 4 describes a survey which the author carried to do this.

The results in Chapter 3 revealed no evidence that a statistical relationship exists between development measures taken during development and final product quality,(see chapter 3 and [1]). Therefore the REQUEST team decided to concentrate on the automatic detection of anomalies found during the development process in the monitoring mode of COQUAMO. The monitoring mode was intended to support the project manager at two different levels - project and component level assessment of project progress.

Chapter 5 describes the use of project-level summary information and the setting of targets to help project managers to control projects. The author suggests the type of information the project manager would find useful and sets some default targets values. Default target values are required because data will not always be available to generate the target value.

Component-based monitoring is described in chapter 6. The author was responsible for choosing an appropriate statistical technique to automate component-based anomaly detection. The technique chosen required some adaption and the chapter describes what the adaption was and why it was required. This chapter also describes the verification and validation process of the prototype and the results found.

The need for an advice system was identified in the survey described in chapter 4. A statistical based anomaly detection only identifies atypical or anomalous values, it does not provide any indication of why the values were atypical. Chapter 7 shows how the use of an advice or simple expert system can help interpretation. The author was responsible for converting expert knowledge into an advice system. The chapter also describes the verification and validation process of the advice system and the results.

The datasets used for the preliminary analysis, the automation of the anomaly detection and the verification and validation process are described in section 1.3 of this chapter.

1.1 Overview of the REQUEST project

REQUEST was one of the ESPRIT Software Technology projects. Its name was derived from **Re**liability and **Qu**ality for **E**uropean **S**oftware **T**echnology. The aim of the REQUEST project was to provide improved and validated techniques for measuring and modelling software quality and reliability. This was to be supported by appropriate prototype tools. The project was concerned with the use of quantitative information and models which provide information to help project and quality management decision making and control.

It was organised into three sub-projects:

Sub-project 1 - Quality measurement, modelling and prediction;
Sub-project 2 - Reliability measurement, modelling and prediction;
Sub-project 3 - Data collection and storage.

The aim of Sub-project 1 was to produce a **C**onstructive **Qu**ality **M**odel (COQUAMO). Originally COQUAMO was assumed to be empirically based, that is, based on observed relationships between quality factors obtained in a product, and various quality indicators which are quantitative in nature. This implied the need to identify techniques which can reveal and formulate such relationships.

The reliability modelling work was concerned with developing reliability models which

incorporate testing information. This involved devising new approaches to modelling the reliability of single and diverse N-version software systems. It included the modelling of ultra-high reliability systems capable of incorporating failure information of single version but taking into account fault-tolerant architectures.

The aim of sub-project 3 was to provide software data to allow the validation of models produced by the other two sub-projects. As part of the work of devising a database, the sub-project also produced data collection forms and manuals.

1.2 Overview of COQUAMO

The idea of COQUAMO, [2], was inspired by the Constructive Cost Model, COCOMO, [3]. COCOMO predicts software effort using cost drivers and COQUAMO was intended to be a set of predictive models which would use quality drivers to predict the end-product quality. Conventially, software engineers seldom consider the issue of software quality achievement as part of their problem. This is partially due to the feeling that quality is achieved by controlling the production methods not the product itself. This has been reinforced by quality assurance development guidelines which identify good techniques and assess projects in terms of whether or not these techniques have been adhered to.

The aim of REQUEST was to link assessment to measurable features of the software as well as adherance to procedures. REQUEST was concerned with the problems of quantitative prediction and assessment. In particular, REQUEST was concerned with the means by which such predictions and evaluations may be used to assist software producers in initial project set-up and the process of product development throughout the software life cycle.

As part of the background to COQUAMO, the REQUEST project had to consider how software quality could be specified and measured. The existing quality models were based on the quality factor, criteria, metric model. This is a hierarchical model that defines a set of quality factors which are intended to characterise quality from the user's viewpoint. The factors are broken down into quality criteria which are related to a set

of measurable attributes called quality metrics. This approach was used by various research workers and their work is described in section 2.2 of this thesis. There are many limitations (identified in section 2.2) with this approach which makes its use in practice difficult.

Gilb [4] suggested an alternative approach to quality modelling where quality and resource requirements should be specified individually for any software system , not a general set of factors as suggested with the above approach. The attributes are not broken into quality criteria but into more and more specific attributes until they are directly measurable.

The REQUEST aim was to update and improve ideas on Quality Models. Their approach was to refine the McCall and Walter's model [5] and use Gilb's approach to quantifying quality.

The REQUEST team realised that the idea of predictive models which cover all phases of software development was not feasible since there was no evidence that a statistical relationship existed between end-product quality and software attributes collected during the development process. This implied that there were two different types of metrics, metrics related to final product qualities and metrics related to software production.

The team decided to divide the model into three modes which reflect the three stages of software development - planning, monitoring and assessment. The metrics related to final product quality are relevant at the planning and the assessment modes and the metrics related to the software production are relevant at the monitoring mode.

Planning

The planning mode was intended for use during the early stages of system planning and feasibility assessment of the quality levels planned. It is intended to:

- help the user identify and specify quality requirements;

- predict final product qualities from the values of measures (quality drivers) available at the start of a project from plans and constraints.

The final product qualities (quality factors) were selected from various quality factors suggested in the literature [5], [6] and [7]. The quality factors which were selected and the selection process are described in [8] and [9]. The quality drivers, along with their function and origin, are described in [10].

The planning mode is the closest to the original idea of COQUAMO. The model for this mode (COQUAMO-1) is operational from the start of the project until high level design when the planning is complete.

Monitoring

The monitoring mode is intended to assist the project manager to monitor and control the development process, using quantitative measures (software attributes) that can be collected during development. Once the planning is complete and the end product qualities have been assessed as feasible, it is assumed that the planned development process will result in the achievement of the final product qualities. The monitoring system (COQUAMO-2) is intended to detect when there is a deviation from plan and provide help in the control of the development process. If there is a deviation from plan this implies that the end-product qualities may not be achieved if no action is taken.

COQUAMO-2 has two major components:

- A range of statistical techniques which will be used to identify unusual values. At the project level this is basically the use of summary information and deviations from set targets. The component level monitoring involves more sophisticated techniques;

- An advisory system which will provide some possible interpretations of the cause of the detected unusual values.

COQUAMO-2 is active throughout the development period from the requirements phase until the product has been completed. The author's research activities were concerned with this model only.

Assessment

The assessment mode is invoked near the end of the integration testing. It is intended to assist final product assessment in determining whether the product quality characteristics observed conform to those initially specified during the planning mode.

The set of models for this mode (COQUAMO-3) provide feedback to COQUAMO-1. These results will be of interest in decisions on product release and in planning support for the operational phase. This should also provide information to other projects to allow better estimation. The models used for reliability and usability are reliability growth models and are described in [11] and [12]. The COQUAMO-3 assessment can start when the software product exists in a sufficiently complete version to allow realistic operational testing.

1.3 Data Descriptions

Four datasets have been used to validate the statistical techniques investigated in this thesis for the analysis of software data.

1.3.1 Datasets 1 and 2

These two data sets were implementations of the same functional requirement of a large operating system and have the same descriptions. Dataset 1 consists of 27 modules and dataset 2 consists 40 modules. For each module eleven measures were collected.

(1) Machine code Instructions (MCI)

This is a count of the number of machine level instructions in the compiled

version of the modules, in bytes. (code based metric)

(2) Lines of code (LOC)

This is a count of the number of lines of code in the module between the first BEGIN and the ENDMODULE lines. A line was included in this count if the first non-space character was **not** a comment character. Blank lines were **not** included. (code based metric)

(3) Fan-out (FO)

This is the number of other modules called by a module. (design based metric)

(4) Data Items (DI)

This is the number of external global static (i.e. common) data items accessed by a module. (design based metric)

(5) Parameters (PAR)

This is the number of parameters on the interface of a module. (design based metric)

(6) n1

This is the number of distinct operators appearing in a module. This metric and the next three metrics can be used in calculating Halstead's Software Science metrics [13]. (code based metric)

(7) n2

This is the number of distinct operands appearing in a module. (code based metric)

(8) N1

This is the total number of all operators appearing in a module. (code based metric)

(9) N2

This is the total number of all operands appearing in a module. (code based

module)

(10) McCabe's V (V(G))

This is McCabe's cyclomatic number which defines the complexity of a module in terms of its control structure and is represented by the maximum number of linearly independent paths through a module [14]. (code based metric because collected at the coding stage, but could be derived from the design)

(11) Changes (CHG)

This is the number of changes made to a module , after the module was put under formal configuration control.

Some combinations of the basic metrics have been used:

$$n = n1 + n2$$

$$N = N1 + N2$$

$$HE = n1*N2*N*\log_2(n)/2*n2$$

where HE is Halstead's Effort metric, mental effort to produce a module. (code based metric)

When the metric names are shown with a preceding N (e.g. NN)it means that the metric has been normalised by dividing it by its size (lines of code).

1.3.2 Dataset 3

This data is from a large operating system and consists of 226 modules. For each module twelve measures were collected.

(1) Fan-in

This is the number of modules which call a specified module.

(2) Fan-out

This is the number of modules called by a specified module.

(3) Input parameters

The number of input parameters on a module interface.

(4) Output parameters

The number of output parameters on a module interface. When parameters are used as both input and output parameters they are included in both counts.

(5) Data reads

The number of data structures (not individual elements) the module reads from, but does not also write to.

(6) Data writes

The number of data structures the module writes to, but does not also read from.

(7) Data reads and writes

The number of data structures the module both reads from and writes to.

(8) Size (LEN)

Module size in lines of code, non-comment, non-blank lines in a module.

(9) Control flow (CF)

Module control flow measured in terms of the number of branches.

Notional branches were included so that IF-THEN-ELSE and IF-THEN-ELSE-IF were both counted as two branches. The number of branches for loops with a single control structure (i.e. FOR,WHILE, or UNTIL) was counted as two, and for loops with a dual control structure (i.e FOR and WHILE, or FOR and UNTIL) was counted as three. The compiler evaluated compound booleans lazily, so each AND and OR in a conditional statement or loop control was counted separately.

(10) Module enhancements (CHNG)

The number of times a module was amended excluding changes for fault clearance. The information was obtained from formatted comments in each module which recorded each change during its development and subsequent evolution.

(11) Known errors (KE)

The number of faults corrected in the module.

This information was obtained from formatted comments in each module, recording each fault cleared during its development and subsequent maintenance.

(12) Subjective complexity

An assessment of the complexity of the module on a scale of 1 (very simple) to 5 (very complex) provided by member of the development group.

1.3.3 Dataset 4

Two datasets was combined to form this dataset. They were:

- SCS1
- TCB

These are two subsystems of a real-time embedded software system. The data was collected by a manual inspection of the design documents, the print-out of the coded data and the subsequent error reports.

The data is made up of a number of procedures, seventeen in the SCS1 dataset and thirty-four in the TCB dataset. For each of the procedures the following variables have been extracted (1 to 13) and calculations made (14-16):

(1) lines of code

(2) comments

(3) paths

(4) returns

(5) global (externally accessible) items

(6) instruction boxes

(7) fan-out

(8) number of entry points

(9) fan-in

(10) testing errors

(11) design errors (testing)

(12) coding errors (testing)

(13) specification errors (testing)

(14) errors per lines of code

(15) errors per instruction box

(16) lines of code per instruction box

In the SCS1 dataset there are 3 modules for which there are no coding metrics. This is because the coding was altered after the initial design document as these procedures were found to be redundant. The latest available design documents were used, together with their complementary coding print outs and error reports.

2. Use of Metrics in Control of Projects

Metrics are quantitative measures of software attributes. For example, lines of code is a metric for measuring size and hours is a metric for measuring effort. The use of measurement is important in most areas of work. The need for it can be summarised by the quote "you cannot control what you cannot measure", [15]. Measurement removes vagueness and helps to provide an objective and common basis of understanding.

Software engineering is no exception to this. The use of measurement is as important in this area as in any other. For example, everyone has their own opinion on what quality means. However, in every specification there is an implied requirement for a quality product. This requirement is difficult to achieve without a common view of the term. The attachment of quantitative measures to the specification would help in two ways:

- it identifies what the client means by 'quality';
- it allows testing of the product to confirm that the measure has been achieved.

Many high level management goals identify 'improvement' as one of their aims, for example, improvement in productivity or quality. You cannot judge whether improvement has occurred or not if an objective baseline does not exist with which to compare.

When relevant quantitative measures are available, they provide information to assist the monitoring and control of the production process. Measurement can help provide control of projects in different ways. For example:

- measures can provide a common baseline. If quantitative measures are collected, along with the definitions and counting rules, they can be used as a baseline to compare future measurements with. This is particularly important for comparison among projects, for example, using similar past

projects to help obtain planned or expected values for the current project.

- measures can be incorporated into a management planning and monitoring activity where estimates of effort, timescale and quality are made part of the project plans. Progress can be monitored by comparing the actual values against the planned values. The planned values can be derived either from external targets or from estimates. The measurement process provides the information needed to assess whether externally set constraints are likely to be achieved and whether there are any potential problems occurring with the development process.

- measures can highlight particular problem areas. Recording information about the nature and origin of a defect provides information to identify where major problems are occurring. Information about how defects are detected provides useful information about the efficiency of testing process.

2.1 Views of Quality

There are many different views of what is meant by 'Quality'. Garvin [16] has described five different views of Quality which identify different meanings of quality. The five views are:

(1) Transcendent

This approach to quality comes from philosophy where quality is equated to innate excellence. It cannot be defined precisely and is felt rather than measured. This type of quality can only be recognised through experience and a person can only recognise the presence or absence of it.

(2) Product-based

This approach comes from economics where quality is related to the content of the product. This implies that quality is an inherent characteristic and that higher quality leads to higher cost. This type of quality can be measured objectively

although an individual's preference for the different product attributes is subjective.

(3) User-based

This approach depends on the assumptions as to what the user wants. It is defined as "fitness for purpose". The problem with this type of quality is differentiating between product attributes which represent quality and those which just increase customer satisfaction. This quality view is external to the producing organisation.

(4) Manufacturing-based

This approach of quality is defined as "conformance to specification". Quality from this viewpoint means meeting the specification precisely and first time. This view is internal to the organisation where cost reduction is achieved by minimising specification deviations.

(5) Value-based

This approach is a combination of User-based and Manufacturing-based viewpoints. This viewpoint of Quality means providing what the customer wants at an acceptable price and conformance to specification at an acceptable cost.

In the software industry, the user view, the product view, the manufacturing view and the value-based view are assumed to apply.

The user view of quality is based on an evaluation of the product in the context of the task it is intended to perform and a product is viewed as a quality product if it meets the users requirements. This view should be stated explicitly in the requirements specification. Quality models attempt to map the user view to the product view by relating external quality characteristics to internal product measures.

The manufacturing view regards a quality product as one that is constructed "right first time" and therefore minimises the rework costs during development and after delivery. This view should be covered in the technical specification which directly converts the requirements specification into how it can be produced. Initially these views of quality

should not be in conflict. However, when the user requests changes to the requirements this results in the user's requirement being in conflict with the producer's goal of minimising work. Under these circumstances, the value-based view becomes important. This view considers the trade-off between cost and quality and a decision has to be made as to what the user is prepared to pay for. Once this has been establish then a cost can be matched to it in order to reduce the conflict.

2.2 Background to Software Quality Modelling

In the 1970's, the approach to modelling software quality was to identify characteristics of quality which were consistent with a user's view of a quality software product. These quality components are called quality factors and should contribute to the user's view of the quality of the software product. These quality factors are then broken down in lower level components called quality criteria. The quality criteria are consistent with a system developer's view of quality.

The management of quality requires the ability to objectively measure quality therefore, the quality criteria are related to a set of measurable attributes of the software called quality metrics.

This approach is called the quality factor, quality criteria, quality metric model. It was first put forward by Boehm et al [6] at TRW. Then McCall, Richards and Walters [5] standardised the terminology and the model. More recently Bowen, Wigle and Tsai [7] have added more quality factors. The additional quality factors have resulted in more quality criteria being added. The quality factors identified by the three sets of research workers are shown in Figure 2.1.

Boehm et al	**McCall et al**	**Bowen et al**
Efficiency	Efficiency	Efficiency
Reliability	Reliability	Reliability
Human Engineering	Usability	Usability
Modifiability	Flexibility	Expandability
Portability	Portability	Portability
Testability	Testability	Verifiability
Understandability	Reusability	Reusability
	Maintainability	Maintainability
	Interoperability	Interoperability
	Correctness	Correctness
	Integrity	Integrity
		Flexibility
		Survivability

Figure 2.1 Software Quality Factors

There are a number of problems with the factor, criteria, metric model:

- The natures of the factors, criteria and metrics are very different for different qualities. The factors are a mixture of general, specialist system and software development process factors. The criteria are a mixture of more detailed descriptions of the quality factor, functional features and development process features. The metrics are a mixture of genuine metrics, checklists and production standards.

- There are overlaps between the different quality factors.

- There is no clear indication of the trade-off relationships between the factors (e.g efficiency against maintainability).

- There is no explicit relationships between criteria, metrics and the life-cycle.

- There is no objective rationale for including, or excluding, a particular quality factor.

- The quality factors are not defined in measurable terms so validation of any relationship between quality factor and quality metrics is difficult.

Gilb [4] suggests an alternative approach to modelling software quality. His view is that of a system user and he aims to provide an agreed statement of quality attributes at the start of production which can be validated directly in the final product.

Gilb suggests that quality and resource requirements should be specified individually for any software system and that the specification process should include an indication of the relative priority of the specified attributes. The attributes are broken down into more specific attributes and they are further broken down during their specification until the attribute can be measured directly.

The REQUEST approach was aimed at the software producer and attempted to synthesis the work of Gilb and McCall et al. Since there was no obvious link between the different level of the system it was decided to work at two different levels - project and component.

Another approach to quality modelling is Quality Factor Deployment (QFD), [17,18]. QFD is a structured procedure which combines the customer requirements and the design requirements/technical characteristics required to meet the customer requirements. It displays them in the form of a matrix which ensures that the customer requirements are preserved throughout the design process and the design requirements are clearly linked to them. This graphical display highlights the customer requirements which are not being met and the design requirements which do not address any customer requirements.

A quantifiable measure, which represents the target value, is attached to each design requirement and must be understandable and relevant to the customer. The QFD matrix requires the strength of the relationship between the customer requirement (the WHAT) and the design requirement (the HOW) to be identified and the relationship between the design requirements to be identified. This helps to identify any potential problem areas.

The QFD procedure also requires cross-functional team participation at all stages of planning. This reduces the need for rework due to a lack of communication and understanding. The QFD can be extended to a sequence of matrices where each level of matrix uses the HOWs selected from the previous level to represent the WHATs in

the current level.

Gilb suggested a similar approach to QFD whereby quality requirements were related to the software engineering techniques intended to implement them. This approach was used by Walker and Kitchenham [59] to develop a system for assessing the feasibility of quality requirements. These ideas were later automated and used to provide support for COQUAMO-1.

2.3 Basis of Thesis

The work in this thesis was concerned with the automation of the work by Kitchenham and Walker [19]. They used the approach to monitoring and control of projects first suggested by Doerflinger and Basili, [20]. Doerflinger and Basili compared a set of measurements collected at several stages (starting at the coding stage) in the development process with baseline values. The baselines were obtained by calculating the average value and standard deviation of each of the measurements taken from a group of similar past projects. The actual measurement value was considered abnormal if the value was greater than or less than one standard deviation away from the group average. Basili later extended this work to included an expert system approach to interpret the measurements [21].

Kitchenham and Walker extended Doerflinger and Basili's original work by extending the scope of monitoring process. They covered project phases from requirements specification to integration stage. They also derived the baselines using robust statistical analyses rather than the classical summary statistics. They felt this was more accurate for software data. Kitchenham and Walker also suggested the use of component-based monitoring as well as project-based monitoring which allows the monitoring of a project even in the absence of similar past projects.

One of the major results of Kitchenham and Walker work was that they highlighted additional problems with anomaly detection which have to be considered. Their work suggests that the same value can have a variety of different meanings. For example a low fault rate could mean high quality or poor testing.

Huff et al, [22], also suggested the use of quantitative information to monitor the development of a product. Their idea was based on the use of quantitative models formulated as a set of equations whose variables describe a particular software development activity. Again monitoring was performed against expected values derived from the models and management action was required if the actual measures deviated from plans. The primary goal was to provide feedback on a particular software activity during the development process. Quantitative models support all phases of project management and are constructed as part of the detailed planning. The input variables for the equations are usually based on expert opinion estimates or values obtained from previous similar projects.

2.4 Latest Research in Area

This section discusses the latest research results in software metrics for project control. Ramsey and Basili [21] evaluated the use of expert systems in software engineering management. They looked at four different prototype expert systems which, given values of identified metrics, provided an interpretation to explain abnormal combination of values. Ramsey and Basili found that the most complete and accurate solutions were from using a bottom-up approach to knowledge acquisition. This approach is application specific as opposed to the more general top-down approach. They also found that rule-based expert systems were better than frame-based since frame-based systems often missed interpretations because of incorrect relationships between the metrics. The interpretation system described in chapter 7 is a rule-based system which uses the bottom-up approach.

Ramsey and Basili used two experts to obtain their knowledge for the systems and found that the required knowledge was not yet well understood. The experts often disagreed on the interpretations and the relationships between the metrics. This confirms what was found by the survey described in chapter 4.

The work on quality since the end of the thesis appears to be concentrating on anomaly detection and the use of classification instead of complex predictive equations. The use of simple non-parametric techniques is increasing but often used in conjunction with the

complex statistical techniques.

Selby and Porter [23] investigated the use of decision trees to identify modules which had high development effort or faults. A 'high' value was defined to be one which was in the upper quartile using past data. They were concerned with the feasibility of automatic generation of a decision tree was and whether such trees could be both simple and accurate enough to deal with the software resource analysis problem, especially when different amounts of data are available to generate the trees. Selby and Porter found that the decision trees correctly identified 79.3% of the software modules that had either high development effort or faults.

Munson and Khoshgoftaar [24] used principal components to obtain uncorrelated metrics and then applied discriminant analysis to classify the programs as fault-prone and not fault-prone. They divided the data into two and randomly selected 260 programs to develop the model, with the remaining 160 used to validate the model. The number of changes were classified into two groups of 0 or 1 change and 10 or more changes (programs with 2-9 changes were excluded from the investigation). At the 10% level of significance the procedure identified 75% of one type of program correctly and 62% of the other type. They emphasised the need for accurate and well-defined data collection.

Briand, Basili and Hetmanski [25,26] have developed models for identifying high risk software components which require extra testing/verification effort. They compared logistic regression, classification trees and optimised set reduction (OSR). OSR is based on machine learning principles and univariate statistics. It uses logical expressions to represent patterns in a data set. The validation indicated that OSR was more accurate than either logistic regression or classification trees. In their environment, Briand, Basili and Hetmanski found that it was a good alternative to multivariate logistic regression. They also thought that classification trees might be too simplistic for modelling high risk software components.

Selby and Basili [27] used the concept of coupling and strength to characterise the structure of a software system. They calculated a 'coupling/strength' ratio for a cluster of routines within a subsystem. Strength is the amount of interaction within a software

component and coupling is the amount of interaction between software components of a system. The cluster were joined bottom-up, that is the components with the lowest ratio were joined first. Selby and Basili found that the routines with the lowest coupling/strength ratios had 7 times fewer errors per thousand lines of code (KLOC) than the routines with the highest ratio values. The errors were also 21.7 times less costly to fix. They also found that subsystems with low ratio values had routines with 4.8 times fewer errors per KLOC than subsystems with high ratio values.

Agresli and Evanco [28] also found coupling had a significant effect on software defects along with information flow. They used multivariate regression to predict software defects. However, they had problems with multicollinearity, i.e. correlation between the independent or explanatory variables in the equation.

Non-parametric techniques are also being used to evaluate software metrics. Courtney and Gustafson [29] used non-parametric techniques to highlight the problem with the 'shotgun' approach to evaluating software measures. The approach involves trying variables until a significant Pearson's correlation coefficient is found. No initial hypothesis is stated. They show that a large number of non-independent variables (multicollinearity) and the limited amount of data available make the chance of finding an accidental relationship high.

Courtney and Gustafson used simulation to highlight the problems of the shotgun approach. They found that although the probability of a Type 1 error (finding a significant relationship when one does not really exist) was fixed at 0.05, it was significantly larger than 0.05 when Pearson's r coefficient was used. They compared Pearson's r with Spearman's ρ and Kendall's Tau. They do emphasise that a true shotgun approach is unlikely to be applied in practice and intend to repeat the experiment with an actual data set. They emphasised the need for more research in software measures and thought the use of non-parametric techniques was important in validation.

Schneidewind [30] used the non-parametric techniques of rank correlation and contingency tables to evaluate metrics against a validation criterion. The validation approach was user-based and consisted of six mathematically defined criteria to relate

a software attribute to software quality. He identified a validated metric as "one whose values have been shown to be statistically associated with corresponding values". He used the rank correlation coefficient to test for consistency and the contingency tables to test for correct and incorrect classifications. He also emphasised that non-parametric techniques played an important part in evaluating metrics.

Khoshgoftaar et al [31] investigated new estimation procedures. They described two new estimation procedures and compared the performance in modelling software quality in terms of the predictive quality and the quality of fit with least squares regression (LS) and least absolute value (LAV) techniques. The two new estimation techniques were relative least squares (RLS) and Minimum Relative Error (MRE). They used the same data that has been used for the analysis described in chapter 3. They confirmed the problem of predicting quality and the need to establish baselines.

Khoshgoftaar et al found that RLS and MRE appeared to have a better predictive quality and MRE produced the best line fit although LAV performed fairly well when outliers were present. They stated that the mis-use of complex statistical procedures may lead to the use of the wrong model for project management and confirmed that least squares did not perform well when the normality assumptions are not valid.

The assessment of quality using metrics has also been applied to Object-Oriented language programs. Henry and Lattanzi [32] have been investigating whether systems written in object-oriented languages were more maintainable than procedural languages and whether they promoted reuse. The study found that the object-oriented systems were reused more often and better reused. Henry and Lattanzi have investigated a suite of metrics for object-oriented language systems (Chidamber-Kemerer). Currently they are investigating the use of quantitative measures to decide when to reuse a component and are developing a tool for collection of these measures. Analysis has shown that use of a combination of metric values can identify problems. They emphasise the lack of object-oriented metrics available for controlling software quality and cost.

Rising [33] evaluated an Information Hiding metric. She compared the ranking of the modules in three different programs with a subjective information hiding value ranking made by the relevant expert for each program. Using Spearman's ρ she found a strong

relationship.

She also investigated whether there was any significant relationship between the information hiding metric and the level of change in a module. Although her case study revealed no correlation, the Mann-Whitney and Chi-squared tests indicated that the metric could identify the modules which were likely to have 'significant' changes.

Brooks [34] identified the need for a well-defined, repeatable and appropriate software process to collect and assess reliability, maintainability and cost with Object-Oriented design.

Some work has been done as a direct result of the REQUEST work. Anderson [35] showed how to use the metrics to establish baselines to allow the metric values obtained in future projects to be interpreted. He described how the analysis of the first data set identified some major problems with the project.

He collected both product and process measures from high level design through to the coding stages of the development process, including information from testing, inspections and documentation. From the comparison of planned and actual effort he found that 20% of the tasks were unplanned which caused an overrun of 30% of the total cost. Also, the tasks which were planned had a 22% overrun. Further investigation allowed him to establish a strategy to reduce the uncertainty of future effort estimates in the short term. This was:

- planned effort for small or large tasks must be carefully reviewed to ensure that no gross under-estimations have been made;

- increase estimates by 5-10% for average or very large tasks to cover any slight under-estimations that occur.

He also showed how to create a basis for changing effort allocation throughout a future project when early phases show an overrun. This can be done by looking at the effort ratios between the different phases.

The ESPRIT research project MERMAID promotes and encourages the use of locally defined models to predict cost since the models are likely to be context dependent. They have confirmed the results of REQUEST that globally defined models which work in a variety of environments are unlikely to deliver accurate estimates. The statistical techniques which were investigated and found useful in the REQUEST project have been incorporated in the MERMAID statistical package. The use of non-parametric techniques has also been advocated in current textbooks (e.g. Fenton's book [36]). The interpretation system described in chapter 7 is currently being used in the SQUID research project.

In conclusion, the work since the thesis appears to be concentrating on the detection of unusual components. There is still a debate about whether intensive statistical analysis using sophisticated tools is better than simple non-parametric techniques. There seems to be a case for both, starting with the simple techniques and then, if necessary, applying the relevant sophisticated techniques.

There is still a major problem in moving between the different levels of the system (i.e. from component to system). Most of the current work on metric validation is being done at the module level, using fault counts as a surrogate for reliability and number of changes for maintainability. There appears to be no evidence of any research addressing the problem of how to move between the levels. In fact, there has even been a step back in the standards arena, ISO 9126 is promoting the McCall et al's principle of set number of general factors with a decomposition despite the REQUEST criticisms of this approach. The ISO standard identifies six factors but does not suggest how these factors can be measured directly.

2.5 Use of Statistics in Analysis of Software Data

Statistics is a tool to summarise raw data into usable information. There are several ways it can do this. For example:

- it provides techniques for collecting, analysing and drawing conclusions from data. This aspect of statistics is used by investigators who attempt

to draw general conclusions from samples or planned experiments;

- it provides a monitoring mechanism for quality control, e.g. in assembly line production;

- it is a succinct means of presenting information. Many news items present their information in terms of statistics, e.g. inflation increased by 3%.

Basic statistical concepts assist in promoting clear thinking about a problem, provide some guidance to the conditions that a problem solution must satisfy and enable an analyst to draw conclusions that could be difficult to obtain by any other way.

Statistics provides data summaries in two main forms - numerical and graphical. Graphical representation is very useful for:

- summarising data;

- providing a simple representation of the results obtained from a more detailed analysis;

- identifying trends or features in a preliminary analysis of the data prior to more formal analysis.

Numerical techniques can serve as objective yardsticks against which the informal conclusions, based on a visual assessment of the information contained in the graphical displays, can be evaluated. They can also provide a more detailed understanding of data by providing more information than can be gained from graphics.

In the software development area, project managers often 'sense' whether the project is running well or not. From experience, they may have some idea of what characteristics of the product are related and which are the most useful for assessing project progress. However, little evidence exists as to how to quantify the expert's intuition. Statistics is useful in the field of software engineering as a means of analysing quantitative

information about what is happening during development and identifying when something goes wrong.

Like data collection, statistics is likely to be more useful to project managers if it can be automated. However, statistics cannot be used in an intellectual vacuum. The requirements of the investigation must be specified before any analysis can be undertaken. In particular, the hypothesis under test or the relationship being investigated must be decided before any statistical analysis can confirm or reject them. For example, if a relationship between size and the number of errors is investigated, the statistical technique of correlation may be used to confirm or reject the existence of a relationship, but it will not identify the correct functional form of the relationship. Selection of the 'correct' statistical technique, therefore, implies that the analyst must understand what information is needed from the data.

The choice of which techniques to use in which circumstance is well-defined in certain fields of use. For example, in the agricultural area, the techniques are well-known and have been used with success for many years. The underlying assumptions of the techniques are known to be approximately valid. In general, these techniques are standard parametric techniques that are based on the assumption that the data is drawn from a Gaussian/Normal distribution, i. e. the data is symmetrical, possess a constant variance and contains a predeterminable number of atypical values. In the software engineering area, the appropriate techniques are not known: in particular, the assumption of a Gaussian data distribution is not necessarily valid.

This highlights the importance of the use of non-parametric techniques in the software engineering area. Non-parametric techniques, [37], make no assumptions about the underlying distribution of the data. They are usually 'pessimistic' techniques. There is added confidence that any relationships which these techniques identify are genuine but they can fail to locate a genuine, but weak, relationship.

Robust techniques, [38], can also be useful since they tend to be insensitive to deviations from the assumptions of the Gaussian distribution and often can be used as an alternative to the classical techniques. Using classical techniques inappropriately can lead to misleading results, therefore the alternative robust and non-parametric techniques

should be used wherever there is any doubt as to the validity of the classical techniques.

This thesis describes how statistics can be used in software engineering and, in particular, concentrates on the use of statistics in the automation of Kitchenham and Walker's work on monitoring and control of projects.

3. Preliminary Analysis

This chapter discusses the type of statistical techniques that were required to formulate, evaluate and use the proposed REQUEST model COQUAMO. Statistical techniques are evaluated in terms of the probable nature and mode of use of COQUAMO, the requirements for model and attribute validation, and the particular problems of software data. The author's requirements were provided by the REQUEST project.

The work in this chapter reflects the project team's view of the COQUAMO model when the author did the work. At that time the monitoring mode of the COQUAMO model (referred to as COQUAMO-2) was viewed as one (or a series of) predictive equations. It was intended to use the attribute values collected during development of a product to predict the final quality of the product. The result of the work reported in the chapter was one of the reasons why the approach to the monitoring mode was changed.

3.1 Aims of Analyses and Implications for Statistical Techniques

In order to identify appropriate statistical techniques, the REQUEST project had to identify the analysis requirements both for the project research workers and for the intended users of COQUAMO.

The research workers required techniques to assist with the formulation and validation of COQUAMO. The process of formulation of COQUAMO involved:

- identifying trends and abnormalities in quality attributes that may be indicative of potential quality problems;
- identifying relationships between quality attributes and final product quality (in terms of quality factors;
- constructing stable predictive models to incorporate any identified relationships.

The process of validating COQUAMO involved:

- establishing the generality of any identified relationships with respect to different environments;
- performing formal evaluation studies to verify particular predictive models.

It was intended that users of COQUAMO would need statistical techniques to calibrate COQUAMO to their own environment. This would have involved:

- techniques to re-estimate the parameters of any predictive model;
- techniques to identify sections of data which should be used in any re-estimation (i.e to select similar projects from a database).

The author's initial requirements were to identify statistical methods that could be used:

1. To identify consistent/general relationships;
2. To identify components with atypical values i.e potential 'anomalies';
3. To construct stable, predictive models.

However, as well as the above requirements an underlying important requirement for the author was to investigate which statistical techniques were valid for analysing software data. The type of the statistical techniques which support the author's aims are discussed below.

3.1.1 Identification of Relationships

In theory there are two types of general relationships that can be used in the formulation of any automatic system of the COQUAMO model - algorithmic relationships and

definition of subgroups with rules for assigning objects to the groups. COQUAMO required a method of formulating relationships between measurements taken during development and final product quality. This suggests correlation and regression techniques when algorithmic relationships are expected, and classification and discriminant analysis when assignment to groups is expected. Discriminant analysis is a technique used to split a large group of data into smaller groups on the basis of certain characteristics.

Algorithmic relationships are formulated as equations. For example, number of errors = module size/lOO. To identify this type of relationship the usual statistical techniques are:

visually: scatter plots(two dimensions only);
and
numerically: regression and correlations.

An example of a relationship which requires assigning ungrouped data to groups is:

Module Group	**Assignment Criteria**
Expected errors per module	*module size*
0	<20
1	21<50
2	51<100

This type of procedure is likely to depend on classification or discriminant analysis.

It may also be important to identify relationships between the attribute measurements collected during development. If several attributes are very strongly correlated among themselves, it is likely that any predictive equation containing all of them would be unstable. Thus, it may be important to identify the underlying dimensionality of the data, and either select a subset of the original attributes, or identify a series of independent linear combinations, which can then be included safely in a predictive equation. This suggests some form of principal component analysis would be useful since it is a technique which transforms a multivariate dataset into a set of new variables

which are linear functions of the original variables and independent of each other.

3.1.2 Identification of Potential Anomalies

It is possible to use measurements collected during development as anomaly detectors (i.e indicators of unusual or atypical events or objects) to allow the software development process to be monitored [20] and [39]. In order to identify anomalies, techniques are required which will locate anomalous components among a group of similar components, (e.g. anomalous modules in a system or sub-system), or identify an attribute value as outside "normal" ranges compared with the values obtained for other similar products.

An example of the first type would be to identify particularly error-prone modules by looking at the error rate per module for a group of modules. An example of the second type would be to identify the testing effort used to produce a product of given size and application type as unusually large or small compared to other similar products.

In the first case a relationship between module size and number of errors might be established and analysis of residuals used to identify anomalous modules. In the second case, a measure of the central location of a group of test effort values from various similar products (e.g. mean, median, mode) plus a measure of the expected variability about the centre of location (e.g. standard deviation, range) is required in order to establish whether the testing effort for a new product is an any sense abnormal.

Thus, theoretically anomaly detection is likely to involve residual analysis and the identification of "normal" values (which for a Gaussian or Normal population would imply, for example, the mean plus or minus two standard deviations).

3.1.3 Stable Predictive Models

Any predictive models identified will need to be evaluated through cross-validation

studies. This might be done, for example, by using a sub-set of the data to specify a statistical relationship. This relationship is then validated against the remaining data. More detailed descriptions of cross-validation techniques are given in Mosteller and Tukey, [38].

3.1.4 Statistical Techniques for COQUAMO-2 users

The statistical techniques selected initially were based on the idea that COQUAMO-2 would be a predictive, general model. This would require the users of COQUAMO-2 to be able to calibrate the model to their own organisation. Since the nature of COQUAMO-2 has changed, the need for techniques to enable calibration of COQUAMO-2 was removed. Therefore, the author did not investigate any techniques which would be specific to a COQUAMO-2 user.

3.2 Implication of the non-Gaussian nature of Software Data

The nature of software data is very important when considering which statistical techniques are appropriate for the analysis of software data for the COQUAMO model. The REQUEST project team [1] had already identified that software data was highly skewed, contained a relatively high number of outliers, and often showed evidence of relationships between the mean and the variance. Such data is obviously non-Gaussian which suggests that statistical techniques, which assume an underlying Normal distribution, must be used with caution.

A number of techniques may be used to avoid problems of non-Normality:

- **data transformation**

  It may be possible to transform the data to make it sufficiently close to Gaussian that the classical statistical techniques can be used. Often log transformations are used (c.f. cost models such as COCOMO, [3]).

  Problems may occur if many variables need to be included in a predictive

model, and each variable requires a different transformation. This would make interpretation of any observed relationships very difficult.

- **robust techniques**

  It may be possible that techniques which are robust, with respect to departure from normality, can be used. Examples of this range from using medians rather than means as statistics of central location, to using statistics based on the jackknife technique or using robust variance-covariance matrices when performing any multivariate techniques such as principal components.

- **non-parametric techniques**

  It may sometimes be useful to techniques which are independent of the underlying distribution of the data. Such techniques tend to be pessimistic in the sense that they may fail to locate genuine relationships, but relationships which they do identify can be treated with some confidence. This has been suggested by the researchers at the US Rome Air Development Centre as the only suitable analysis technique for their data, [7].

- **other distribution-based techniques**

  Even though the data cannot be assumed normal, it may follow another distribution, for example the exponential or the Poisson distributions. If the particular distribution can be determined, statistical techniques appropriate to that distribution can be used.

The rest of this chapter describes the analysis and the results of two data sets using some of the above techniques that the author believed were appropriate to meet the requirements set by the REQUEST project. The only technique for avoiding the problem of non-normality which was not investigated at all is "other distribution-based" technique. Since the work on the author's thesis has been completed, further work has been done which investigates the use of the Poisson and Negative Binomial distributions [40].

3.3 Determination of Relationships among Software Attributes

The aim of this section of the thesis was to identify relationships between attribute measurements collected during the development of the product and subsequent change-proneness. The author used datasets 1 and 2 because they comprised of data collected from two implementations of the same subsystem. The author expected that if any general relationships existed, they would be found in both of the datasets. Also, a range of different attributes measures had been collected, many of which were likely to be of use to COQUAMO. These attribute measures were:

- Machine code Instructions (MCI)
- Lines of code (LOC)
- Fan-out (FO) [i.e. the number of programs called by the program]
- Data Items (DI) [i.e. the number of global data items accessed by the program]
- Parameters (PAR) [i.e. the number of arguments on the program's interface]
- Number of distinct operators (n1)
- Number of distinct operands (n2)
- Total number of operators used (N1)
- Total number of operands used (N2)
- McCabe's cyclomatic complexity V (V(G))
- Changes (CHG)

Some combinations of the basic metrics have been used:

n = n1 + n2

N = N1 + N2

HE = n1*N2*N*$\log_2$(n)/2*n2 (Halstead's E)

A full description of the datasets 1 and 2 is given in section 1.3

Relationships can either be bi-variate or multivariate in nature. Different statistical techniques are required for each type.

3.3.1 Bi-variate Relationships

The author used correlation coefficients and scatter plots to investigate the nature of two-dimensional relationships. Although the identification of error-prone and change-prone components are unlikely to be based on the value of a single metric, the exercise was useful in highlighting some potential problems with the nature of the data and allowing them to be investigated without the additional problem of applying a complex statistical technique which may not be valid.

A correlation coefficient is a measure of the extent of association between two variables. The usual correlation coefficient used is Pearson's correlation coefficient, r. This coefficient assumes an underlying Normal distribution which is unlikely to exist with software engineering data, therefore it should be used with caution. The correlation coefficients for the two datasets are given in Tables 3.1 and 3.2 below. The significance of the correlations are shown in the tables by the following key:

*	$p<0.05$
**	$p<0.01$
***	$p<0.001$

$p<0.05$ means that there is a 95% chance that the significant relationship did not occur by chance, similarly $p<0.01$ implies a 99% chance and $p<0.001$ implies a 99.9% chance.

	MCI	LOC	FO	DI	PAR	n1	n2	N1	N2	VG	CHG
MCI	1										
LOC	0.94 ***	1									
FO	0.44 *	0.46 *	1								
DI	0.62 ***	0.80 ***	0.33	1							
PAR	0.40 *	0.34	0.16	0.09	1						
n1	0.72 ***	0.70 ***	0.83 ***	0.48 *	0.45 *	1					
n2	0.86 ***	0.78 ***	0.67 ***	0.45 *	0.37	0.88 ***	1				
N1	0.91 ***	0.97 ***	0.28	0.79 ***	0.28	0.55 **	0.87 ***	1			
N2	0.94 ***	0.98 ***	0.32	0.78 ***	0.28	0.58 **	0.72 ***	1.00 ***	1		
VG	0.87 ***	0.91 ***	0.49 **	0.89 ***	0.52 **	0.77 ***	0.78 ***	0.83 ***	0.84 ***	1	
CHG	0.66 ***	0.55 **	0.61 ***	0.31	0.54 **	0.78 ***	0.71 ***	0.44 *	0.46 *	0.88 ***	1

Table 3.1 SS1 - Pearson's Correlation Matrix

	MCI	LOC	FO	DI	PAR	n1	n2	N1	N2	VG	CHG
MCI	1										
LOC	0.98 ***	1									
FO	0.58 ***	0.65 ***	1								
DI	0.78 ***	0.79 ***	0.69 ***	1							
PAR	0.34 *	0.33 *	0.06	0.35 *	1						
n1	0.87 ***	0.88 ***	0.77 ***	0.81 ***	0.30 *	1					
n2	0.93 ***	0.91 ***	0.61 ***	0.77 ***	0.37 *	0.92 ***	1				
N1	0.98 ***	0.97 ***	0.56 ***	0.76 ***	0.34 *	0.85 ***	0.93 ***	1			
N2	0.99 ***	0.97 ***	0.56 ***	0.77 ***	0.34 *	0.85 ***	0.91 ***	1.00 ***	1		
VG	0.94 ***	0.93 ***	0.50 ***	0.76 ***	0.38 *	0.79 ***	0.86 ***	0.93 ***	0.93 ***	1	
CHG	0.72 ***	0.67 ***	0.47 **	0.46 **	0.19	0.65 ***	0.73 ***	0.70 ***	0.69 ***	0.69 ***	1

Table 3.2 SS2 - Pearson's Correlation Matrix

Software data is unlikely to follow a Normal distribution therefore the author also calculated a non-parametric correlation coefficient. Non-parametric techniques do not make assumptions regarding the underlying distribution of the data. The author investigated the use of two different non-parametric techniques, contingency correlation C and Spearman's ρ [37]. Spearman's ρ cannot cope with a large number of tied values therefore it could not be used with this data. The contingency coefficient C is particularly good for categorical data and for data which has a lot of tied values. The contingency coefficient C is interpreted in the same way as the parametric coefficients with the only difference being that it can only take a value between zero and one

inclusively. This means that a positive or negative relationship cannot be detected, only a significant correlation. The correlation coefficients for datasets 1 and 2 are given in tables 3.3 and 3.4 respectively.

	MCI	LOC	FO	DI	PAR	n1	n2	N1	N2	VG	CHG
MCI	1										
LOC	0.58 ***	1									
FO	0.43 *	0.53 ***	1								
DI	0.85 ***	0.58 ***	0.53 ***	1							
PAR	0.26	0.25	0.30	0.21	1						
n1	0.62 ***	0.62 ***	0.48 **	0.54 ***	0.26	1					
n2	0.65 ***	0.65 ***	0.48 **	0.58 ***	0.25	0.62 ***	1				
N1	0.71 ***	0.58 ***	0.43 *	0.65 ***	0.25	0.62 ***	0.65 ***	1			
N2	0.65 ***	0.65 ***	0.53 ***	0.58 ***	0.25	0.62 ***	0.71 ***	0.65 ***	1		
VG	0.58 ***	0.58 ***	0.43 *	0.58 ***	0.44 **	0.54 ***	0.62 ***	0.58 ***	0.58 ***	1	
CHG	0.48 **	0.48 **	0.53 ***	0.49 **	0.25	0.44 **	0.58 ***	0.48 **	0.58 ***	0.38 *	1

Table 3.3 SS1 - Contingency C Correlation Coefficients

	MCI	LOC	FO	DI	PAR	n1	n2	N1	N2	VG	CHG
MCI	1										
LOC	0.70 ***	1									
FO	0.37 *	0.50 ***	1								
DI	0.51 ***	0.51 ***	0.29	1							
PAR	0.34 *	0.34 *	0.10	0.15	1						
n1	0.61 ***	0.65 ***	0.41 **	0.48 ***	0.19	1					
n2	0.71 ***	0.70 ***	0.37 *	0.51 ***	0.34 *	0.60 ***	1				
N1	0.70 ***	0.63 ***	0.37 *	0.48 ***	0.34 *	0.54 ***	0.70 ***	1			
N2	0.70 ***	0.63 ***	0.37 *	0.48 ***	0.34 *	0.54 ***	0.70 ***	0.70 ***	1		
VG	0.57 ***	0.51 ***	0.20	0.51 ***	0.42 **	0.41 **	0.57 ***	0.57 ***	0.57 ***	1	
CHG	0.45 **	0.50 ***	0.21	0.45 **	0.27	0.42 **	0.45 **	0.45 **	0.45 **	0.52 ***	1

Table 3.4 SS2 - Contingency C Correlation Coefficients

In general the contingency correlations are lower than the parametric correlations, which was expected. However, they are still significant.

There are strong correlations among the basic Halstead measures. This suggests that some caution should be used exercised when interpreting the meaning of any synthetic metric based on basic measures. Also, the high correlation between the these measures and the size measures suggests that the Halstead measures are equivalent to measures of size.

Some of the measures mentioned will be available earlier in the development process than module size and the number of times a module was changed, therefore they may be useful as early indicators of size or change-proneness if significant correlations exist. Some of the measures, such as modules called and data items used appear to be correlated to both size and number of changes. These correlations were found in both datasets, so may be fairly general in nature. Since they are also available early in the software design process they might be useful as early indicators.

However, it should be noted that not all relationships were found to be consistent. For example, the number of parameters was significantly correlated with changes in dataset 1 but not for dataset 2. This implies that some metrics will be unsuitable for inclusion in a general predictive model.

Scatter plots are useful for visually investigating the nature of relationships between two variables. The advantages of scatter plots are that they are simple and easy to use without the need for any underlying assumptions to be made about the data.

As was mentioned earlier, (see section 3.2), data transformation may be useful as a technique to deal with non-normality in the data. Figure 3.1 shows an example of some of the problems with the nature of software data and figure 3.2 shows how useful the logarithmic transformation can be in reducing these problems. Figure 3.1 is the plot of size, measured in lines of code against McCabe's Complexity V, V(G). Intuitively it was expected that as size increases, the value of McCabe's Complexity V increases because it is derived from the number of conditions and loops in a program. The more conditions and loops the more statements are needed to code them. This plot shows a strong positive relationship but evidence of heavy, positive skewing. Skewing can be detected by a high density of modules in a particular range of values and, in this case, the skewing is positive because there is a tendency towards the smaller values. There are also more anomalies than would normally be expected from 27 data points. It is also possible that the increased number of apparent anomalies is caused by an increase in the variation of the data. An increasing variance is usually overcome by applying a logarithmic transformation.

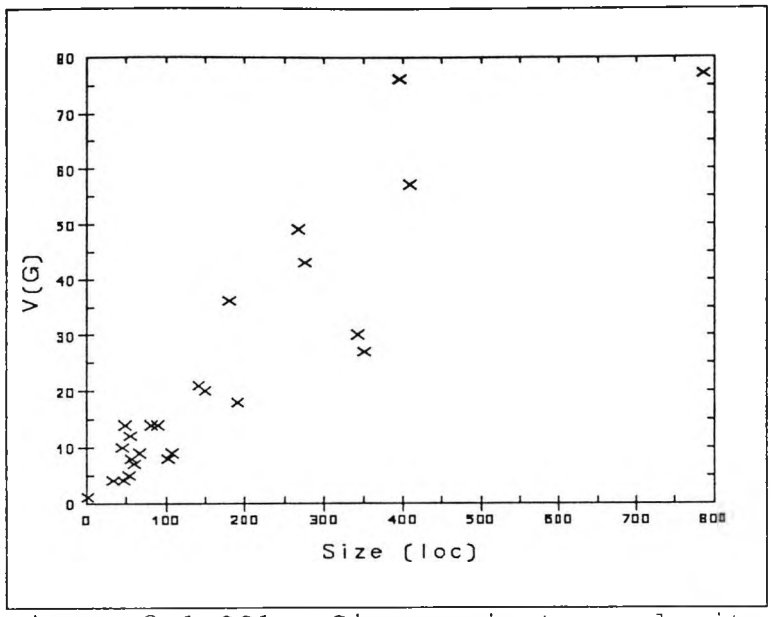


Figure 3.1 SS1 - Size against complexity

The author applied a logarithmic transformation in an attempt to stabilise the variance. Figure 3.2 shows the data after the measures have undergone a natural logarithmic transformation. The relationship between V(G) and size is clearer, with the effects of skewing and anomalies greatly reduced. Therefore it appears that more confidence can be placed in the detection of the true underlying trend between the metric values of McCabe's Complexity V and the size of the module if the data is transformed.

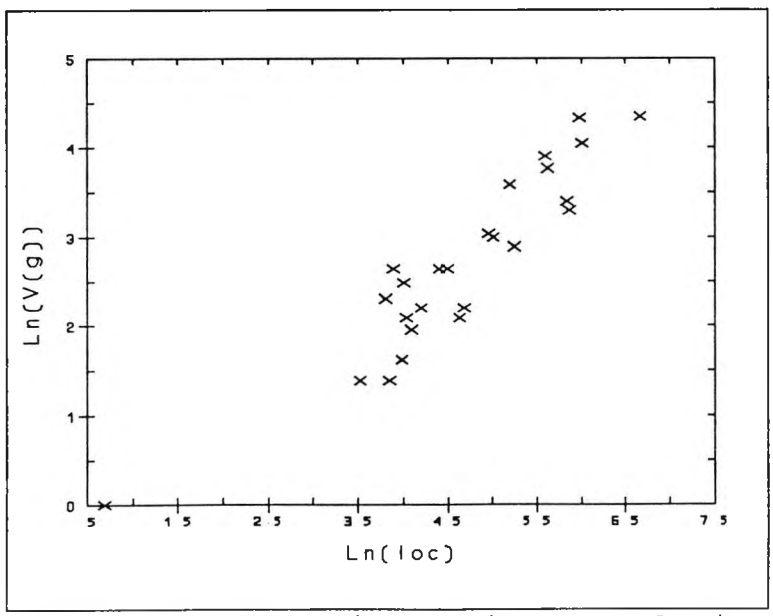


Figure 3.2 SS1 - Size against complexity using natural logarithm transformation

From Figure 3.3, it can be seen that the scatter plot for dataset 2 does not show any evidence of bivariate outliers, only one large value. It does, however, appear to have positive skewing and evidence of an increasing variance similar to that of dataset 1.

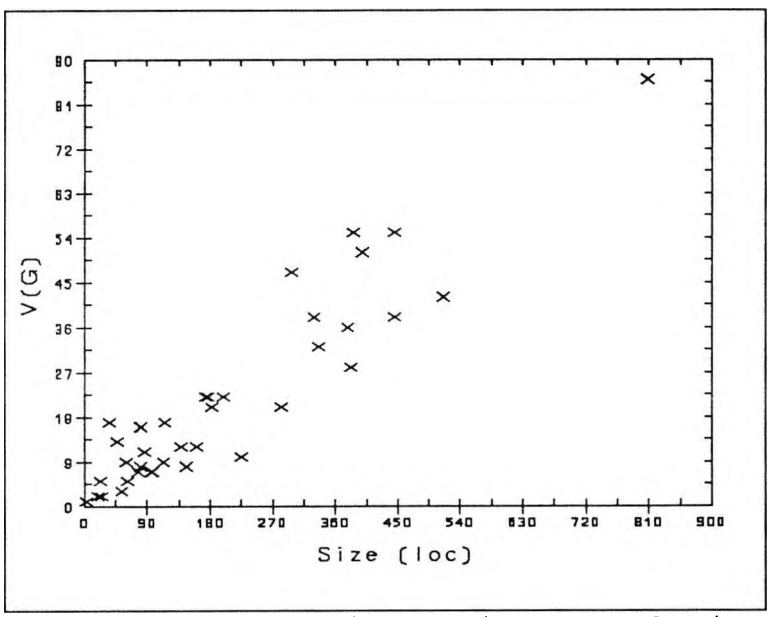


Figure 3.3 SS2 - Size against complexity

Plotting the log of the dataset 2 measures (Figure 3.4), as expected reduced the effect of the increasing variance and the positive skewing. It is interesting to note that the scatterplot now appears to have at least one outlier. If a logarithmic transformation had not been used, this anomalous module would have gone undetected in the plot. This highlights a problem of visual representation of the data where a choice of scale, which appears to be the optimum one, actually obscures subjective detection of anomalies. The transformation has reduced the problem of the increasing variance and the anomalies.

Due to the high correlation between the size of the module and McCabe's V(G), it was doubtful whether there was any benefit from using McCabe's V(G) rather than the size of the module to identify change-prone modules. To investigate this, the author normalised the McCabe's V(G) measure with respect to size and plotted the resulting measure against the number of changes per module. After normalisation there was no longer a significant relationship. The correlation coefficient dropped from 0.68 ($p<0.001$) for the raw data, to 0.21 which was not significant for the normalised

measure. In addition a plot of changes against the size of the module (Figure 3.5) shows the same basic relationship as changes against McCabe's V(G) plot, and identifies the same anomalous module.

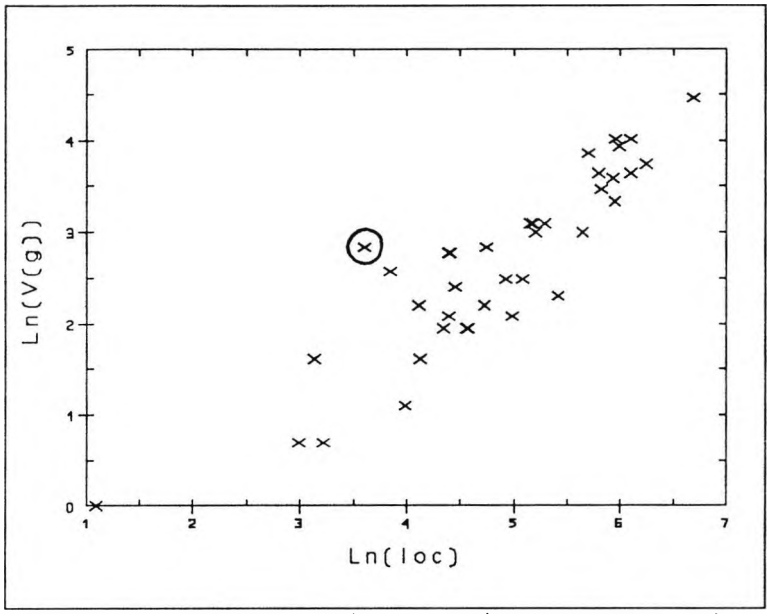


Figure 3.4 SS2 - Size against complexity using natural logarithm transformation

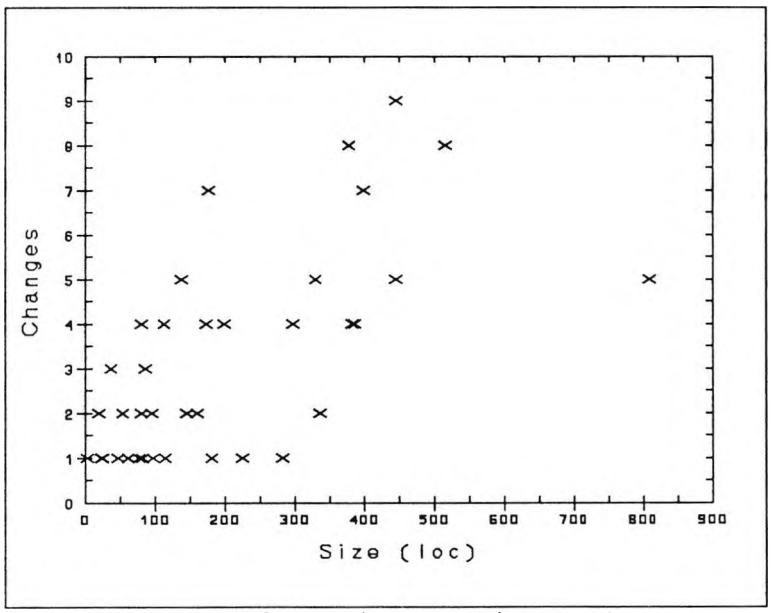


Figure 3.5 SS2 - Size against changes

For dataset 2 data, the correlation for the normalised McCabe's V(G) against number of changes was also insignificant. Thus the two datasets indicate that the size of a module can provide as much information for identifying or predicting change-prone modules as McCabe's V(G). It is possible however, that McCabe's V(G) measure may

be available from detailed designs rather than from the code and thus provide information earlier in the development process than a code-based size measure. McCabe's V(G) is also important from the viewpoint of test management since it provides a simple measure of the number of test cases needed to execute each branch of code in a module.

3.3.2 Multivariate Relationships

The author investigated the use of regression as a technique for determining whether design and code-based metrics can indicate whether a module is likely to require a large or a small number of changes. Regression is a technique which can be used to identify a relationship between a single variable (the dependent variable) and a set of one or more other variables (explanatory variables), [41]. The dependent variable is the variable being estimated or predicted in a regression (e.g number of changes) and the explanatory variables are the variable being used to estimate the dependent variable.

Initially an Ordinary Least Squares (OLS) regression was used to obtain an indication of both the nature of the relationship between all the metrics and the number of changes (CHG), and the problems likely to exist with software data.

The significant regression obtained with data set 1 was:

$$CHG = 0.042MCI - 0.075N + 0.00002HE \quad (1)$$

where CHG is the number of changes, MCI is the number of machine code instructions in bytes, N is the total number of operators and operands and HE is Halstead's E. This equation had an adjusted R^2 of 0.57. An adjusted R^2 is a measure of the proportion of variation in the dependent variable that is explained by the independent variables in a multiple regression (more than one explanatory variable) adjusted for the number of explanatory variables in the regression. The adjusted R^2 is used instead of the unadjusted because as the number of variables in the regression increase the value of the unadjusted R^2 increases, regardless of whether the variables have a significant effect on the dependent variable.

Equation (1) suggests that a combination of the values for the number of machine code instructions (MCI), the total number of operators and operands (N) and Halstead's E (HE) are a significant predictor for the number of changes (CHG). There are some doubts as to the validity of this equation as a predictive equation of change-proneness because of the negative sign for N. The negative sign is saying that if N increases this reduces the number of changes the module will require. Therefore from a predictive viewpoint the above equation does not conform with the REQUEST's team intuition. However, the high correlation between the explanatory metrics themselves might be causing this negative impact. This might indicate that not all of the explanatory variables, which the regression has shown as having a significant effect on change-proneness, are having a significant independent effect.

The residual plot (Figure 3.6) showed five potential anomalies. A point was assumed to be anomalous if the value was extreme with respect to either the predicted number of changes or the residual value. A residual is the difference between an actual observed data value and the equivalent value predicted by a regression model. A residual plot is used to identify any systematic patterns in a group of residuals to evaluate the adequacy of the regression model [41].

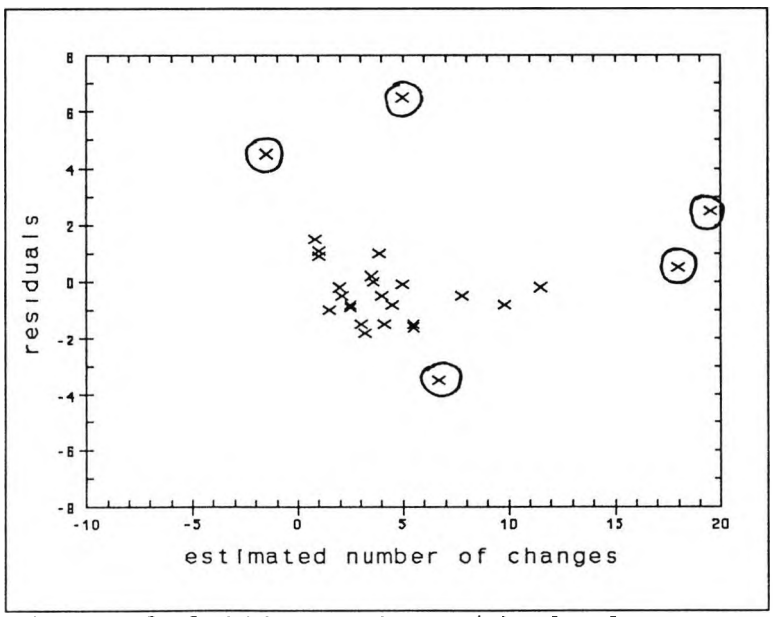


Figure 3.6 SS1 - OLS Residual Plot

The removal of the anomalies reduced the adjusted R^2, which indicates that the

anomalies were contributing to the significance of the equation, i.e. the regression was being noticeably altered by one or two data values.

The metric "lines of code" was now also significant in the equation and all the coefficients were marginally significant. Therefore, the anomalies are obviously affecting the detection of any relationship and it is likely the non-Gaussian nature of the data is contributing to the untrustworthy results obtained from the OLS regression.

The results obtained from dataset 2 were totally different from dataset 1:

$$CHG = 0.25FO - 0.53DI + 0.09V(G) \quad (2)$$

where CHG is the number of changes, FO is fan-out, DI is the number of data items and V(G) is McCabe's complexity V. This equation had an adjusted R^2 of 0.57. Instead of being totally code attributes, the predictive equation is based predominantly on design attributes. Again, there is an unexpected negative sign in the equation.

However, the residual plot did not show any of the attributes to be having an independent effect on change-proneness. One potential anomaly was identified by the residual plot (Figure 3.7). When this was removed the measure of fan-out (FO) was not significant.

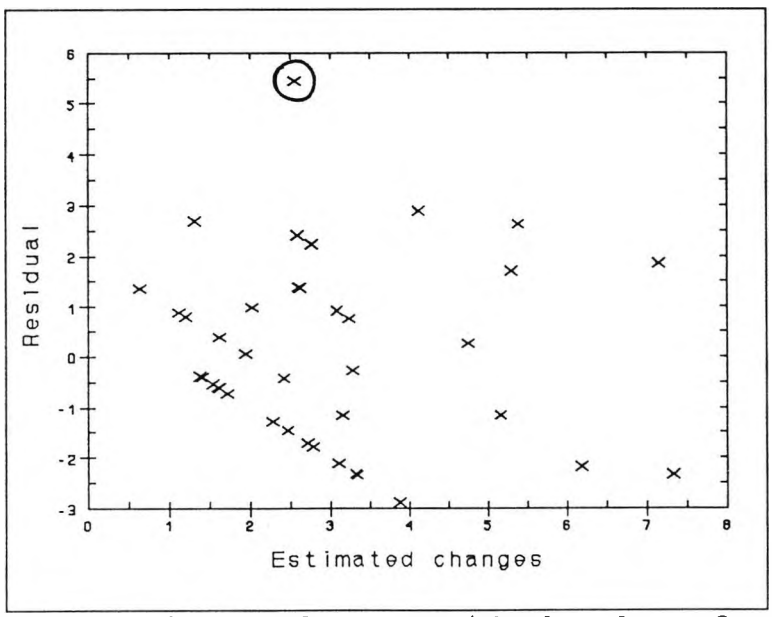


Figure 3.7 SS2 - Residual plot for multivariate regression

The use of complex multivariate equations may not be necessary since there was little reduction in the value of R^2 for both of the equations when only a single variable was used in the regression. The best single variable regression for dataset 1 was:

$$CHG = -2.097 + 0.234n1 \quad (3)$$

with an R^2 of 0.582. The residual plot (see Figure 3.8) shows bias in the equation since it indicates that a curved fit would be more appropriate. It also shows four outliers, one of which is different to those shown with the multivariate regression. This highlights a problem with the use of residual plots for outlier detection since the outliers are not independent of the equation chosen and there are many equations which would be equally valid.

The results of dataset 2 are similar to dataset 1 in that the best single variate equation had a similar R^2 to the complex multivariate equation:

$$CHG = 1.190 + 0.007MCI \quad (4)$$

with an R^2 of 0.524. However, these results are inconsistent with dataset 1 results with respect to the best single variable. The residual plot (Figure 3.9) shows one outlier, which was different to the multivariate equation outlier.

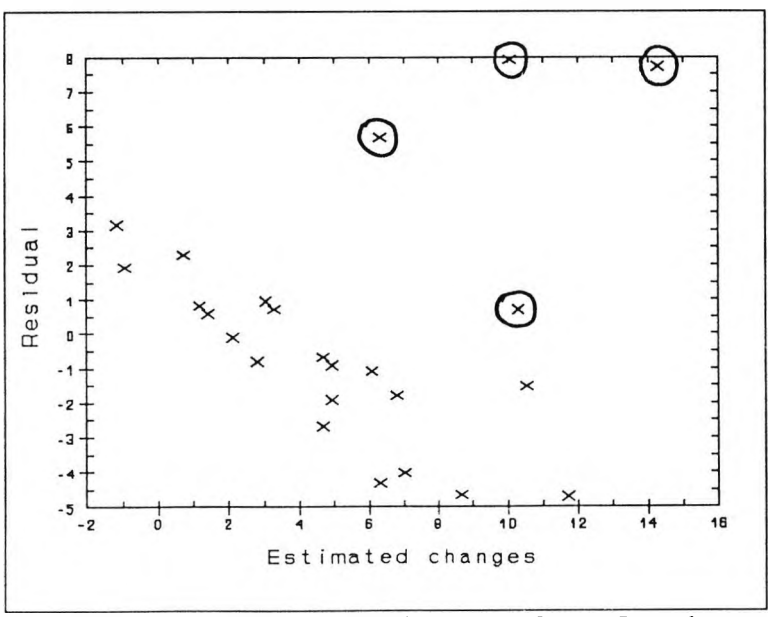


Figure 3.8 SS1 - Residual plot for best single variate regression

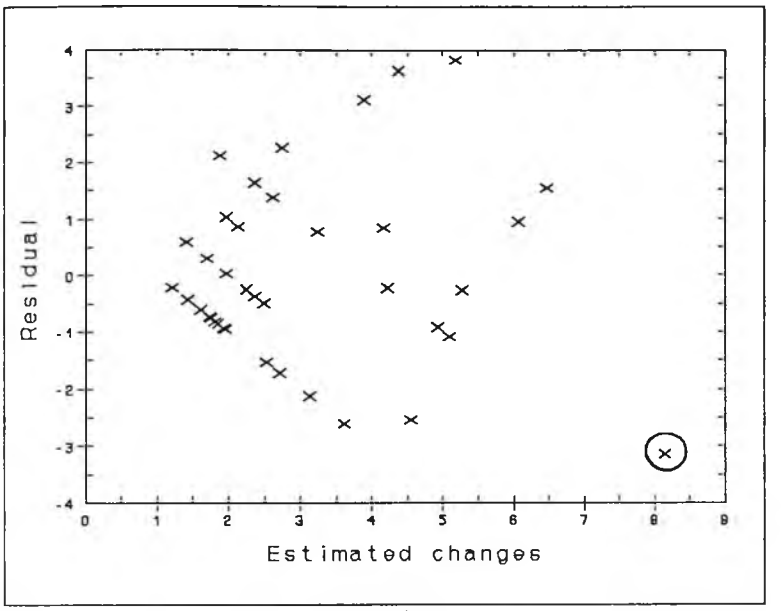


Figure 3.9 SS2 - Residual plot for best single variable regression

The conclusion from applying OLS regression is that there does not appear to be any stable predictive equation for change-proneness with the given attributes since both the datasets had completely different equations. In addition, both multivariate equations appear to be unstable with coefficients which do not conform with the team's intuition.

3.3.3 Problems with OLS regressions

The OLS regression technique assumes that the data is from a normal distribution. Software datasets available to the REQUEST project did not follow this distribution. The OLS regression technique has been shown to be highly influenced by outliers and may also be influenced by the increasing variance. In this section the author has investigated some techniques which may remove the effect of these problems and so allow an identification of a relationship if one exists.

3.3.3.1 Transformations

One of the scatter plots (Figure 3.1) shows evidence of an increasing variance and, since an increasing variance can cause insignificant coefficients to look significant, the author

decided to apply a natural logarithmic transformation to the data.

With both data sets, taking natural logarithms of the data dramatically altered the result obtained. With data set 1 none of the previous attributes shown to be significant are now significant. The only significant attribute now was the number of parameters. Again no confidence can be put in this result because its correlation with change-proneness is low.

With data set 2, the effect of the transformation was to show that only McCabe's V(G) had a significant effect on change-proneness.

This implies that the predictive equations using the available quality indicators were not stable. It is likely that these quality indicators are not good predictors of change-proneness.

3.3.3.2 Robust Regressions

Another possible method of dealing with the characteristics of software data is by using a robust regression method. Instead of reducing the effect of the characteristics present in software data, it attempts to be less sensitive to them. This should mean that deviations from the Gaussian assumptions required by the OLS regression will not cause the regression technique to give invalid results.

The two robust regressions the author investigated were a 'least absolute residual' regression and a 'one-step Andrew's' robust regression [38]. The 'least absolute residual' regression minimises the sum of the absolute values of the residuals and the residuals can be used as input into the 'one-step Andrew's'.

The raw original data was used for the robust regression. The significant regression equation was:

$$CHG = 0.013MCI - 0077N + 0.00002\ HE \qquad (5)$$

where CHG is the number of changes, MCI is the number of machine code instructions, N is the total number of operators and operands and HE is Halstead's E. This equation had an adjusted R^2 of 0.92. However, if this equation is compared to the OLS equation (1) there appears to be very little difference except that the R^2 is 0.25 greater. This is unexpected since there are anomalies present which have been shown to have dramatically altered the OLS regression results and the robust regression is supposed to be insensitive to their presence.

The residual plot (Figure 3.10) indicated four potential anomalies. The regression coefficients, after the three anomalies were removed from the analysis, were all totally insignificant but the equation still had a high adjusted R^2 of 0.83. This implies that it was not the majority of the points showing a relationship with change-proneness but only the three anomalies. Thus it appears that with this data the robust regression is not in fact resilient to anomalies and does not produce a more trustworthy predictive equation. In fact, the robust regression, instead of reducing the problems that were occurring with the use of OLS regression, appears to increase them. The robust regression including the anomalies, has shown 25% more confidence in the predictive power of the equation. Therefore, the technique appears to produce more optimistic results than OLS, without any identified (at present) underlying cause.

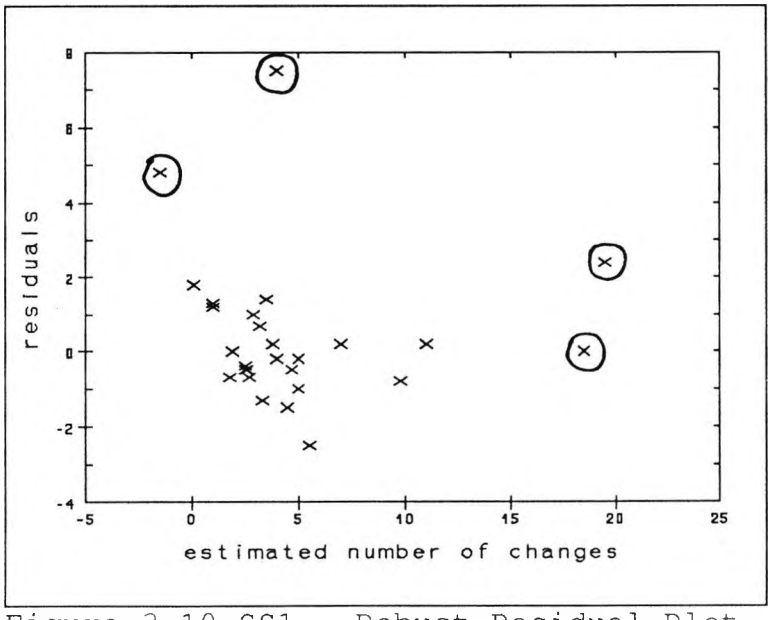


Figure 3.10 SS1 - Robust Residual Plot

The equation obtained using robust regression on dataset 2 was similar to that obtained for data set 1, equation (5). However, only the machine code instructions coefficient was significant (although only at the 80% level) with an adjusted R^2 of 0.65.

The residual plot (Figure 3.11) highlighted five potential anomalies. After they were removed and the regression repeated all the coefficients were insignificant.

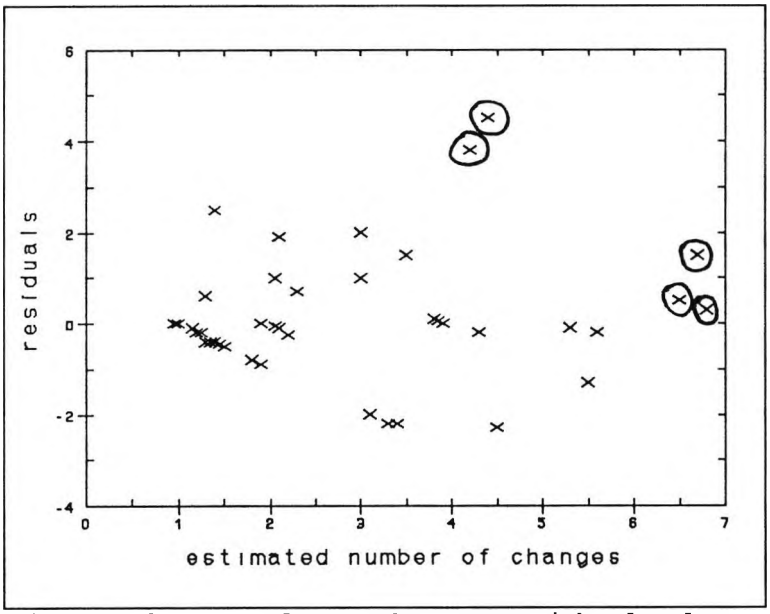


Figure 3.11 SS2 - Robust Residual Plot

In conclusion both data sets have shown that the robust regression was sensitive to anomalies and has not provided reliable information about the nature of the relationship of the metrics to the number of changes in the presence of these anomalies. Work undertaken since this, [36] has shown a simple Theil's robust regression [42] was useful with software data. The only problem identified with Theil's regression is that the calculations become very cumbersome with large amounts of data therefore a powerful computer would be required.

3.3.4 Problems with Data Characteristics

The previous sections have shown that problems exist with the nature of the attribute values not just the statistical techniques. Some of the identified problems are addressed

in this section.

3.3.4.1 Obtaining Independent Measures

The previous section has shown that the attributes are highly correlated. The presence of highly correlated attributes can cause insignificant results to appear significant and result in unstable equations. The author investigated two methods in an attempt to overcome this problem. The two methods were:

(1) normalisation of the data;
(2) use of principal components to produce independent variables.

(1) Normalisation of the data

One of the problems when using regression on this data is that instead of the attributes being independent of each other most of them are correlated with size. Therefore, the author decided to remove the effect of size from all the attributes, that is to "normalise" the data with respect to size by dividing the value of each attribute for each module with a measure of size for that module. The dependent variable (CHG) was also normalised. The reason for normalising changes was to investigate the relationship of the rest of the attributes on the number of changes, without the influence of size since other studies have shown that size affects the number of changes.

To investigate what power function of size was required for the normalisation of the attributes, the logarithmic transformation of each attribute was plotted against the log of the size. The coefficient of the regression line through this plot indicates the power of the relationship [38]. Most of the coefficients were approximately one, therefore the attributes were simply divided by the module size in lines of code.

With dataset 1, using OLS regression, the regression equation was:

$$NCHG = -0.004 + 0.378NPAR + 0.122Nn1 \quad (6)$$

with an R^2 of 0.985. However, the normalised number of parameters (NPAR) and the normalised number of distinct operators (Nn1) are correlated with an r=0.752 although the individual coefficients are highly significant. The residual plot (Figure 3.12) showed no major outliers but did indicate the presence of an increasing variance for the larger predicted values for the normalised changes.

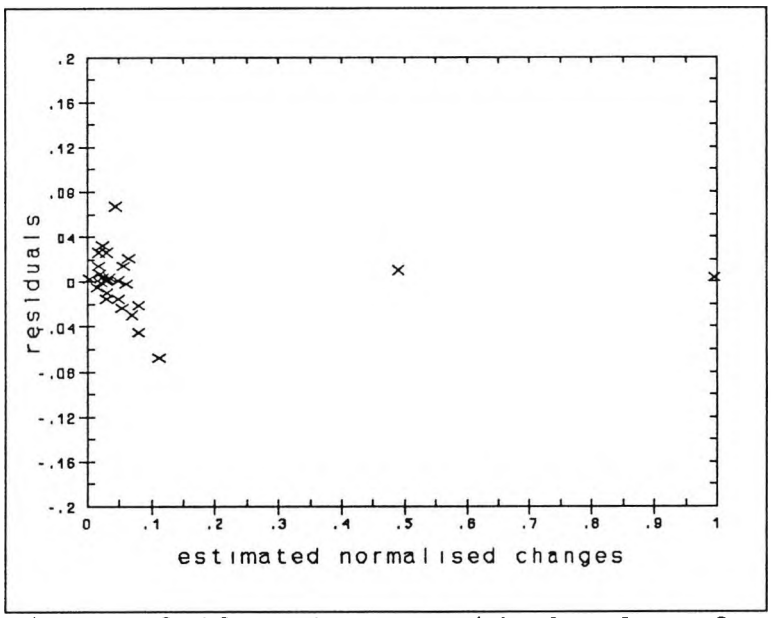


Figure 3.12 SS1 - Residual Plot for Normalised Equation

The normalised measure of distinct operators is significant in the equations for both the datasets. However, the other measures are different and have different signs. The regression equation for dataset 2 was:

$$NCHG = 0.002 + 0.212Nn1 - 0.594NFO - 0.083Nn2 \quad (7)$$

with an R^2 of 0.913.

The residual plot (Figure 3.13) highlighted four major outliers (the cross at '1' represents three module values). They related to very small modules which after the normalisation transformation produced relatively large attribute values which have a dramatic effect on the regression.

The regression was re-run after the outliers were removed. Now none of the coefficients

were significant which suggests that the original relationship was based on one or two influential modules only.

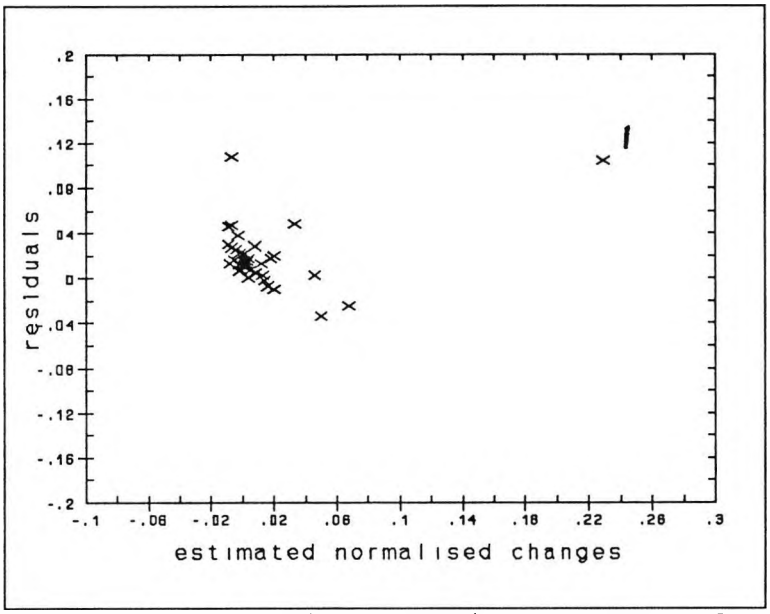


Figure 3.13 SS2 - Residual Plot for Normalised Equation

The conclusion from normalising the data is that no stable predictive equations could de detected.

(2) Use of Principal Components

The technique of Principal Components transforms the original data into new variables which are linear combinations of the old variables [43]. The new variables are not correlated with each other. In this way the original attributes can all be used, if they have a significant effect on change-proneness without the problem of correlated variables.

Principal Components cannot be based on the covariance matrix because it is heavily influenced by the scale of the factors. This means that attributes like Halstead's E dominate the components because their values are so much larger than the other attributes. Therefore principal components should be based either on the correlation matrix or the covariance matrix after the variables have been standardised to have a zero

mean and unit variance.

With dataset 1, the significant OLS regression was:

$$CHG = 5.22 + 1.30PRIN1 + 2.28PRIN2 \quad (8)$$

with an adjusted R^2 of 0.66. The third principal component was insignificant.

With dataset 2, only the first principal component was significant:

$$CHG = 3.05 + 0.61PRIN1 \quad (9)$$

with an adjusted R^2 of 0.43.

Although both regressions do not have any anomalies in the residuals, the two datasets give conflicting results. The first principal component for both datasets consisted of all the measures except the number of parameters. The second principal component differs between the two datasets, the second principal component for dataset 1 consists of fan-out and the number of distinct operators, whereas the second principal component for dataset 2 consists of the number of parameters and fan-out. Also, even if they were producing the similar results it is difficult to interpret the regression equation because the principal components do not separate into any interpretable high level factors, for example the first component all the code measures and the second component all the design measures. Table 3.5 shows the correlations between the first two principal components and the raw data[1]

[1]The correlations for the principal components have been calculated using slightly different Halstead E values due to problems with the statistical package. The author believes that the differences cause little impact on the results.

	Dataset 1		Dataset 2	
Measure	Comp 1	Comp 2	Comp 1	Comp 2
MCI	0.964	-0.057	0.987	-0.043
LOC	0.980	-0.149	0.982	0.017
V(G)	0.928	0.049	0.933	-0.134
n1	0.798	0.567	0.922	0.197
n2	0.870	0.378	0.951	0.010
n	0.869	0.450	0.958	0.071
N1	0.931	-0.352	0.979	-0.065
N2	0.948	-0.306	0.978	-0.066
N	0.940	-0.332	0.979	-0.065
DI	0.743	-0.303	0.841	0.116
PAR	0.416	0.251	0.380	-0.763
FO	0.543	0.693	0.666	0.560
HE	0.881	-0.388	0.854	-0.114

Table 3.5 Correlations between Principal Components and Raw Data

3.3.4.2 Division into Design and Code-based Metrics

The previous regressions did not show evidence of any consistent relationships. Therefore, the author decided to investigate whether regression results could be improved by splitting the explanatory variables into design-based and code-based metrics. Since the attributes have come from different phases of the development, it is feasible to assume that they will have certain characteristics which are unique to their development phase. It may also be possible to investigate whether the design-based attributes can be used as early indicators of module size and/or the change-proneness of

the modules. Similarly, the author investigated whether code-based attributes can be used as indicators of change-proneness. The design-based attributes were FO, DI and PARS, with the code-based attributes MCI, LOC, n, N, V(G) and HE.

To reduce the effect of the increasing variance and the relatively large number of outliers, the author applied the logarithmic transformation on the attributes. The attributes were not adjusted for size.

Since there were six code-based metrics which were all highly correlated, it was decided to use their first principal component when investigating the relationship between design attributes and size. This removed the need to decide which was the most appropriate size attribute.

With dataset 1 an OLS regression showed the following significant linear relationship of design attributes against size (i.e. the first principal component).

$$\text{Size} = -4.70 + 1.08\text{FO} + 1.37\text{DI} \tag{10}$$

with an adjusted R^2 of 0.58. The residual plot indicated a tendency for the above equation to under-estimate size.

With dataset 2, the equation was:

$$\text{Size} = -6.27 + 1.10\text{FO} + 0.86\text{PAR} \tag{11}$$

with an adjusted R^2 of 0.79. The number of parameters is a significant attribute with dataset 2 but not with dataset 1. Although all the coefficients were significant, the most significant was the constant.

Since the datasets did not display the same general underlying relationship, no conclusions could be drawn about general relationships, only about specific relationships.

The author investigated the effect of the design-based attributes on change-proneness.

With dataset 1, the resulting equation was:

$$CHG = 0.38FO + 0.48PAR \quad (12)$$

with an adjusted R^2 of 0.59.

There existed a conflict between the two techniques used to evaluate the significance of the coefficients in this equation, the t-statistic and the residual plots. In the results reported so far, the two techniques have complemented one another. If the t-statistic was borderline the residual plot indicated whether the attribute was having an effect. However, in this equation the t-statistic indicated that the FO coefficient was marginally more significant than the PAR coefficient, whereas the residual plot indicated that PAR has a more significant effect on changes. No reason could be found as to why the conflict was occurring in this one case.

Dataset 2 showed a different result. No relationship appeared to exist between the design attributes and changes. Therefore, the author concluded that these datasets showed no evidence of a consistent relationship between design measures and number of changes.

When the author examined the relationship between code-based attributes and change-proneness, dataset 1 displayed no evidence that any of the attributes affected changes and dataset 2 indicated that only McCabe's complexity V(G) was marginally significant.

The conclusions from the division of the data seem to be:

(1) the code attributes appear to have been masking any effect that the design attributes might have had;

(2) no consistent general relationships were found.

The author then decided to apply principal components to investigate how the attributes within the design-based and code-based groups affected each other.

The principal component analysis of the design-based attributes showed the first principal component to be an equal weighting of the number of modules called and the number of data items. The second component was composed of the number of parameters.

The result explained why both fan-out and the data items attribute were never significant in the same equation. Mathematically they could be used as one attribute although this is not consistent with their meaning and is likely to be a feature of these datasets. Both datasets showed the same results.

The principal component analysis on the code-based attributes also showed consistency between the two datasets. The first component indicated that all the code-based attributes measures of size and the same amount of information could be obtained by using a single measure which was a weighted average of all the values.

3.4 Identification of Atypical Values

The monitoring mode of COQUAMO required statistical techniques to identify atypical values. It is possible to use attribute values as anomaly detectors (that is, indicators of unusual or atypical events or objects) to allow the software development process to be monitored [21,16]. Anomaly detection can be univariate, bi-variate or multivariate. Different statistical techniques are required for each type.

3.4.1 Comparison of Single Attribute Values

Univariate detection requires identification of an attribute value as outside the 'normal' range compared with the values obtained from other similar products. This can be achieved by using a measure of central location of a data set plus a measure of the expected variability about the central location. For example, it may be useful to determine whether the testing effort used to produce a product of given size and application type is unusually large or small compared with other similar products. A measure of central location of a dataset of testing values from various similar products

plus a measure of the expected variability about this central location is required to establish whether the testing effort for a new product is in any sense abnormal.

The author's first step was to calculate a measure for the centre of location. The usual measure of the centre of location is the arithmetic mean. This is the numerical average of the values for each attribute. One alternative to the mean is the median, which is the mid-point of the attribute values for the dataset. To assess which statistic was better, the author compared the mean and the median for datasets 1 and 2.

Tables 3.6 and 3.7 show the comparison between the mean and the median as measures of the centre of location for Data sets 1 and 2 respectively. For both of these data sets, the mean value is usually far larger than the median. This is because of the presence of extreme values.

	Mean	Median
MCI	335	161
LOC	162	90
FO	4	4
DI	7	4
PAR	3	2
n1	31	29
n2	50	41
N1	226	99
N2	200	95
V(G)	21	14
CHG	5	4

Table 3.6 SS1 - Mean and Median

	Mean	Median
MCI	263	177
LOC	189	126
FO	5	4
DI	3	2
PAR	4	4
n1	32	32
n2	49	38
N1	180	128
N2	175	125
V(G)	20	14
CHG	3	2

Table 3.7 SS2 - Mean and Median

Figure 3.14 shows the mean and the median for machine code instructions. It is clear that the median is a more plausible measure of the centre of location of the datasets since it is not distorted by atypical values or anomalies. The median, therefore, appears to be a more stable choice for a 'centre of location' statistic.

The boxplot is a useful technique for indicating the distribution of values in a dataset which also explains why the mean and the median are not the same. A boxplot provides graphical representation of the following features of the data:

- centre of location (median);
- spread;
- range of data;
- outlying anomalous data points;
- skewness.

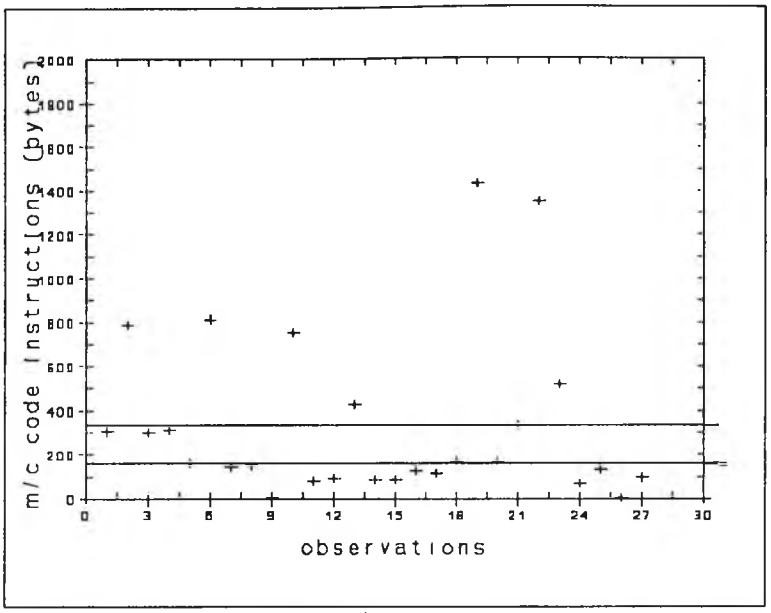


Figure 3.14 Comparison between Mean and Median

This compact data display is also very useful for comparing several groups of data. Figure 3.15 provides an example of the general result the author found when she applied this technique. The distribution for each metric is shown as a boxplot in Appendix A.

The median value is the value which divides the dataset in half and this is represented by the line or crossbar in the box. In the example (Figure 3.15) the median values are 161 and 177 bytes for subsystem 1 and 2 respectively. The position of this line within the box indicates the skewness of the data. The data from both subsystems is heavily positively skewed since the median lies to the left of the box. If the data was symmetric then the median would be in the centre of the box.

The average spread of the data is shown by the position of the box. The edges of the box show the upper and the lower fourths of the data, therefore the box represents the middle 50% of the data or the fourth spread. In the example the fourth spread is from 93 to 380.5 bytes for subsystem 1 and between 92 and 429.5 bytes for subsystem 2.

The 'normal' range of the data is shown by the lines which extend from the edge of the box to the most remote data values in the dataset which are not outliers. These lines are called the upper and lower tails of the data distribution. The upper and lower tail values are defined as $F_U + 3/2d_F$ and $F_L - 3/2d_F$ where F_U and F_L are the upper and lower

fourths and d_F is the difference between the fourths, i.e. the fourth spread. The 'normal' range for machine code instructions in the example is between 0 and 812 bytes for subsystem 1 and between 0 and 936 bytes for subsystem 2.

The outliers are defined as those values which are greater than the upper tail value and less than the lower tail value. The subsystem 1 outliers are indicated by the crosses at 1350 and 1425 bytes and the subsystem 2 outlier at 994 bytes.

As can be seen from the comparison of the two boxplots of the machine-code instructions, the distribution of the data values is similar between the two data sets. Since boxplots for the other metrics were also similar there appears to exist some consistency in the distribution of software data values in this environment.

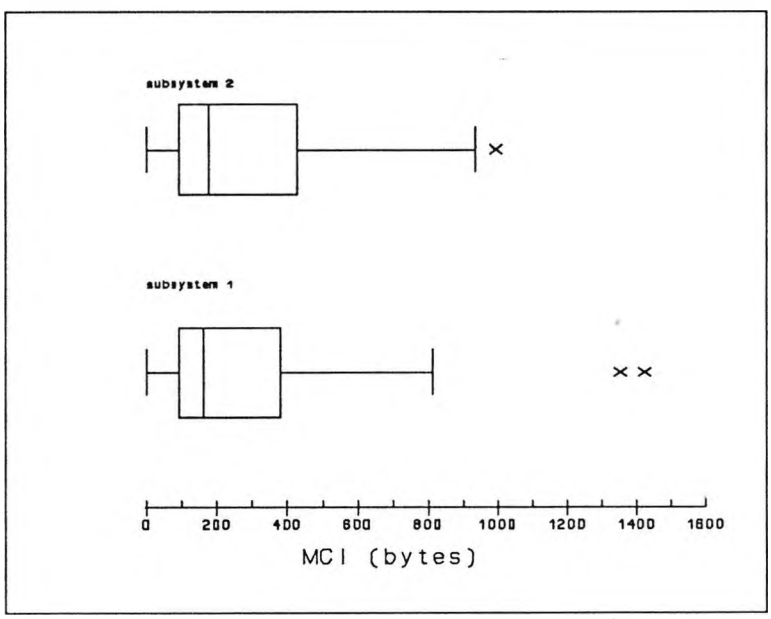


Figure 3.15 Boxplots for Machine Code Instructions

Looking at all the box plots provides some idea of the nature of the data and the dangers in the use of the mean rather than the median. If the underlying distribution of the data is symmetrical about its mean then the mean and the median coincide. However, if the distribution is highly skewed, as it is with the data under study, then the median is a more intuitively appealing measure of central location than the mean. In addition, there is a danger in applying the mean when extreme outlying points are present in the dataset since they will distort the value of the sample mean and give misleading results if the sample mean is used to construct 'normal' ranges. The further the outliers are from the

rest of the data values the more misleading the mean can be.

Most of the data values, as well as including outliers, were also heavily positively skewed. This is not unexpected since programmers, as a matter of good programming practice, tend to keep the modules as small as possible. This trend can, therefore, be expected to be present in most software data. It is also probable that other software datasets will reveal a number of outliers. This does not mean that every module will be small because the optimum size depends on the underlying problem (i. e. the function being coded). If the problem is large then the module representing the solution will probably be large.

The boxplot appears to be a very useful technique for automating the detection of univariate anomalies since it does not make any assumptions about the nature of the data and is not subjective. The use of the boxplot for automating anomaly detection of the monitoring mode of COQUAMO is described in Chapter 6 "Component-based Anomaly Detection".

3.4.2 Bivariate and Multivariate Detection

Bivariate and multivariate anomaly detection requires a relationship or trend to be established such that those components which do not follow the identified relationship can be easily identified. With bivariate detection a simple scatter plot is may be sufficient. The only problem with a scatterplot is that visual inspection and identification of anomalies is subjective. The use of scatterplots for bivariate anomaly detection is discussed in Chapter 6 "Component-based Anomaly Detection".

Regression can be used to establish a relationship between attribute values but a technique is also required to detect when a component (or components) deviates significantly from the general relationship. Analysis of the residuals obtained from the regression line may be used to identify anomalous components because the size of the residual identifies by how much a particular component deviates from the identified relationship. For example, to detect particularly error-prone modules within a subsystem, the error rate per module for the modules within the subsystem can be

compared with the average (or median) rate. This is achieved by establishing a relationship between the number of errors and module size.

In order to investigate multivariate anomaly detection the author used untransformed data. The consequences of abnormalities in multivariate data are intrinsically more complex than in the univariate case. One reason for this is that a multivariate abnormality can distort not only measures of location and scale, but also measures of correlation. The author investigated the use of two techniques to identify anomalies:

- A plot of the first two principal components;
- A residual analysis from a regression.

The type of anomaly that may be detected by the Principal Component plot is one which is inappropriately inflating the variances and correlations upon which the principal component analysis is based.

The principal component plots (Figures 3.16 and 3.17) appear to identify modules that are abnormal only when a number of different attribute values are considered together, as well as the modules which have extreme values for all attributes. An analysis technique which considers more than one metric is likely to be more useful to the project manager than one which only identifies very large or very small modules.

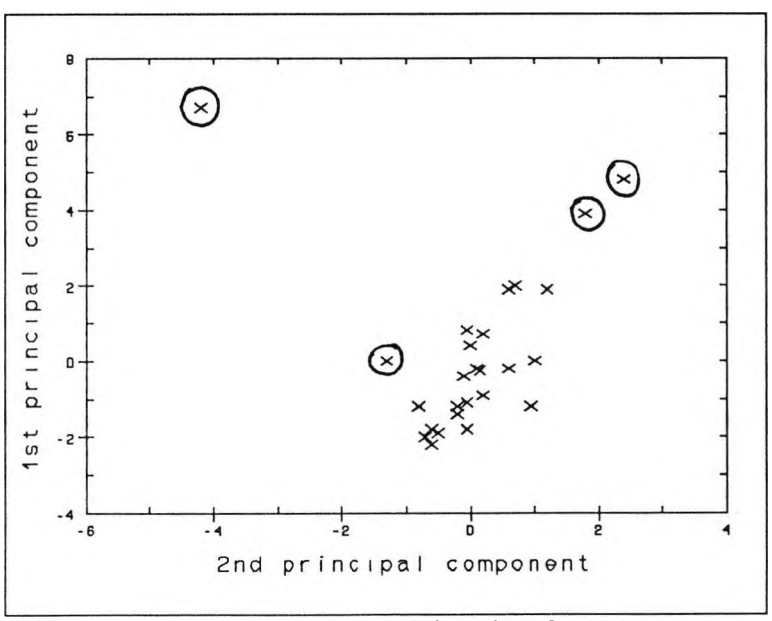


Figure 3.16 SS1 - Principal Component Plot

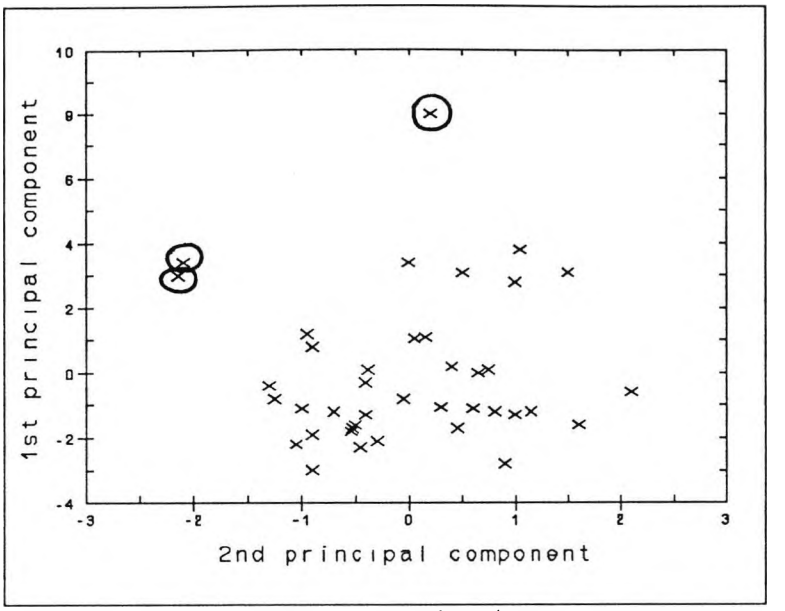


Figure 3.17 SS2 - Principal Component Plot

The second method the author used was to identify the anomalies by visual inspection of the residual plot. The Ordinary Least Squares regression for data set 1 (Figure 3.6) highlighted the modules which were relatively change-prone due to their large size. It is useful to have a technique which identifies these modules but it is likely that a project manager would already be aware of the potential problems concerning large modules which exhibited large values for all attributes. More usefully, however, the technique identified one module that only appeared abnormal when all attribute values were considered together. This module had relatively high values of fan-out, data items and number of operators and operands for its size. The residual plot for the robust regression (Figure 3.8) identified all the modules that the OLS plot identified as potential anomalies except for the one mentioned above. The residual plots did not highlight any modules which had a relatively low number of changes with respect to all the other attribute values.

The residual plots for data set 2 from the OLS regression (Figure 3.7) and the robust regression (Figure 3.11) were not consistent. The robust residual plot shows five potential anomalies which can be split into two categories:

- large modules (i.e. modules for which all the metrics had large values;

- unusually change-prone modules (i.e. modules which were more change-prone than would have been expected for their size).

The OLS residual plot did not highlight any anomalous modules.

Therefore, of the two techniques used, the principal components appeared to highlight the most interesting and useful potential anomalies. It is unclear whether this is because the regression technique did not detect the best relationships or whether the residual analysis technique is not useful for detecting multivariate anomalies.

3.5 An Evaluation of Some Design-Based Metrics

COQUAMO-2 assumed that prediction of product quality would be made throughout the product lifecycle. It was therefore important to investigate attributes that would be measurable during the early stages of development. Therefore the REQUEST project undertook an investigation of some design metrics.

The aim of this investigation was to evaluate the Henry and Kafura's 'Information Flow' design metrics, [44], in comparison to the simpler code-based metrics of size (lines of code) and control flow (number of branches). Henry and Kafura define a local flow of information from module A to module B as occurring if one of three following conditions hold:

- A calls B
- B calls A and A returns a value to B, which B subsequently utilises
- C calls both A and B passing a value from A to B.

The comparison involved investigating the ability of the information flow metrics to identify change-prone, error-prone and complex components in comparison with the use of code-based metrics. This work is also documented in Software Engineering Journal [45].

The information flow metrics measure the links among components in terms of the flow

of information among components and are relevant to any system which has been developed using a structured design technique or can be represented using a structure chart [46,47]. The data used in this study was dataset 3 (see section 1.3). The procedure code size metrics used were lines of code (LEN) and control flow (CF). The measures used to assess the final characteristics of a procedure were number of known errors (KE), number of planned changes (CHNG) and subjective complexity (SC). The design metrics were informational fan-out (IFO), informational fan-in (IFI) and informational flow complexity (IFC).

The definitions of the design metrics were used in this study are:

- IFI = number of procedures which call the procedure
  + number of data structures from which the procedure receives data

- IFO = number of procedures called by the procedure
  + number of output parameters on the procedure's interface
  + number of data structures into which the procedure places data

- IFC = $(IFI * IFO)^2$

These are not exactly the same as the definitions used in [44] since these definitions of the information flow metrics caused some problems. Henry and Kafura themselves have used different definitions in other papers, [48] and [49]. Also, the description of the conditions under which a flow of information occurs is not sufficient to provide unambiguous counting rules for information flow. For example, the counting rules are not fully defined for the third type of local information flow. The counting rules associated with the third type of information flow are either:

(1) for A, add 1 to fan-out count
for C, add 1 to fan-in count
add 1 to fan-out count
for B, add 1 to fan-in count

or

(2) for A, add 1 to fan-out count
for C, leave fan-in and fan-out counts unchanged
for B, add 1 to fan-in count

The second counting rule seems more logical but would then appear to contradict the counting rule implicit in the definition of the second type of local information flow.

Another difficulty with the counting rules occurs if module C processes a value returned from A, prior to its input to B. The returned value from A should then be regarded as a fan-in to C but it is not clear whether it should be counted as one fan-in to B (from C) or two fan-ins to B (one from C and one from A). Therefore, the counting rules for the third type of local information flow were not totally clear and it would have been difficult to count such indirect flows manually.

The multiplication of IFC and program size, to obtain a procedural complexity metric, was ignored, because for both evaluation and interpretation purposes, Henry and Kafura consider the information flow part of their procedural complexity metric separately from the code size part. In addition, if information flow metrics are to be used to evaluate a design prior to coding, they should not include a measure of code size which would be unavailable.

Henry and Kafura found their metrics were able to identify change-prone metrics in a UNIX 3 environment. If these metrics were of general use this could be of value to software engineers and managers. The evaluation procedure used was to investigate the relationship between the design metrics and:

- the number of changes to components that resulted from system enhancements;
- the number of changes to components that resulted from component faults;
- the subjective assessment of component complexity, provided by team leader of system developers.

The work was done in conjunction with Dr. Barbara Kitchenham, who did the actual evaluation while the author did the statistical analysis. We were looking for attributes which would identify the largest proportion of change-prone, error-prone and/or complex components while maintaining a relatively low false identification rate.

The attributes were divided into 'quality indicator metrics' (design and code based attributes) and 'quality characteristic metrics' (changes, errors and subjective complexity). The boxplots showed, as expected, that the data was heavily skewed. The boxplots are shown in Appendix C. One point worth noting is that the information flow complexity metric has a large number of anomalies compared to the number found for its constituent parts. This may cause too many benign anomalies to be identified as a potential problem if the metric was used for quality control purposes. There is also no reason why its constituent parts cannot be used instead of the compound metric since they have to be investigated before the anomaly can be interpreted.

3.5.1 Relationships Between Measures

An initial assessment of the relationships between the 'indicator' metrics and the 'characteristic' metrics was obtained by a correlation analysis. However, due to the nature of the software data, described previously, the Pearson correlations shown in Table 3.8 must be treated with caution. The additional problem of a large number of "ties" (i.e. components with the same value) for the characteristic metrics meant that the Spearman's rank correlation coefficient could not be used instead. This is because the components with the tied value are randomly allocated a position in the ranking which may result in a misleading coefficient. Therefore, the non-parametric Contingency C coefficient was calculated to compare with the parametric coefficients and are shown in parenthesis in Table 3.8. The contingency tables were constructed by splitting the metric values into four groups (relating to the boxplot) containing the values:

<= lower fourth

> lower fourth and <= upper fourth

> upper fourth and <= upper tail

> upper tail

Quality indicator metrics	Quality characteristic metrics		
	KE	CHNG	SC
IFC	0.07 (0.31**)	0.06 (0.35**)	0.12 (0.37**)
IFI	0.03 (0.25)	0.08 (0.18)	0.09 (0.25)
IFO	0.53** (0.43**)	0.45** (0.42**)	0.48** (0.49**)
LEN	0.65** (0.53**)	0.44** (0.35**)	0.58** (0.58**)
CF	0.65** (0.46**)	0.43** (0.37**)	0.57** (0.59**)

Table 3.8 Correlation Coefficients

The Pearson correlation coefficients showed that the informational flow complexity metric based on Henry and Kafura's approach was not significantly correlated to any of the quality characteristic metrics. However, if the components of the informational flow complexity metric were considered separately, it appeared that informational fan-out was significantly correlated to all three quality characteristic metrics whereas informational fan-in was not. The contingency correlations, however, indicated that the informational flow complexity metric was significantly associated with the quality characteristic metrics although the association is less than that observed for the other indicator metrics. Information flow fan-out still significant and informational fan-in remained non-significant.

The results contradict Henry and Kafura's results but are consistent with a study by Troy and Zweben, [50]. They used fan-out and fan-in metrics based solely on procedure calls. In their study, they observed that fan-out was related to errors and fan-in was not. The code metrics exhibited larger correlations with known errors and subjective complexity than the informational fan-out metric. This study is less encouraging than the Kafura and Canning [39] study, which indicated that informational flow was at least as good a predictor of error-prone procedures as size, and a better predictor than McCabe's cyclomatic complexity metric [16]. There were problems with the extraction of the metrics and the identification of the underlying causes of the metric values.

The extraction problems were due to a combination of two problems - ambiguous published data definitions and difficulty in manually collecting some of the primitive counts. Metrics cannot be properly validated without good definitions and it is unlikely that the metrics can be used in practice in a software production environment without data collection and analysis tools.

The information flow metrics are obtained by combining the values of different counts. The main problem with the combined metric is that it can have different underlying causes, for example, the program with the largest informational fan-in value was a frequently used function which was called by many different paths and therefore was a critical program. However, the program with the second largest fan-in values had a large value because it read from a large number of data structures but is only called by one other program. The different underlying causes may lead to an incorrect diagnosis. Therefore the study suggested that a simple measure of fan-out was a more useful metric than the more complex information flow metrics.

The study did confirm that design metrics could be used as early predictors of problem modules. The results of the study suggested that it might be cost-effective to give special attention to programs with high informational fan-out values. Extra time spent on 18% of the programs would have been 82% effective (82% of the programs with high fan-out values also had high quality characteristic values) and 45% effective (45% of the programs which required extra development time would have been identified).

3.6 Conclusions of Preliminary Analysis

It is clear that the underlying nature of software data causes a number of problems when using classical statistical techniques. These techniques generally assume a Gaussian distribution which is symmetrical, has constant variance and a low expected number of outliers. In contrast, software data often appears to be skewed, has an increasing variance and a relatively high number of outliers.

This does not, however, imply that classical techniques are completely inappropriate for software data but only that care must be taken to check that violation of their

assumptions do not lead to misleading results. If the effect of the violation is that results are misleading, or infeasible, then either the data must be transformed so that the assumptions are met, or robust techniques or non-parametric techniques are required.

In the author's view statistics does have a place in software engineering, but must be used with care. It is essential to have a clear idea of why statistical techniques are required and a 'hypothesis' about expected software engineering principles being investigated. If no expectations exist, statistics cannot be used effectively.

The results from this chapter have implications for the derivation of the COQUAMO-2 model.

- No common relationships were detected between the datasets when module relationships were considered. Therefore, there is little hope of identifying a 'general model' between module attribute and product quality attributes.

- Some techniques were good at identifying outliers/anomalous components irrespective of the particular attributes. These techniques were boxplots for the univariate outliers, scatterplots for the bivariate outliers and principal components for the multivariate outliers.

Therefore, the goal of COQUAMO-2 was changed to monitoring the development process using quantitative information collected during the development process and detecting when there was a potential problem. This involves identifying atypical values with respect to planned values and comparison between individual component values. These aims are described in chapters 5 and 6. The interpretation of detected atypical values is described in chapter 7. The work in this chapter (excluding section 3.5) is also reported in [51] and [52].

4. Anomaly Detection Survey

This chapter describes a survey which the author organised to identify whether the concept of controlling anomalous components was easily understood by expert project managers and quality assurance managers. This was done by investigating the type of atypical components an expert project or quality assurance manager believed to be anomalous.

The aim of the thesis is to identify statistical techniques which are appropriate for the analysis of software development metrics and to investigate how they might be used to support quality management procedures. The results from the survey were intended to be used as "best practice" against which various automated statistical techniques could be compared. The aim was to investigate which statistical technique would best emulate the expert project manager in detecting anomalous components. This investigation is described in chapter 6 "Component-based Anomaly Detection" along with the statistical techniques under review.

Initial analysis of software data (see Chapter 3) indicated that datasets of measures of software items (modules/projects) usually included items that exhibited unusual values or unusual combination of values. Such anomalous items often exhibited quality problems (e. g fault-proneness). The REQUEST project concluded that statistical techniques that assisted the identification of such items would assist project and quality managers to monitor and control their software projects. The REQUEST project assumed that project and quality managers would already be using this sort of approach informally. Therefore, the author devised an survey to review the way in which project and quality managers currently assessed unusual components.

The work in this chapter is part of the monitoring mode of the REQUEST model (called COQUAMO-2). It was intended that the subjective information gained from the project managers would be used to compare the effectiveness of different statistical techniques supporting automatic anomaly detection.

The survey had two aims, with the first being the major one:

(1) to determine what a project manager would identify as an unusual component from a scatter (or a density) plot;

(2) to determine whether initial standardisation of the attribute values affects the detection of unusual components.

The reason for the use density plots,as well as scatter plots, was due to the nature of software data. Software data often has tied values which overlap on a scatter plot. When this occurs a scatter plot often gives a misleading impression of the density of the points. An example of this can be seen in Figure 4.1, where 226 data points are shown as only 24 on a scatter plot (dataset 3).

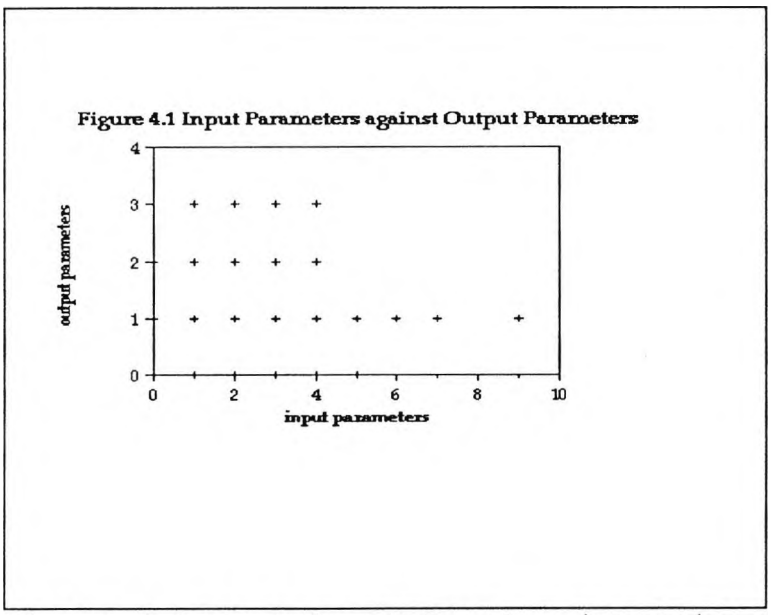


Figure 4.1 Example of Overlapping Points

The author observed that approximately a quarter of the scatter plots in this dataset had this problem. The author investigated several ideas to see the best way to represent this data, given the capabilities of the available data presentation tools. The method selected was a density plot. The density plot shows the number of points in each cell. Within each cell, the points are randomly scattered although they all have the same value and would overlap if plotted on a scatter plot. The reason for scattering the points is to provide some visual impression of the density within each cell. The density plot can be viewed in a similar way to the scatter plot allowing potentially anomalous points to

be highlighted.

4.1 Design of Survey

The author chose eleven graphs for the survey. The attributes included in the graphs were:

- size;
- control flow;
- module enhancements;
- errors;
- parameters;
- data items;
- fan-in (number of calling modules);
- fan-out (number of called modules);
- subjective assessment of complexity.

The author chose the attributes and graphs which were most useful in helping to detect where problems were in the development of the project. This was assessed by a manual analysis of the data (by Dr. Kitchenham), [53]. The actual graphs are included in Appendix D. The managers approached to take part in the survey were chosen because they were experienced managers and they were expected to have experience of using measurements to identify software development problems.

Each project manager was given a brief description of the survey, explaining why it was being run, and a short questionnaire to allow the possibility of any discrepancies in the managers' identification of anomalies to be explained by the most likely causes. The managers were then asked to mark any modules which they regard as unusual on a range of scatter plots (or density plot, when a scatter plot is not appropriate), indicating whether the marked module was likely to be unusually favourable or unfavourable (or no mark at all if it is unclear, or could not be determined by the information provided). The questionnaire given to managers is given in Appendix D.

Since there were two distinct aims of this experiment, two distinct methods were required to deal with them. The first, and most important aim, involved identifying a common set of anomalies for each plot to summarise the information from the project managers. The resulting plots were then intended to be used as a control for the automated detection techniques. This meant that each technique's results were to be compared with those derived from the manager and checked for similarity. The anomaly detection technique which identified the majority of the unfavourable anomalies identified by the project managers would be chosen for inclusion in the COQUAMO prototype.

The second aim intended to be dealt with using the 'density grid' method, which does not depend on the use standardised values. This technique could be compared to the control, using both standardised and raw values. If the results of this comparison showed that there was a difference between the anomalies detected when standardised metrics are used, then any technique which requires standardised metrics would have to be rejected.

4.2 Survey Results

This section gives the results of the survey and the problems encountered in the analysis of the questionnaires.

Two batches of questionnaires were handed out. The first batch of 120 questionnaires was handed out at the QA Forum. Ten replies were received, three of which were from the same company. The second batch of questionnaires was given out to various managers in a software company. Seven replies were received.

Due to the small number of replies and the wide background variety of the respondents, none of the results could be generalised. Also, the differences between the managers replies could not be explained.

4.2.1 Background of Responders

The majority of the respondents, as expected from the QA Forum, were involved with Quality Assurance although a few used to be project managers. They included a combination of quality assurance and consultants. A large majority had over ten years experience in the software field and a few had a software/hardware background. None of the respondents had a purely hardware background. The software company responders, which were a combination of project managers and consultants, all had more than ten years experience and, like the other responders, the majority had software experience.

With the QA Forum responders, although 6 out of 8 (two did not provide any information) used metrics during development, only 3 of the responders used them to detect problems. The majority were not familiar with the use of scatterplots to detect problems, which might be one of the reasons why the managers had problems viewing the plots. With the software company responders the situation was slightly different. Six out of seven responders used metrics to assess project progress and to detect problems. However, the results from both groups were equally varied.

The QA responders had a wide variety of type of company, implying that they deal with a wide variety of software products. Although the software product was the same for the software company responders, it is a large product with a variety of different functions.

4.2.2 Results of Analysis

The type of response varied from plots being completely unmarked, to the provision of added information about what would be investigated to identify whether the marked module was truly anomalous or not. There did not appear to be any significant difference between the two groups of replies. There was, however, a difference between the consultants' replies and the practitioners. The consultants all had some idea of what they thought was the problem and what they would look for next (although they did not all agree with each other).

There was a wide variance in anomalies detected by the managers. This caused a slightly different approach to be taken when gathering the information from the plots. It was obvious that different managers would not all mark the same modules as anomalous but it was envisaged that there would be a common sub-set. However, it was not possible to identify any common sub-set since there was little consensus of opinion between the managers as to which modules were anomalous (in fact, no one module was identified as anomalous by all managers). It was therefore necessary to lower the 100% agreement for identification of an anomalous module before the module was included in the set of agreed anomalies. However, even reducing the agreement level to 50% did not allow the detection of a common sub-set of anomalies.

One reason for the lack of consensus might be that the managers had a poor understanding of the use of software attributes during development and were only interested in the attributes which they already use, namely errors and changes which are commonly used at the end of development. One response which supports this view was from a manager who stated that the number of data items used was not important to him. Another said he was only interested in the number of errors and he did not care, what any of the other attribute values were. In one respect this might be good news for an automatic system because it would hopefully guide the manager into looking in more detail for potential problems. However, developing an automatic anomaly detection system consistent with a manager's approach appears impossible because there did not appear to be any consistent approach.

For a small sub-set of the plots (those which contained fan-in and fan-out) there was no response from the majority of the managers. Since fan-in and fan-out may not be as familiar to managers as other metrics, it was suspected that one reason for the lack of response might have been lack of understanding of what the implications were for an unusual module. Managers did not appear to be familiar with the use of software attributes. They do not understand how to interpret them or use them.

In some cases, it appeared that the managers were looking for a particular type of pattern and identifying modules which do not conform to the expected pattern as anomalous. In many cases the managers did not know which patterns are correct for the different attribute combinations.

Some of the responses were screened out after observing the type of modules marked as anomalous on some of the plots. Examples of this are:

- In a couple of cases the cell of the grid marked, on a density plot with the highest density was marked. In one case a manager marked the highest density cell, which represented the modules with average size and average complexity, as being an unfavourable anomaly;

- There was a tendency to view the plot as a uni-dimensional plot, especially with respect to errors.

The background information supplied by the managers did not explain the reason why the variety of replies were found. The replies within the single software company were as varied as those from many different companies, although the background information was similar for the single company.

As well as identifying potentially anomalous modules, the project managers were also asked, if possible, to indicate whether they thought the anomaly was favourable or unfavourable. The results of this were not conclusive. The only point to emerge from this, and because of the small sample size cannot be taken as general, was, when a small set of modules were looked at, that a higher proportion of the anomalous modules identified were marked as unfavourable. This highlights that managers are only interested in detection of modules which are regarded as potential problems. However, a statistical technique which automatically detects anomalous modules cannot differentiate between unfavourable and favourable anomalies. This highlights the need for an advice system to be linked to the anomaly detection, to provide some help on interpretation of the detected anomalies. Anomaly detection on its own is of limited use to the project manager.

The need for a linked advice system was also confirmed by the result that often managers did not indicate whether they thought combination of attribute values was favourable or unfavourable. This was not surprising because each plot was being regarded independently. This condition is required for the automatic detection of anomalies but the decision as to whether an anomaly is favourable or not often requires

more information than can be obtained from a single bivariate plot. However, some managers identified certain anomalies as having favourable implications and other managers identified the same anomalies as unfavourable for the same plot. This again highlighted the managers lack of understanding of the use of software attributes during the development of a product.

Another situation which was occurring, was the tendency for the managers to assume that zero or a low number of errors, are an indication of high quality, regardless of any other attribute values. They did not question whether the module had been adequately tested. An extreme example was where one responder decided that all components which had zero errors during development were good and all those which had non-zero errors were bad. The COQUAMO automatic system should not emulate this but it was a good advertisement for the need for the COQUAMO model which discourages considering only one attribute value in isolation.

The overall conclusion from the experiment was that the managers' replies were not consistent enough to be used as a control in the experiment for the choice of the most appropriate technique for automatic anomaly-based detection. The managers may have confused interpretation with detection and therefore made it difficult to isolate their detection strategies.

It appeared from the results that managers are not familiar with the use of software attributes. They do not understand how to interpret and use them. This contradicted the author's original assumptions when devising the survey. This highlights two issues:

(1) For such a survey to be effective it would need to be re-done after proper training;

(2) New attributes need to be related to project management issues if they are going to be used successfully.

Although they could not be used for the original purpose, the results were useful because they showed the potential benefit an automatic anomaly detection system might be to project managers if some help on interpretation of the detected anomalies was

provided.

Since it was impossible to use the results of the survey, identification of the most appropriate statistical technique was based on a consultant's view of what constituted an anomaly. This was justified because the survey did indicate that consultants are using measurements to detect anomalies during the development process. The investigation of statistical anomaly detection and results are described in chapter 6 "Component-based Anomaly Detection".

5. Project-based Monitoring

This chapter is concerned with monitoring the overall project using quantitative information collected during the development of the product at specific defined milestones. This is called project-based monitoring. The aim of the thesis is to identify statistical techniques which are appropriate for the analysis of software development metrics and to investigate how they might be used to support quality management procedures. This chapter shows how summary quantitative information can be used to help the manager monitor his/her overall project and can therefore control the development process.

The work identified in this chapter forms part of the monitoring mode of the REQUEST model (called COQUAMO-2). The REQUEST project team aimed to build a prototype tool to support COQUAMO. As part of the prototype, the author designed the summary reports which the COQUAMO-2 prototype would display. She had no involvement with the implementation of the prototype.

The author's task was two-fold:

(1) To identify a sub-set of essential information required to allow a project manager to control his/her project. The sub-set was taken from a list of potentially useful attribute measurements [54];

(2) To set targets, where appropriate, for each attribute measure identified in (1) above. Monitoring is based on the principle that when an actual attribute value exceeds the target value then it is likely to be a potential problem in the development. Thus, monitoring relies on identifying target values of attributes. Targets can be derived from past history but if no past history results are available default targets are needed.

5.1 Introduction to Project-based Monitoring

The aim of the project-based monitoring is to enable a project manager to identify

whether a project is deviating from what was originally planned or expected. This type of monitoring is concerned with the behaviour of the whole project, at well-defined points in time, not the behaviour of particular product components.

In order to assess whether a project is progressing appropriately, a project manager must have a formal plan or informal expectation. For example, if a project is using too much effort, an expected or planned value for effort must exist to compare the actual values with.

The reason for a deviation from expected can be due to one of three causes:

- the planned values (i.e. targets) were unrealistic and inaccurate;
- an unexpected problem has occurred during the project;
- a deliberate or known change has been made to the project but the planned values have been unaltered. For example, if a problem was found in high level design it might have required extra effort to correct it in detailed design. Therefore, the actual values are showing the impact of the solution not the problem.

Deviations can be due to serious problems and can, therefore, require solutions that can seriously affect all the plans. An automatic measurement system cannot identify the reason for a deviation from target, only the project manager can do this but it should allow for re-planning.

Project managers are trying to control their projects. They want to be able to tell whether their project is going well or whether there are problems. Since the earlier a problem can be detected, the easier and cheaper it is to solve, project managers need regular feedback to maintain a particular standard of product. Early feedback minimises the time it takes to respond to a potential problem.

Support for timely feedback means collecting information at regular intervals throughout the product development. The REQUEST approach is to use the end of each lifecycle

phase as a checkpoint for analysis and reporting on data.

Measurements can help project managers detect potential problems with their projects. As well as providing a quantitative measure of project progress, they can assist diagnosis of the particular type of problem that has occurred.

For project planning and control to be effective, a project manager must set targets for quantitative attributes that are capable of providing appropriate information about the project environment or development process. For example, there is no benefit in setting a target on the percentage of re-use in a project, if re-use is not an important issue. However, whatever specific goals a project or company wishes to achieve, if project-based monitoring is being used, a base set of goals are assumed to be desired. Three main goals can be assumed to be general for any development:

(1) Keep the development on time;
(2) Keep the development within budget;
(3) Maintain the quality of the product throughout development.

The user must define what he/she means by quality so that the REQUEST model can be of most benefit to him/her. It should be noted that, for the purposes of this work, quality has been taken to mean a fault-free and stable product, and monitoring of quality in the REQUEST prototype was taken to be similar to that of statistical quality control. To support the three goals identified above, different types of attribute measurements will be required:

(1) time;
(2) resources;
(3) quality indicators.

At present the third goal is too imprecise to be quantified. The goal needs to sub-divided so that the appropriate measurements can be chosen to show whether a particular sub-goal has been achieved. The following questions can be asked, sub-dividing the quality goal 3 into issues 3, 4 and 5:

(3) Is the software faulty?

(4) Is the software unstable?

(5) Are all the tasks such as testing being performed properly?

It is an assumption of the system that the product being developed supports the requirements, that is, all the required functions are provided. It is possible for the system to assess this issue by the use of traceability metrics.

The targets can be sub-divided into those which are external to the project and those which are internal.

(1) External Targets

The external targets are usually those set on timescale and budget. These targets may not be under the control of the project manager and cannot be changed without a contract re-negotiation. Monitoring against these targets will highlight whether the external constraints on the project are feasible or not.

The need to monitor budget and schedule is readily accepted by project managers. Budget and schedule targets are usually set by higher management, when a project is approved, and are usually quantified. They are monitored to ensure that the project will complete within the required time and cost, and to detect any problems which might mean that the targets would not be met. The project manager is likely to be pressurised into meeting budget and schedule targets and the success of a project is often judged on these goals. This is because when a budget or schedule target is missed it is easily detected and the effect is easily understood and translated into monetary terms. However, budget and schedule should not be the only consideration. The original budget and schedule targets may have been unrealistic or an unforeseen problem may occur. In these situations, there is a temptation for a project manager to reduce the product development timescale in later stages of the project, or reduce reviews and testing of the intermediate products (or even the final product). Although the effect of budget or schedule compression may not be immediately obvious, it is likely to result in a poor quality product which will be costly to maintain, and will reduce customer satisfaction. Therefore, quality indicators should also

be monitored to ensure that meeting short term productivity goals is not achieved at the expense of product quality, resulting in higher long terms costs for the company.

(2) Internal Targets

Internal targets are planning values set by the project manager. They include defect levels, change levels and the distribution of effort and duration across the phases.

One of the major problems a project manager has when using measurements, is to decide which measures are most important to their own particular project. Some general guidelines can be provided to a manager to help with this task. Information is required that will help a manger to detect quickly if any major problems are occurring with the project and, if so, to identify the particular underlying problems and how they can be resolved.

As part of the REQUEST COQUAMO prototype, a set of screens have been defined which identify, in general, what information might be required to identify problems in a project and to show how the project, as a whole, is progressing. The original set was very detailed, identifying information which would be useful for a complete analysis of the project.

The author felt that the amount of information asked from the project manager would be overwhelming with some of the benefits only visible in the long term. This would discourage many managers from using any automatic system, especially those who are just starting to collect measures. Therefore, the author identified a sub-set of essential information which would help a project manager control his/her project by identifying only major problems. This will hopefully avoid managers wasting time solving minor problems at the expense of the major problems. Also, if a project manager is asked to provide too much information, he/she will not feel that the use of the COQUAMO model (and the prototype of an automatic tool which was developed by REQUEST) has sufficient cost benefit.

From the questions related to the three goals, appropriate attribute measures can be

collected to adequately monitor the development process:

(1)	within schedule?	monitor	timescales;
(2)	within budget?	monitor	effort;
(3)	software faulty?	monitor	number of faults;
(4)	software unstable?	monitor	number of changes;
(5)	testing done properly	monitor	test coverage.

This basic set of five measures can be sub-divided into more specialised measures but they are all extensions to the basic set taken to a greater level of detail and specialisation for a particular environment or project. For example:

- Total project effort can be split into effort expended to construct the software and effort expended to review that software once it has been constructed;

- Changes can be classified by cause;

- Fault classification can be sub-divided extensively because faults have a fault introduction, detection and removal process, all of which can provide specific information about the product and its development processes. For example fault classification can identify where the majority of the faults were introduced, how various types of faults were detected, and how long faults remain in the product. All this can be used to assess process effectiveness.

5.2 Setting Targets

Target values need to be set at the start of the project but should be amended as necessary throughout product development. Target-based monitoring can be either continuous or at defined checkpoints. In the REQUEST terminology, the term "Continuous monitoring" involves collecting actual values throughout a project phase at regular points in time. Time-series curves are then drawn and checked against expected curves. For example, the cumulative number of faults found during testing per week should flatten out after a point in tine (Figure 5.1). If this pattern is not seen then

either the testing period has not been long enough or the product is particularly fault-prone. For each time series to be checked automatically, all the expected curves would have to be identified in advance. This type of monitoring is not included in the current version of the prototype.

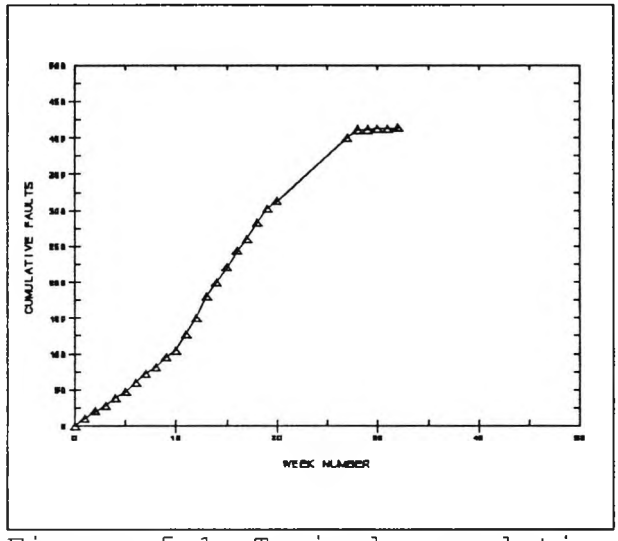


Figure 5.1 Typical cumulative fault v time graph

"Checkpoint monitoring" involves assessing the project status at phase-end or other pre-determined project milestones. Project managers should choose checkpoints which are appropriate to their environment and lifecycle/process model. The advantage of using a lifecycle model is to ensure that the data collection is incorporated into normal working methods with the minimum of disturbance. The reason for monitoring a project by looking at attribute values at the different checkpoints instead of waiting until testing is to provide information early in the development lifecycle about how well the project is progressing. This permits early detection of problems [3]. The earlier the problem is detected the less expensive it is to fix, therefore monitoring is intended to help the project manager to control the project and to reduce the overall cost of the production.

Attributes used for monitoring at each checkpoint are usually environment dependent. This is because different methods can be used to develop a product, for example, object oriented methods, formal methods, structured methods, etc. It is important that the project manager decides what he/she regards as the most useful information to assess at the start of his/her project, since each environment (and sometimes individual

projects) have different conditions and influences. The target values that are planned in the project are also environment dependent and the project manager should not expect targets set in a different organisation to be relevant in their own.

This raises a problem, because project managers may not know what values they expect for a particular attribute target, and may not have any data from previous similar projects from which to assess a reasonable value. Therefore, any automated system must provide default values for some of the more general planning attributes and some guidelines for calibrating the system to the users' own organisation.

The default targets were set on a per phase basis and can be checked against the actual values at the end of each phase. The default targets can be used as an initial starting values but would need to be amended as data about a particular organisation is accumulated. The phases used in the COQUAMO-2 prototype were taken from the standard V model but the beginning and the end of each phase must still be defined by the project manager.

The author identified the default target values using a variety of methods:

- experience (values taken from experienced project managers);
- various literature studies;
- industry values;
- values calculated from some existing, available data sets.

The following list identifies the attributes suggested for collecting to allow a project manager to control the development of a product:

- effort;
- duration;
- re-work;
- faults;
- changes.

The target/planned values were based on a 3GL environment and assumed a well-

structured and stable environment. These target values are likely to be less relevant to other organisations, for example a small software house dealing with be-spoke software.

A single value for a target is not sufficient to detect project problems. The project manager needs to decide what variability he/she will allow in the actual values before a problem is highlighted and action is required. Therefore, for each target, the author has provided some guidelines for an acceptable range around the target value. If an actual value is outside this range then it would be highlighted as anomalous.

The author, jointly with another colleague from the REQUEST project, decided to classify the checkpoint information into different levels of detail. The higher the level number is, the more detailed the information is. The top level or phase summary presents the minimum information required for controlling a project. Table 5.1 gives the target values associated with this information. The author did not assign any target values to effort and duration since the REQUEST project assumed that a cost model (e.g. COCOMO) would provide the resource constraint information. A potential problem or missed target would be flagged if the actual was larger than the estimate.

	Attribute	Target	Anomaly detection range
resource constraints	effort duration re-work	"cost model" "cost model" 10% total effort	 +/- 5% re-work
outstanding problems	unresolved faults faults not cleared outstanding changes	5% faults found 10% faults found 5% faults found	>0.05 * Fault distribution >0.1 * Fault distribution >0.05 * Change distribution

Table 5.1 Phase Summary

Included in the Phase summary was the number of unsatisfied targets and that value as a percentage of the total number of targets. The fault and change distributions show the percentage breakdown expected of the total number of fault and changes found during

reviews, inspections and testing. Table 5.2 shows the fault and change distributions along with the fault origin distribution.

Phase	% faults discovered	% fault origin	% changes identified
requirements	5	15	25
high level design	20	25	20
detailed design	25	35	15
code	10	10	10
unit testing	15	5	10
integration testing	25	10	20

Table 5.2 Fault and Change Phase Distributions

Table 5.3 shows the targets for the level two information. This identifies the breakdown of the resource constraints and event information.

	Attribute	Target	Anomaly detection range
resources	effort	from "cost model"	
	duration	from "cost model"	
	checking effort	prep. + insp. effort	+/- 5%
	other activities	10% effort	actual/total > 0.1
	re-work - faults	10% of re-work	> 0.1 * re-work
events	# changes	0	> #mods,sections
	rate/requ't	0.4	> 0.4

Table 5.3 Level 2 Target Information

The prototype screen for this information also includes number of agreed and number of outstanding changes although no targets have been set on this information.

Table 5.4 shows the level three information. This information reports the information at the inspections/reviews, process assessment and change request classification. Although the information is reported at the project level, it requires information to be available at the component level.

	Attribute	Target	Anomaly Detection Range
Process assessment	prop. planned effort used	from "cost model"	actual > 0.1*estimate
	prop. planned duration used	from "cost model"	actual >0.1* estimate
	fault detection efficiency	50% estimated faults	< 50%
Inspection/review	av. # inspections	3 per component	> 3
	prep. effort	2 * # inspectors hrs	<0.1*target+prep.eff >0.1*target+prep.eff
	av. # inspectors	3<=5	<3, >5
	inspection effort	#total faults/4	<0.1*target+insp eff >0.1*target+insp eff
	inspection rate	4 faults per hr	<3.6 faults per hr >4.4 faults per hr
	clearance rate	2.5 hrs/fault	< 2.5
	#faults found	5*size/100loc 5*size/1000words	#faults*100/size<4.5 #faults*100/size>5.5 #faults*1000/size<4.5 #faults*1000/size>5.5
	major faults	25% of faults found	> 0.25 *faults found
changes	user requirements	25% of changes	> 0.25 * #changes
	internal standards	25% of changes	> 0.25 * #changes
	external standards	25% of changes	> 0.25 * #changes
	others	25% of changes	> 0.25 * #changes
quality assessment	fog index	9-instruction man. 12-technical docs	> 9 > 12

Table 5.4 Level Three Targets

All the figures for effort and number of faults need to be multiplied by the relevant phase percentage taken from the fault and change distribution value for the required

phase. This level of screen also includes a check both on the number of test cases run against those planned and on the test coverage achieved although there are no targets. The quality assessment was assumed to assess subjectively for "completeness", "correctness" and "traceability" attributes.

The information at level four only includes the detailed fault and structural information. The author only set targets on the fault information. Table 5.5 shows the targets for fault classification which is linked to the inspection information.

Fault classification	type of fault	Target	Anomaly Detection Range
Discovery of fault	missing	phase distribution * 0.4	> target +0.1*target
	wrong	phase distribution * 0.45	> target +0.1*target
	extra	phase distribution * 0.1	> target +0.1*target
	query	phase distribution * 0.05	> target +0.1*target
Origin of fault	missing	phase distribution * 0.4	> target +0.1*target
	wrong	phase distribution * 0.45	> target +0.1*target
	extra	phase distribution * 0.1	> target +0.1*target
	query	phase distribution * 0.05	> target +0.1*target

Table 5.5 Level 4 fault classification

The total number of faults was estimated at 5 faults per 100 loc or 1000 words. The reviews and inspection were assumed to be 60% efficient, therefore the residual number of faults is 2 per 100 loc or 1000 words. The author only looked at major faults since she assumed that project managers were more concerned with the major faults.

Reasons for collecting the information and setting a target depend on the intended use of the information. Suggested reasons are given in Chapter 7 "Interpretation".

The default targets are unlikely to be representative of any particular organisation but should help the project manager to set his/her own targets. Targets require calibration to the particular environment in which the project is taking place. They can be calibrated in several ways:

(1) Use of experience - where the project manager knows what to expect if the project is running well from past experience and therefore knows what target to set.

(2) Past project development history data - use of values obtained from previous like projects. If enough project data existed it would be useful to compare actual values from projects which were successful and projects which failed.

(3) Use of models (e. g. COCOMO) to obtain synthetic estimates for a particular measure (e.g. effort, size, etc. for cost prediction).

(4) Use of REQUEST defaults amended with actual values when available - this establishes a database of past projects (see 2 above).

Since default values should only be used in the absence of any other information, this means that calibration of the target values will be the first and most important task of any automated monitoring system. Therefore, it is important to ensure that the calibration facilities are simple enough to encourage project managers to use them. In the current prototype, we have been able to provide tool support in the form of an analysis package (to analyse past project data) but we have not yet included any help facilities.

5.3 The Automatic System

In the prototype of the automatic system, the information is organised into a hierarchy with the most important global information being presented to the manager first. From the global information, the project manager can see at a glance how the project is progressing through the checkpoints or phases, and can identify any major problems. After this, the project manager can ask to see more detailed information on any particular values shown at the global level. The information is arranged to allow the manager to choose the level of detail he/she wishes to see or has collected the relevant information for. The more detailed levels provide more information and therefore help in diagnosing the cause of problems but they require more detailed data to be collected which may not always be possible.

The prototype did not go through a verification and validation process. The current status of the prototype is that it is a working prototype which is linked to the component-based monitoring and interpretation prototypes which are described in chapter 6 and 7 respectively.

6. Component-based Anomaly Detection

This chapter is concerned with monitoring individual components using quantitative information collected during the development of the product. This is called component-based monitoring. The aim of the thesis is to identify statistical techniques which are appropriate for analysis of software development metrics and to investigate how they might be used to support quality management procedures. This chapter shows how attribute measures obtained throughout the development can be used to detect potential problem components. This work forms part of the monitoring mode of the REQUEST model (called COQUAMO-2). The author had no involvement with the implementation of the prototype of COQUAMO-2 after the specification stage.

The author's tasks were:

(1) To investigate different statistical techniques with a view to identifying anomalous components;

(2) To identify statistical techniques which would automatically detect univariate and bivariate anomalous components;

(3) To specify the automatic anomaly detection algorithms.

The aim of component-based anomaly detection is to enable the project manager to detect potential problem components as early as possible in a project. It should be noted that the attribute measures are being used to monitor the process not the final product. This is because the quality indicators available throughout the development do not have a strong enough relationship with the final product to predict the effect that a particular value will have on the final product quality (as shown in Chapter 3).

In order for the monitoring to be effective, managers must identify the goals they wish their projects to meet in order to select the attributes which are most suitable for tracking those goals. For example, a manager may be interested in the quality control of the process but uninterested in the re-use of the product (or parts of it).

As with project-based monitoring, the choice of attributes to measure may be different for different environments since many attributes are dependant on how the software is being developed, e.g. whether the design is predominantly driven by control flow or data flow. However, as a guideline only, the following are examples of the type of attributes which can be used for monitoring and detecting anomalies, on a per component basis:

- size;
- control flow;
- data flow;
- information flow;
- faults;
- enhancements;
- effort - construction and review.

All of these attributes can be viewed as quality indicators for the individual components throughout the development. Size is often used to normalise attribute values for comparison purposes e.g. defects/100 lines of code, control flow paths/100 lines of code. This allows component attribute values to be compared for different components. In addition, it is possible to identify quality trends against size e.g. large modules with low defect rates or small modules with large control flow complexity.

In order to achieve early detection of problem components, measurements need to be collected at each stage of the development process. This serves two purposes in that it provides information that a particular component is a potential problem early in the development process (permitting remedial action to take place), and can track a particular component over-time to indicate whether or not the remedial action was effective. One way of identifying an anomalous component is simply by subjective judgement, for example, looking at a scatterplot of two attributes. There are disadvantage with this method:

1. It is very time consuming, especially when trying to keep track of all the attribute values of a component to provide information about the likely cause of the anomaly.

2. Lack of consistency. No rules are laid down to define what constitutes an anomaly therefore it depends on individual judgement. This also makes it more difficult to justify taking remedial action, since a particular component is not necessarily regarded as a problem by everyone.

Therefore, there appears to be a good case for automating the anomaly detection process. The advantages which can be gained are:

1. an objective, and therefore, consistent way of identifying anomalies;

2. a reduction in time required to monitor project progress;

3. a defined base for identifying what constitutes an anomaly, and therefore a basis for changing system if required;

4. a means of de-skilling the process which is useful for training inexperienced managers and introducing a company standard.

In theory, three different types of anomalies can exist:

1. a particular component with an attribute value significantly higher or lower than the rest of the component values;

2. a particular component has a pair of attribute values which do not exhibit the same pattern as other components;

3. a combination of attribute values for a particular component that is significantly different from the combinations observed on other components.

The first type of anomaly is found when a single attribute is being examined. Each component value is compared with the distribution of the attribute values and components with unusual values can be identified. This depends on a method of defining the distribution of attribute values.

The second type of anomaly is concerned with the relationships between attributes. The general relationship for all the components can be identified and each component compared to this to identify whether it follows the general relationship. This indicates the need for a bivariate technique which deals with relationships among pairs of attributes.

The third type of anomaly is concerned with how metrics relate to each other when a combination of metrics are examined together. This indicates the need for a multivariate technique which deals with relationships (among many attributes).

The author investigated which of the three types of anomalies described above exist in practice using values from dataset 3 (see section 1.2). She subjectively identified anomalies using:

- the distribution of the attribute values;
- bivariate plots to identify general relationships (subjectively);
- Principal Component Analysis leading to a bivariate plot of the first two principal components.

The author discovered that any components exhibiting the third type of anomaly, had already been identified by either the first or the second type. Therefore, since no new anomalies were being highlighted, she decided not to investigate automation of multivariate detection except as a check on the anomalies detected by the univariate and bivariate techniques (this was felt to be required since relatively few data sets have been investigated). Therefore, the research effort reported in this thesis concentrated on finding statistical techniques which would automatically detect univariate and bivariate anomalies.

6.1 Automatic Univariate Anomaly Detection

Automatic univariate anomaly detection is employed to detect components which have

an attribute value which is significantly higher or lower than the rest of the components' values for that attribute. The technique the author considered for this was a modified version of Tukey's boxplot [38]. The reason this technique was considered is because it:

- provides a summary of the distribution of attribute values;
- does not require the values to follow a Normal distribution (or any other distribution);
- provides quantitative values outside which the value can be regarded as anomalous;
- for heavily skewed data boxplots provides more accurate summaries than the more conventional mean and variance statistics (see chapter 3 for examination of nature of software data);
- was designed to assist the detection of anomalies (outliers).

However, the technique only identifies statistically significant anomalies, which tend to be so extreme, that a project manager would already be aware of the problems. Anomaly detection was intended to do more than just highlight problems which were obvious to the project manager. It was intended to highlight components which were not necessarily a problem at present but might become problems later in the development process when it would be more difficult and expensive to correct. In software development, project managers are used to (and accept) the hypothesis that 20% of the components cause 80% of the problems, known as the Pareto effect. Therefore it was envisaged that a system which identified approximately 20% of components, as potential problems would be acceptable.

Working with Dr. Kitchenham and Sue Linkman, the author attempted to evaluate how successful a quality approach to the analysis of attribute values would be, [45]. Unusual quality indicator values at the design and code stage were identified and three factors were investigated:

- what proportion of components were anomalous with respect to each indicator metric and each quality characteristic (where a quality indicator was a metric collected during the design or coding phases of the development and a quality characteristic was an attribute collected during testing, e.g. faults and changes);

- the efficiency with which the indicator metrics identified 'critical' components, i.e. the proportion of components with anomalous indicator metrics that were also anomalous with respect to the quality characteristics metrics;

- the effectiveness with which the indicator metrics identified the 'critical' components, i.e. the proportion of components with anomalous quality characteristics metrics that were anomalous with respect to the indicator metrics.

Figure 6.1 shows the difference between effectiveness and efficiency.

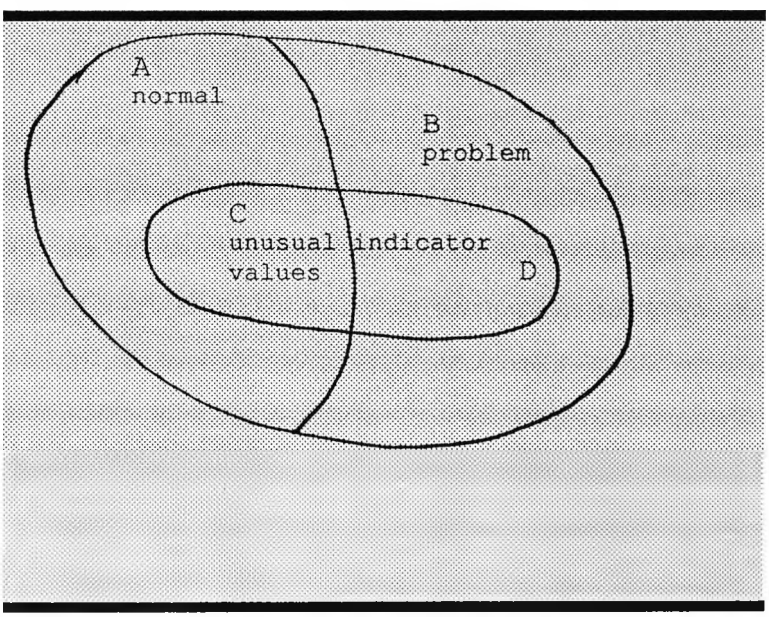


Figure 6.1

The total number of components is (A+B), where A is the number of components with normal characteristic attribute values and B is the number of components with unusual characteristic values. The total number of components with unusual indicator values is

(C+D). Therefore:

Efficiency = D/(C+D)

and

Effectiveness = D/B

The components were regarded as anomalous if the attribute values were greater than the upper fourth (i.e. lay in the top 25% of the data). In the boxplot terms this means that the cut-off values (C_U and C_L) were changed from:

$$C_U = F_U + 3/2d_F$$
$$C_L = F_L - 3/2d_F$$

to

$$C_U = F_U$$
$$C_L = F_L$$

where F_U and F_L are the upper and lower fourths respectively of the attribute value's distribution and d_F is the difference between the upper and lower fourths.

However, for an automatic anomaly detection system the percentage of components identified as anomalous by the above cut-off values is likely to be too high. This is because they will cause too many components to be identified as anomalous in a large system and this may lead to unacceptable burden on the project manager. Therefore, an automatic detection system should identify those components which are not necessarily formal outliers but are most likely to cause the project managers problems later in the development process. If the automatic system identifies too many components that the project manager may not have the time or the inclination to investigate them all and may stop using the system. This defeats the main purposes of the automatic system which is to increase the benefit of using measurement and provide more information in return for a minimum amount of time spent. This led to the author choosing cut-off values for the detection of univariate anomalies to lie between the original Tukey's values (i.e. + 1.5 d_F) and those given above (i.e. + d_F).

The most effective value of the d_F multiplier was chosen by examining which anomalies

were detected for three attributes of dataset 3 and then subtracting 0.25 of the tail lengths from the value and repeating the process. The value was taken to be too low if the cut-off points were classifying more than 20% of the data as being anomalous. Table 6.1 gives a summary of the results of this process and shows the average percentage of anomalies which the value of the d_F coefficient was detecting.

d_F multiplier	% of data identified as anomalous
0.75	~20-25
1	~10-15

Table 6.1 Percentage of Data Identified

With the multiplier higher than 1, the technique was identifying only one or two more components as outliers each time the procedure was invoked. However, when the coefficient was lower than 1, then there was a large increase in the percentage identified. This is probably due to the presence of a large number of tied values in some of the attribute measures and when the cut-off value close to a value with many ties it is starting to identify part of the majority of the data as anomalous. Therefore, the cut-off points which were implemented into the automatic system are:

$$C_U = F_U + d_F$$
$$C_L = F_L - d_F$$

The results from the investigation, using the upper fourth cut-off values, suggested that although detailed project control activities still rely on the expertise of the project manager, metrics could provide useful input to the design process both at the design and coding stages of the development process. For example, extra time spent on the 18% of the components with high fan-out values would have been 82% efficient and 45% effective, compared to 17% and 8% respectively, if extra time was spent on a randomly selected 18% of the components.

Percentiles of the data could be used by managers if they only wished to detect

univariate anomalies, e.g. look at the top 10% of the high valued components. However, the advantage of the boxplot technique is that it also provides an input for automatic interpretation of the anomalies.

6.2 Automatic Bivariate Anomaly Detection

A bivariate technique is required to detect anomalies that occur when a component has an unusual combination of two attributes. The relationship between any two attributes is not necessarily a linear one and it is difficult, if not impossible, to predict the relationship in advance.

In the absence of a known underlying distribution, no simple statistical technique exists which is designed to detect bivariate anomalies. The author decided to use a bivariate plot to show the relationship because a manager might use scatterplots and therefore would be likely to understand. While the univariate anomaly detection can be easily understood, using a bivariate plot it is not as easy to define an outlying point. This makes automation difficult, although more useful.

The author provisionally chose three statistical techniques as potentially useful for automatic detection purposes since they were capable of identifying atypical values when a combination of values were unusual. These techniques are described below in section 6.2.1.

6.2.1 Suggested Techniques for Bivariate Anomaly Detection

Three techniques for detecting bivariate outliers were considered. These techniques were:

(1) Nearest neighbour clustering;

(2) Sum of Euclidean distances from all other points;

(3) Frequency of points in a grid (i.e. density plot).

Investigation of other techniques, such as tensor analysis, was rejected due to lack of time.

It was envisaged that one technique would not identify the anomalies on every scatter plot, and that two techniques may be required. Visual inspection of scatter plots showed that problem components are not necessarily those that have attribute values which deviate from a direct relationship but can be those which fall between two clusters of points.

If two techniques had been necessary, then both of the techniques would have had to be automated as part of the automated tool or prototype. Both of the techniques would have to be applied and outliers identified by each method combined to produce one set.

6.2.1.1 Nearest Neighbour Clustering technique

This technique starts with each of the points (i.e. values related to a particular component) on a plot being regarded as a separate, individual group. A distance matrix, giving the Euclidean distances between each point and all others, is obtained. The technique then fuses the groups according to distance between the groups, with the smallest distance being fused first. Each fusion decreases the number of groups by one. A dendogram [55] shows graphically which groups have been fused, and in which order. Any point which is separate or outlying from the other point(s) should be easily detected since it will be the last point(s) to be fused. It was envisaged that this technique would probably be capable of detecting the type of anomalies shown in Figure 6.2.

Potential Problems Identified Prior to Use

This technique requires the attribute values to be standardised. It was unclear whether standardising measurements drawn form non-Normal distributions would have unexpected side-effects.

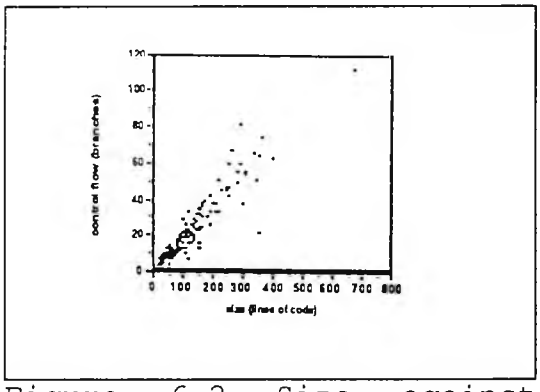


Figure 6.2 Size against Control Flow

The technique is unlikely to work when the points on the plot form a composite pattern, i.e. a pattern which has been derived from one or more patterns, as shown in Figure 6.3

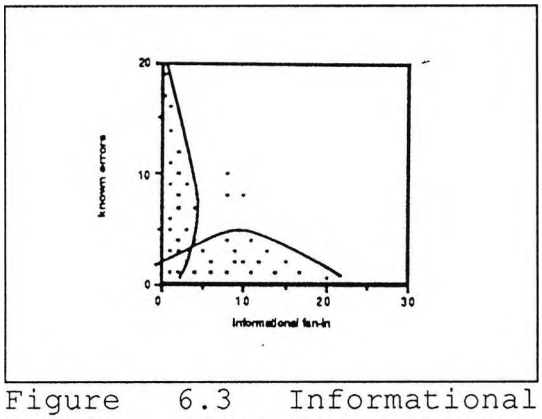


Figure 6.3 Informational Fan-in against Known Errors

There might be difficulty in transferring the information from dendograms directly into an automated interpretation system. The information is used by the statistical packages but the output from such packages is usually in graphical form, not numerical form.

The points added to the cluster last, which would be regarded as potential outliers, still have to be identified. This entails deciding what distance will be used as the cut-off point i.e. what is the maximum summed distance allowed before an individual component is identified as anomalous. This value may have to be related to the actual data rather than independent of the particular data set. This problem might be resolved by standardising the data first.

6.2.1.2 Sum of Euclidean Distances

This technique also requires a distance matrix with the Euclidean distance from each individual point to all other points. All that is then required is to sum all the distances from an individual point to all others and compare the sums. This may be automated by drawing a boxplot based on the summed value for each point.

Potential Problems Identified Prior to Usage

This technique requires the attribute values to be standardised. It was unclear whether standardising the metrics would alter the number of and which particular components would be detected as anomalous.

The technique is unlikely to work when the points on the plot form a composite pattern.

The points which would be regarded as potential outliers, still had to be identified. This entails deciding what distance the cut-off point will be, which this may have to be related to the actual data. This problem may be resolved by standardising the data first.

6.2.1.3 Density plot

This technique involves dividing the plot into grid areas and calculating the frequency of points in each area, as shown in figure 6.4. All the points which are in grid areas with less than a certain percentage of points in them will be regarded as potential outliers. At present, there appears to be no reason why this technique will not detect anomalies from all types of pattern.

Potential Problems Identified Prior to Usage

The optimum percentage cut-off level still had to be decided. There was also the problem of relating the grid to the range of attribute values. Standardising the metric

values might ease the problem because it is likely that only one standard grid would then need to be chosen.

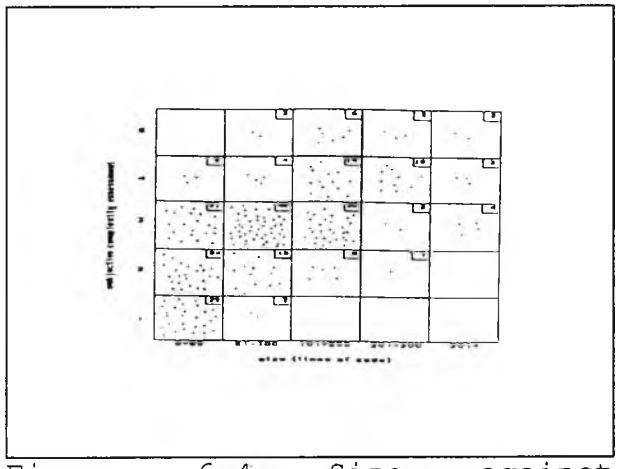


Figure 6.4 Size against Subjective Complexity Assessment

6.2.2 Choice of Statistical Technique

Since the object of the automatic detection was to emulate the performance of an expert project manager in detection of problems, it was decided to investigate what types of anomaly an expert project manager would detect and choose the statistical technique which detected the most of those anomalies. The author ran an experiment to identify what type of anomalies an expert project manager would detect (see chapter 4). The experiment had two aims, with the first being the major one:

(1) to determine what a project manager would identify as an unusual component from a scatter (or a density) plot;

(2) to determine whether initial standardisation of the attribute values affects the detection of unusual components.

The full description of the experiment was given previously in Chapter 4. The results from the experiment indicated that the managers' replies were not consistent enough to be used as a control in an experiment to identify the most appropriate technique to use for automatic anomaly detection.

Since the project manager's replies from the experiment could not be used to identify which of the techniques identified and described in section 6.2.1, was the most appropriate technique for automatic bi-variate anomaly detection, another way of choosing the technique had to be employed. The method used was to identify the technique which best replicated an expert consultant's subjective detection of anomalies on an available project.

Frequently, it is useful to consider pairs of variables as ratios, for example lines of code per month or defects per 1000 lines. At first glance this would appear to make the anomaly detection easier because it would reduce bivariate detection to univariate detection. However, the same ratio value can have different meanings, e.g. a large component with a large number of faults can have the same ratio value as a small component with a small number of faults. The individual attribute values are required for interpretation of the anomaly. For example, a large module with a small number of faults would have a low defect rate, and using ratio values would be assumed to be a good module. This would be incorrect if the low number of faults was actually a result of poor testing.

The technique of 'nearest neighbour' clustering was initially regarded as the most promising. One advantage of this technique was that as well as identifying the majority of the anomalies detected by the expert consultant, it also divided them into clusters of different types of anomalies. This helped with the interpretation of the anomaly. When the technique was used to identify the clusters, it showed some initial promising results from the identification viewpoint. However, the trial investigation also highlighted a problem with completely automating the technique. The data required to automatically identify the different clusters was not easily accessible from SPSS, the statistical package used. Also, if this technique was used in the automatic system the user would need to have available a large statistical package, which is expensive and requires a substantial amount of memory. Therefore, a technique was sought which could be relatively easily automated internally as part of the REQUEST prototype and therefore could be provided as an integrated part of the automated system to the user with minimum overhead cost.

The technique investigated was an amended version of the density technique. It was

amended to capture some of the capability of the clustering technique and because the density in the different grid cells alone was not sufficient to detect the anomalies. This was because it would identify all points in a grid cell as being anomalous if the density was low but took no account that the cell might be surrounded by the majority of the data, due to the placing of the grid not the distribution of the data.

The author decided to add a neighbouring concept to the density grid, which removed the above problem. The advantage of this approach was ease of implementation and linking to other facilities, like the interpretation prototype which is an important part of monitoring the development process. The reason that the link is so important is that the automatic detection system can only detect that a component has attribute values which are different to the majority of the components, it cannot detect whether this difference is likely to cause a problem or not. Since this would result in the project manager having to check many components that are not problems, it might discourage the use of anomaly detection tool unless a method of interpretation is also provided.

Certain analysis criteria had to be decided before the technique could be automated and implemented in the prototype:

- Size of grid to place on scatter plot;
- Rules for identifying whether a component is anomalous or not.

The initial grid size was chosen for software engineering reasons, not for statistical reasons. In software it is quite common to collect information on a five level ordinal scale. For example, defining complexity to be very low, low, average, high or very high. This led to the choice of dividing each axis into five equal sections. A useful enhancement would be to set a grid size which made more use of the information contained in the numerical measures in the data set and would therefore be more sensitive, e.g. use of percentile information.

The rules for identifying whether a component is anomalous or not were derived by analysing the values obtained from dataset 3, the same dataset that was used to choose the univariate detection technique. The objective was to find a set of rules which combined the influences of both the density of each grid cell and whether it was near

any other highly populated grid cells, to produce a technique which most consistently identified the anomalies that had already been identified subjectively by an expert consultant for the project. Different combinations of density and number of neighbouring cells were investigated on plots with different combinations of attributes. The plots used were the same as those used by the expert consultant. The set of rules eventually chosen for inclusion into the automatic system were those which identified most anomalies (if not all) on the majority of the plots.

The final set of rules which were implemented into the prototype are:

(1) density of grid cell is either one or two
and
number of non-empty neighbouring cells is less than three.

(2) density of grid cell is greater than two
and
density of grid cell is less than 10% of the total number of components
and
the number of non-empty neighbouring cells is less than two.

A component (or components) is identified as anomalous if either of the above rules is true. The implementation of these rules was carried out by another member of the REQUEST project who followed the author's specification.

6.3 Verification and Validation of Automatic Detection Prototype

The basic approach was to compare the results of the automatic anomaly detection routine with an analysis which was undertaken to show that the COQUAMO-2 model worked when the principles applied by an experienced analyst. The analysis was carried out using data from dataset 4. The analysis was carried through to the advice and the developers of the product verified that the majority of the components identified subjectively as anomalous by the expert were problem components. The analysis was completed prior to the Verification and Validation exercise and completely

independently of the tool. This analysis also met the criterion that the data set used had not been used to derive any of the anomaly detection rules.

Two anomaly detection routines were implemented in the prototype. These routines were an implementation of the rules for univariate and bivariate anomaly detection outlined in this chapter. They were implemented by a mixture of shell script and C programs, details of which are given in [56]. The aim of the prototype was to demonstrate that the techniques could be automated and were capable of detecting the same type and level of anomalies as an experienced consultant detected in the COQUAMO-2 model simulations, [53] and [57].

Since the results of the anomaly detection routines is part of the input to the interpretation system, verification and validation of the output of the routines was felt to be necessary to avoid advice being given on misleading information.

6.3.1 Verification Process

The tester who performed the verification was given a description of the rules that the univariate and bivariate anomaly detection should follow and then used this information to check whether the tool was executing the rules correctly.

The process used to implement the tests was to make use of the anomaly detection routines which were available as an user's option on the "User's Analysis Tool". This tool allows the users the option of inspecting and analysing their data at the lowest level of the summary report prototype.

The tester called up each of the analyses used in the manual analysis and checked whether the prototype obeyed the anomaly detection rules identified in this chapter. If the rules were not obeyed, the problem was flagged to the implementor who diagnosed the problem and implemented the solution. The analysis where the problem was detected was re-run and checked to see if the implemented solution had solved the problem. The verification uncovered four implementation problems. Once they were corrected the prototype appeared to be detecting anomalies correctly with regard to the

rules.

6.3.2 Validation Process

Once the verification process had been completed and the problems solved, the validation process was started. The tester looked at the same set of analyses but this time checked whether the anomalies which the tool had detected were the expected anomalies. The expected anomalies were the ones detected by the subjective identification performed by the expert. Any problems found were reported to the author.

6.3.2.1 Validation Results

The validation showed that the automatic univariate technique detected all the expected anomalies. The automatic bivariate technique detected 96% of the expected anomalies when all the anomalies from all the graphs were considered. Many of the anomalies were found on more than one plot and therefore were detected more than once, in fact 81% of the anomalies which the expert detected were found on every graph which the expert used. The automatic technique also identified a component as anomalous which the expert consultant had not. This component was not diagnosed as being problematic in the system under investigation but it caused serious problems when the particular component was ported to another application. This may have been coincidence but it would be interesting to check if the anomaly detection routines are capable of detecting components which will be problematic when ported or re-used in another application. However, this is outside the work of this thesis.

Four conditions were highlighted as problems, where the technique would have been expected to have identified the anomaly but did not (although the rules were correctly applied). The following plots show the anomalies which were only subjectively identified as anomalous in bold type-face and the anomalies only identified by the tool in italics. The rest of the marked anomalies were identified subjectively and by the tool.

(1) Figure 6.5 shows an example of where more anomalies have been identified than

expected. Component 4 has a combination of attribute values which is both feasible and acceptable but the rules have detected it as anomalous because it is the only component in the grid cell which has only one neighbour. This occurs when the majority of the data is captured in one cell of the grid and the rest randomly scattered around the plot. This will mainly be a problem with classified errors, which have many zeros, but will also cause a problem with any other attribute which has a large number of tied values and a low range of values.

Figure 6.5 also highlights the need for the grid setting to be different for attributes with a small range of values to avoid the situation of a line of grid cells whose limit boundaries fall between integers with no whole integer value contained in them and can therefore have no values in them.

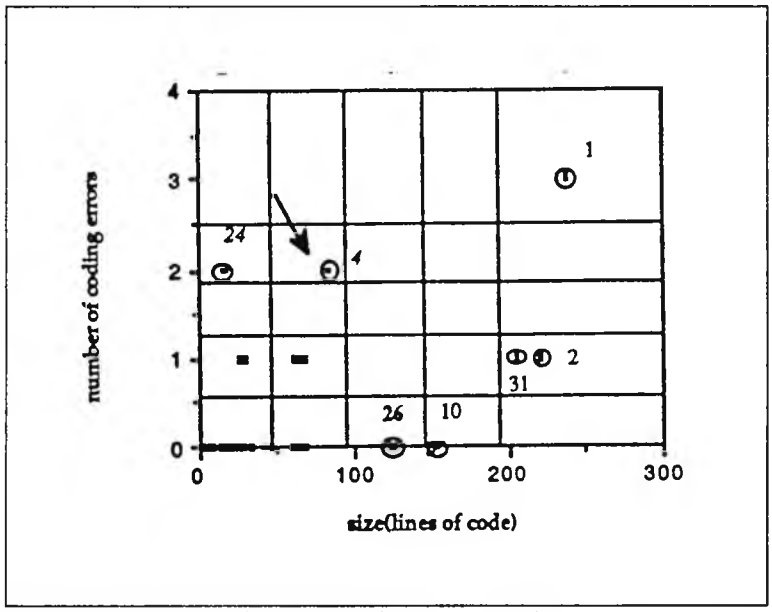


Figure 6.5 Size against Coding Errors

(2) Figure 6.6 shows an example of where the neighbouring rules are failing to detect all of the expected anomalies. The component marked with an arrow has not been identified because its grid cell has three neighbours although component 26 has been identified as anomalous.

This occurs when most of the values of one of the attributes fall in a small range. The range of the other attribute causes most of the data points to lie in

a vertical or horizontal line.

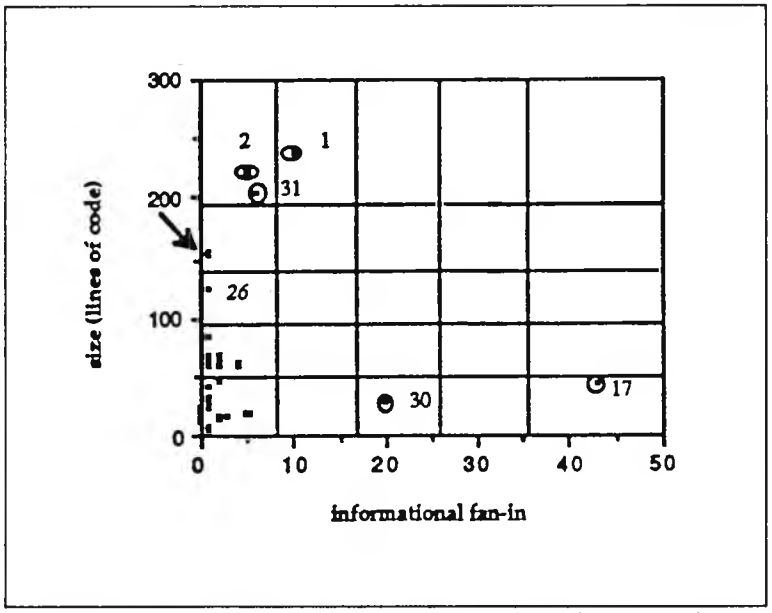


Figure 6.6 Informational Fan-in against Size

(3) Figure 6.7 shows another example of where the neighbouring rules are not detecting all of the expected anomalies. Component 10 has not been detected because it has three neighbours.

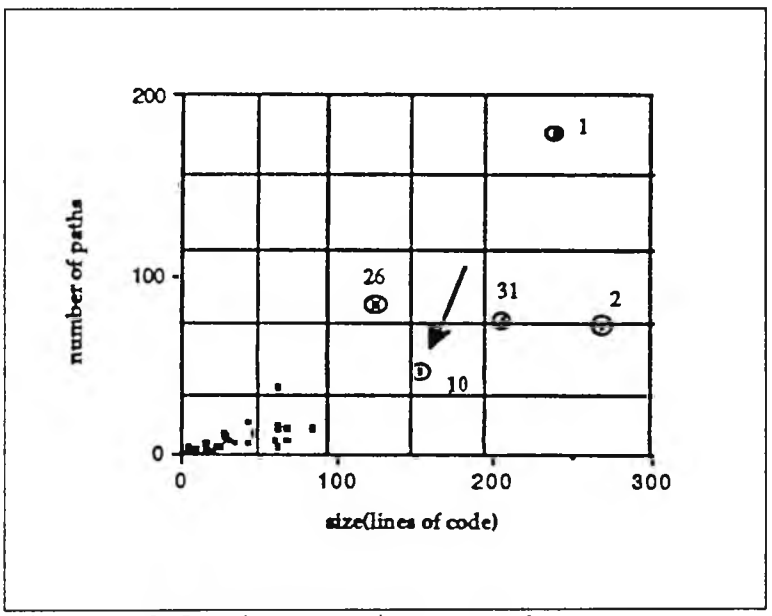


Figure 6.7 Size against Paths

This occurs due to the positioning of the grid lines on the plot. Although the

two components 31 and 2 have similar values for number of paths (or branches), the grid line has, by chance, been placed between these two values. This has resulted in the components being in different cells and they are both acting as neighbours to component 10, which is undetected since its grid cell has three neighbours.

(4) Figure 6.8 shows another example of where the neighbouring rules are not detecting all of the expected anomalies. Component 1 has not been detected because it has three neighbours.

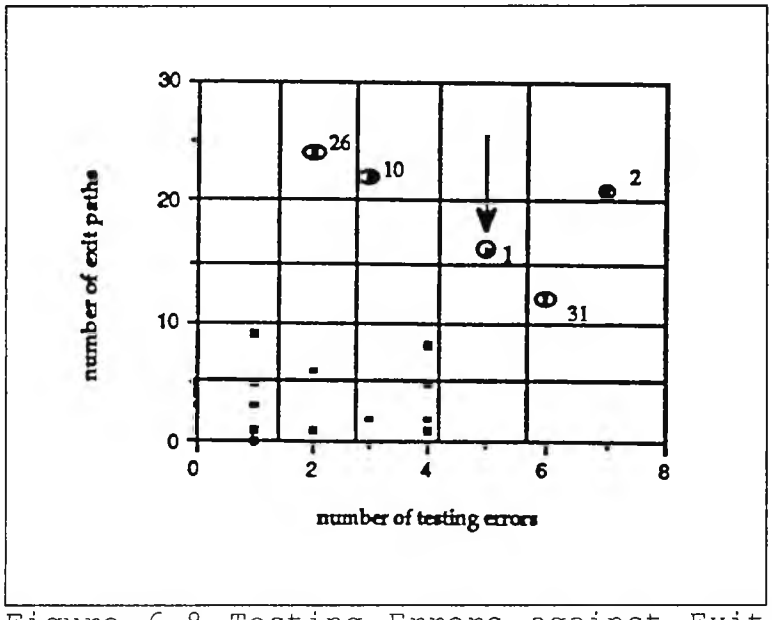


Figure 6.8 Testing Errors against Exit Paths

This occurs due to the positioning of the grid lines for the plot, similar to problem 3. However, this problem is caused by components 10 and 1 falling in neighbouring, diagonal cells, although they are relatively far apart in value. Therefore, they are both having an influence as a neighbour.

Problem 5 was not directly detected from plots in the validation exercise but could be foreseen as a result of those which were. The circled components are those which the anomaly detection rules would detect as anomalous.

(5) When all the data points lie in a diagonal line (see Figure 6.9), the cells with less

than 10% of the data in them will be detected as anomalous. This is because the grid cells will only have two neighbours, on the diagonal. The extreme case of this could cause all the components to be detected as anomalous.

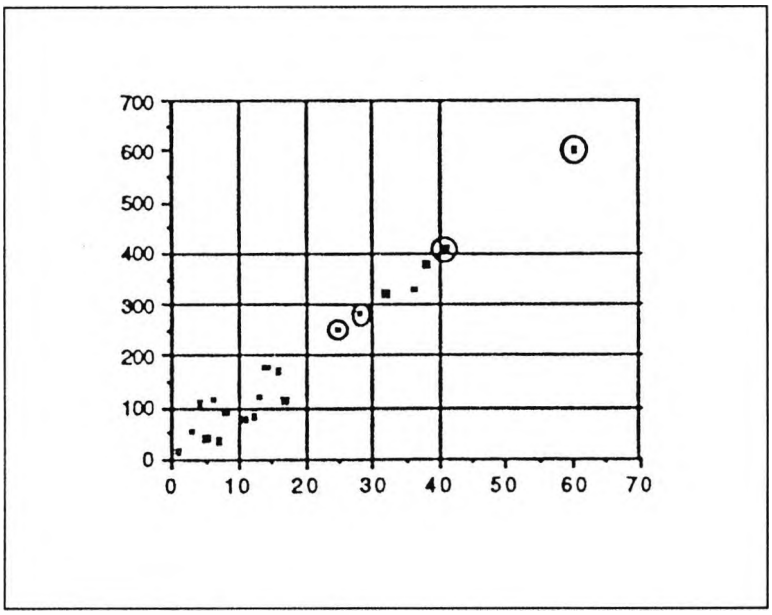


Figure 6.9

6.4 Summary of Verification and Validation Results

The system appears to be successfully detecting anomalous components and is largely consistent with the expert consultant, with only a few exceptions. The anomaly detection routines have been shown to be effective on a data set which was not used in the original definition of the rules and routines. The validation problems are mainly sensitivity problems caused by the process of setting the grid on the plot. However, these problems occur relatively rarely, except for the problem with attributes with a large number of tied values and low variability. Any future enhancements to the routines would have to include the solutions to the above mentioned problems. The rules for anomaly detection have only been used with data sets in the range of 50 to 240 data points. It is unlikey that the rules will be valid for either very small data sets (e.g. less than 20 data points) or very lrage data sets. More reserach is required to identify more appropriate rules for very large data sets. These rules would have to take account of the size of the data set.

7. Interpretation

This chapter is concerned with the interpretation of anomalies detected by project-based and component-based monitoring. The aim of the thesis is to identify statistical techniques which are appropriate for the analysis of software development metrics and to investigate how they might be used to support quality management procedures. This chapter shows how the likely cause of the anomaly can be detected with the help of a simple expert system which looks at a combination of attribute values for diagnosis.

The author's task was two-fold:

(1) To liaise with an expert consultant to obtain knowledge about what an expert considers when diagnosing a problem;

(2) To set up an simple expert system, or advice system which incorporates the knowledge obtained from the expert.

The potential anomalies were identified by the anomaly detection routine as described in chapter 6. The development of the system diagrams involved turning the expert's knowledge into interpretation trees and then into a set of rules for input into a rule-based system. The author did not write the expert system shell but was responsible for deriving the rules.

The interpretation system is intended to automate the interpretation of project and component anomalies. For automatic anomaly detection to be of any significant benefit to a project manager, some interpretation is required to identify the likely cause of the anomaly and its effect on the project. This was emphasised by the results of the experiment described in chapter 4. Automatic anomaly detection can only identify a component as being significantly different from other components, it cannot identify why it is different. An anomalous component may differ because there is a problem but it may differ because it has been exceptionally well developed. Beneficial anomalies occur for many different reasons, for example, a complex component might have been given to an experienced developer or subjected to more rigorous testing. To minimise time spent checking components, it is useful to differentiate between components that

are potential problems and those that are not. Without this facility, project managers may find an automatic monitoring system inefficient.

The objective of the automated interpretation system is to diagnose the likely cause of any anomalous components detected by the automatic detection routines. It cannot replace the project manager's experience, and was not intended to, because there can be many causes of an anomaly, the majority of which will be unforeseen or unknown to any general interpretation system.

7.1 Project -based Interpretation

Like anomaly detection, anomaly interpretation can occur at two different levels, project and component level. The concept of interpretation is the same for both except that a project-based anomaly detection is always assumed to be a problem. Project targets are set so that a missed target is a problem. However, unusual component values are not necessarily indicative of a problem.

Although both levels of interpretation need to identify the likely cause of the anomaly, the method used differs between the two types of monitoring.

For project-based monitoring, the interpretation of an anomaly, or missed target, depends on what the goal was when the target was originally set. For example, if project managers want to keep a check on how stable their projects are, they may want to set a maximum target on the rate of change per requirement (number of changes/number of requirements). If a project exceeds this target it may indicate instability of the requirements resulting in an unstable development.

The following tables provide suggestions as to why a project manager may wish to monitor the attributes on which initial default targets have been set in Chapter 5 "Project-based Monitoring.

Table 7.1 provides some suggestions as to why a project manager may want the information contained in the Phase Summary (see Table 5.2 for targets).

	Attribute	Reason for Monitoring
Resource constraints	effort	Check spending too much/little effort up to current phase. Too much effort may result in insignificant effort for testing. Too little effort spent may indicate some activity has not been properly completed
	duration	Check for schedule slippages
	re-work	Check for excessive amount of rework
Outstanding problems		Check efficiency of problem clearance process

Table 7.1 Reasons for Phase summary information

Table 7.2 suggests some reasons why a project manager may want to have the Level 2 Target information.

	Attribute	Reason for Monitoring
Resources	effort	Check adequate amount of effort spent on current phase
	duration	Check adequate amount of time available for current phase
	checking effort	Check adequacy of reviewing and inspecting
	other activities	Check amount of effort on activities other than development
	rework-faults	Check software/document not becoming unstable
Events	rate of change /req't	Check stability of requirements

Table 7.2 Reasons for Level 2 Information

Table 7.3 suggests some reasons why a project manager may want a review summary.

Attribute	Reason for Monitoring
av. # inspections per document/item	If number > 3 then inspection/review not being effective and software should be checked with a view to rewriting problem areas.
preparation effort	Check if spending adequate amount of time preparing for reviews.
av. # inspectors	Check on having too many (waste of money) or too few (not effective reviewing) inspectors.
inspection effort	Check spending adequate amount of effort in a review.
inspection rate	Check inspections are effective at finding faults.
clearance rate	Check clearing detected faults effectively.
# faults found	Check fault removal process is adequate (not too few) and check product not too faulty and likely to be fault-prone later.
category of fault	Likely problem if more than 25% of faults found are major.

Table 7.3 Reasons for Review/Inspection information

Table 7.4 suggests some reasons for monitoring process assessment.

	Attribute	Reason for Monitoring
Process assessment	prop.planned effort used	Check planned effort used too quickly with a danger of no effort left to test. May also indicate existence of unplanned for functions
	prop. of planned duration used	Check if likely to produce product on time
	check effort/ production effort	Check software/document produced too quickly at expense of testing effort.
Test plan	number of test cases	Check that planning of test cases is adequate and no unplanned functions developed
Readability	Fog Index	Check documents are readable.
Subjective assessment		Check quality attributes are at least average

Table 7.4 Reasons for Process assessment

Table 7.5 suggests reasons why a project manager may wish to see fault and change classification information.

	Distribution	Reason for Monitoring
Change classification	change request	Detect which type of fault is predominant.
Fault classification	discovery distribution	Check efficiency of review process.
	origin distribution	Check for problem phases
	category	Detect unexpected levels of fault types.

Table 7.5 Reasons for Fault and Change Classifications

It is not the intention of the interpretation system to collate the advice provided for each missed target and produce an assessment of the most likely problem for the project. The final view of how well the project is developing and the final identification of any problems remains the responsibility of the project manager. The automatic system is only intended to provide some additional information and advice to help the manager in this task.

7.2 Component-based Interpretation

Automatic interpretation is more important at the component level because it is not obvious what impact an anomaly may have on the development. Project managers are more likely to be interested in the underlying cause of the anomaly and what they have to do to resolve it. This was also emphasised by the results of the experiment, described in chapter 4, where some of the responders did not understand what the measurements or attributes meant and therefore could not identify when a component was anomalous. Therefore, interpretation is an important part of the monitoring process.

In the COQUAMO-2 prototype the results of both the project and the component-based detection are input to the interpretation system, however, the results of the component-based detection are not shown to the project managers as they are for the project-based

detection (via summary reports). The list of anomalous components detected is passed directly to the interpretation system, where the likely cause of the anomaly is assessed.

The component-based interpretation falls into two distinct parts:

- identification of likely cause of anomaly;
- corrective action for likely cause.

This chapter concentrates on describing the techniques that were used to identify the likely cause of a component-based anomaly. There are two distinct stages involved in the identification of the likely cause of the anomaly. The first stage requires summarising the nature of the anomaly by classification of the attribute values. The classified values are then used as input for the second stage which involves interpretation of the likely cause of the anomaly.

7.2.1 Classification

Prior to investigating the underlying cause of the anomalies, the nature of the anomaly needed to be summarised. This was done by classifying the attribute values onto a simple ordinal scale. This was required because the absolute values are not as important as relative values for interpretation. The classification scheme developed by the author was intended to be meaningful to project managers. It was based on defining an attribute values on the following five point scale:

- very low
- low
- medium
- high
- very high.

A consistent method was required to transfer all the attribute values to this form. The univariate automatic detection technique already required the calculation of a five-point summary of the attribute values, therefore the author used the boxplot information to

map the values onto the ordinal scale. This information is shown in Table 7.6 below.

Boxplot Range	Ordinal scale classification
below lower tail value	very low
between lower tail value and lower fourth	low
between lower fourth and upper fourth	medium
between upper fourth and upper tail value	high
above upper tail value	very high

Table 7.6 Transformation to Ordinal Scale Values

7.2.2 Interpretation System

The first step in choosing a method to automate component-based interpretation was to investigate what a consultant did. Although the component-based detection method was not based on multivariate methods, interpretation of anomalies is often based on many different attributes. Also, consultants use more information than the comparison of attribute values. They incorporate software experience when they decide what should be examined to explain why a particular combination of attribute values has occurred.

However, a consultant also takes into account the variety of conditions under which the project is operating. Dr. Kitchenham suggested that these different conditions could be captured by the use of scenarios. A study [58], incorporating the simulation of possible scenarios, showed the infeasibility of coping with many different possible conditions. The interpretation system was therefore restricted to the development of a simple expert system.

The interpretation system was required to cover all the phases which the anomaly detection system covers (from requirements to integration testing). The component-based interpretation deals with the phases:

- high level design;
- detailed design;
- coding/implementation;
- unit test.

The requirements and integration test phases are concerned with the project as a whole, so anomaly detection and interpretation is not relevant. These phases are covered by phase-based monitoring.

Due to the lack of available experts in the metrics area and the variation between expert opinion, only one recognised expert was used to obtain the original knowledge for the expert system rules. The expert was Dr. Barbara Kitchenham. The author and the expert decided to limit the identification of the likely cause to major problems which could occur in the development. This would inform the user whether the anomalous component was likely to be a major problem although it would not provide any help for more minor problems.

The expert's knowledge was documented in the form of a hierarchical tree for each phase and the attribute chosen to start each tree was the size of the component. A list of general problems which a project manager may face was identified. This list was reduced to major problems that could be found from investigating the component's attribute values. The resultant list of potentially dangerous conditions included:

- fault-prone components;
- inadequately tested components;
- unstable/difficult to enhance components;
- insufficient or excessive effort per component;
- complex component;
- difficult to test components.

The above set of conditions are relevant to the first three phases. Most conditions are detected at different stages of the development and can be either causes or symptoms. Causes and symptoms of problems are difficult to distinguish because at one stage of the development a condition might be regarded as a cause of another problem and the

next stage as a symptom. The automatic interpretation system was only concerned with identifying potentially dangerous conditions. It was not concerned with whether the condition was a cause or a symptom of some other problem, only that if would result in a major product problem. In practice, conditions can be caused or compounded by other conditions e.g. a component may be fault-prone because it is complex. Further, if the component is also given to inexperienced staff then the condition is even more serious. Currently, the automatic interpretation system stops when it identifies a potentially dangerous condition which is likely to result in a problem. This problem may have many causes which are not explicitly identified. A useful enhancement of the system would be to deal with multiple causes by continuing diagnosis after the first dangerous condition is identified. This would result in a more complete interpretation of the anomaly.

Unit test is conceptually different so it is not surprising that different types of problems are encountered:

- test planning inappropriate;
- test coverage not achieved;
- testing unsuccessful;
- unstable/difficult to enhance components.

If none of the problems are likely to be affecting the component then, within the limitations of the advice system, the component is assumed not to be a problem.

The different types of problem are investigated in a particular order (the same for all phases) by the comparison of attribute value classifications for each anomalous component. In many cases the attributes can only narrow the scope of the problem to its likely area. Additional information is required to identify the particular problem or explain the reason why the combination of attribute value classifications has occurred but it does not constitute a problem. The most common reasons for the combinations are queried by asking the project manager relevant questions when a particular combination is found. The interpretation tree for each phase is given in the following pages and has the following key:

plain text	-	questions internally answered (by attribute value);
underlined	-	questions externally answered (by user). These require a yes/no reply.
bold	-	likely cause of anomaly.

Figure 7.1 shows the interpretation tree for the high level design phase. The first characteristic looked at is the relative value of the high level design size. The answer to whether it is large or small will dictate which branch of the tree the user will be directed down and which set of questions will be asked.

The system will continue down the tree asking questions until a likely cause is identified. The main path down the tree represents the "no problem" path derived from the attribute value classifications. The branches either examine more attributes, ask clarifying questions or identify the likely cause. The answers to the questions either identify a likely cause, or identifies "no problem" and the flow of the tree returns to the main path. The interpretation trees for the other phases are as follows:

detailed design	-	Figure 7.2
coding	-	Figure 7.3
unit test	-	Figure 7.4

They all have the same structure.

At present, the interpretation trees are limited to major problems which may exist in the environment. However, because of the way they have been designed, the trees can be modified relatively easily. This means that the interpretation can be modified to reflect the problems which are most prevalent in the intended user's own organisation. Also, if particular attributes cannot be measured, and/or the problems they can highlight are uncommon in an environment, the attribute can be removed from the main path, with all its branches, without the need for modification to the rest of the tree.

Since the structure of the trees is consistent across the phases, it is easy to see where an attribute would need to be inserted into the tree for all the phases once its position in one tree is decided. At present, the system assumes the existence of only one likely

cause and stops when it identifies one. Therefore, the order in which the problems are investigated can be important. If a tree is amended to include checks for an additional problem, the significance of the problem should determine the position in the tree where the check should be inserted.

Once the system has identified a likely cause and terminates the diagnosis, it suggests a corrective action for that problem with the suggested diagnosis. A summary report is presented which provides the complete list of the anomalous components identified in a particular phase, their likely causes, and what diagnosis was identified in a previous phase (if any).

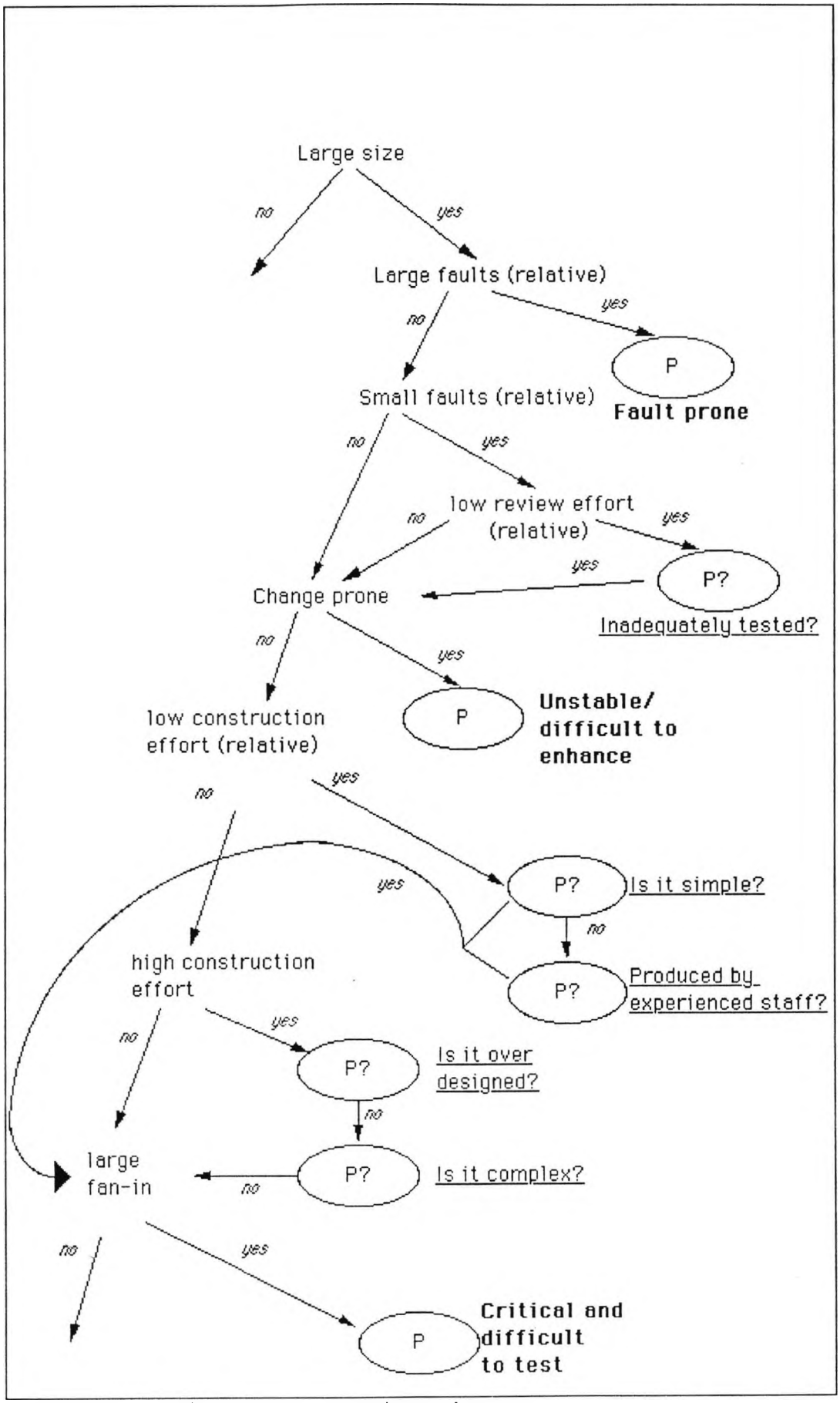


Figure 7.1 High level design phase

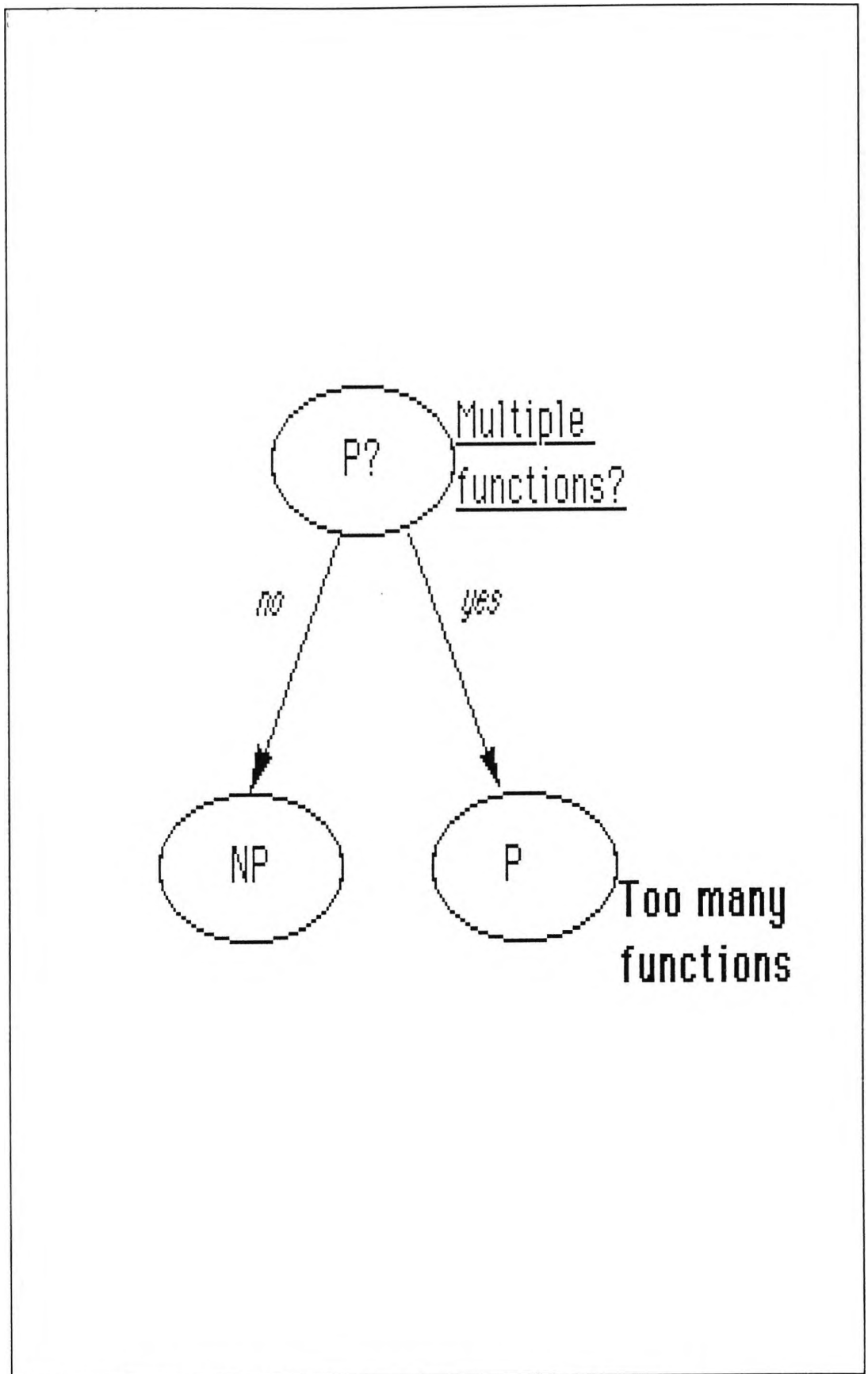


Figure 7.1 High level design phase (continued)

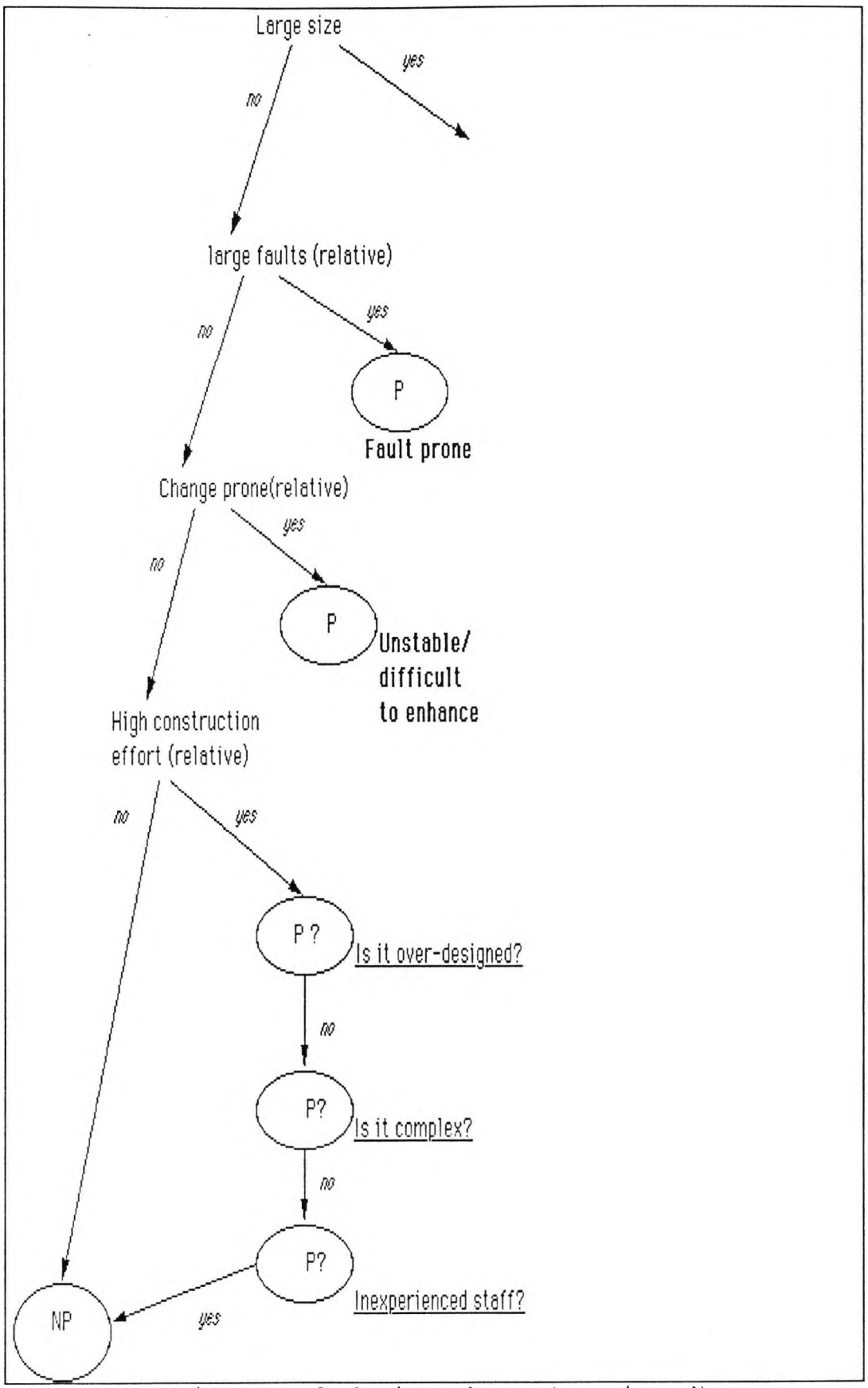


Figure 7.1 High level design phase (continued)

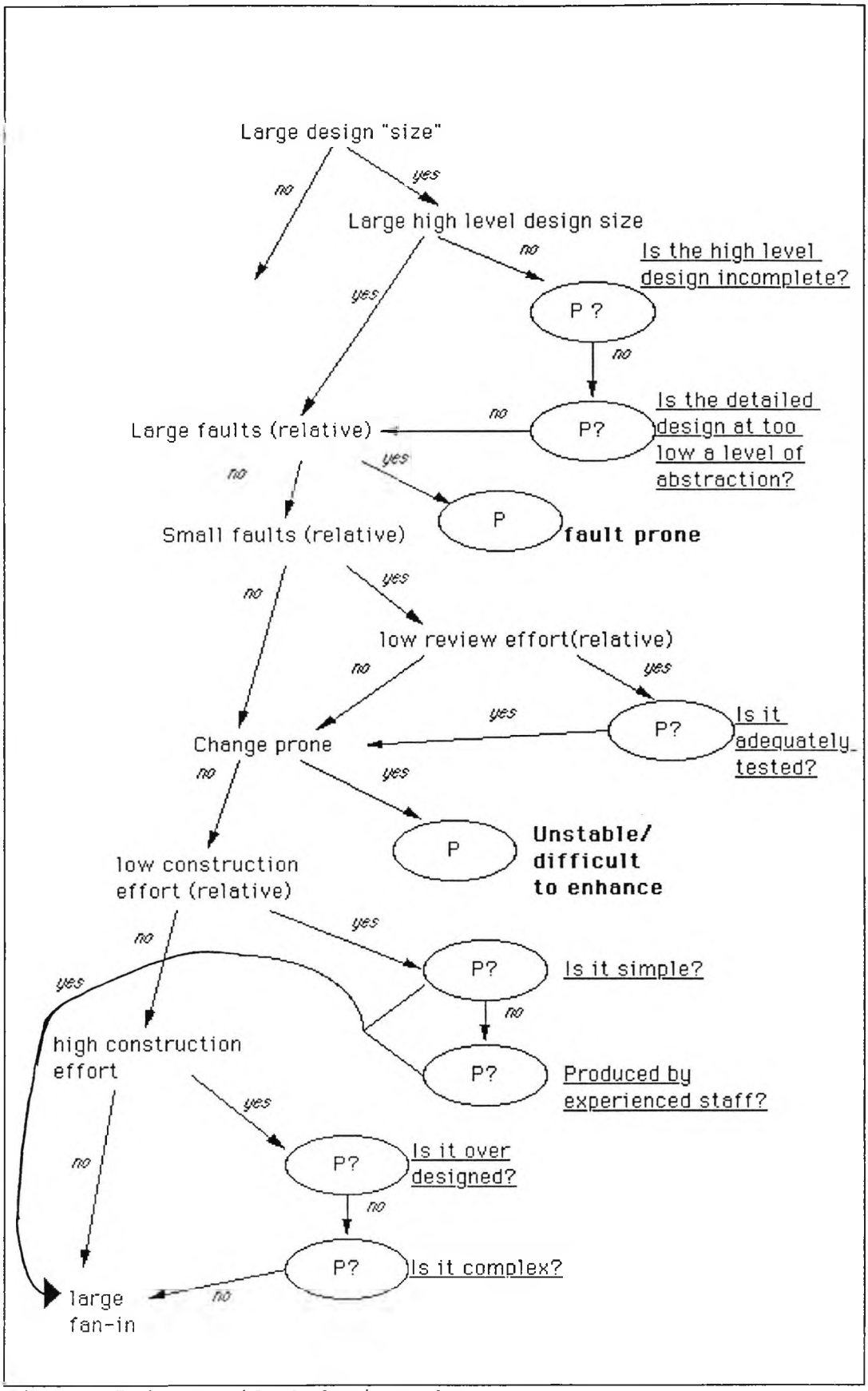


Figure 7.2 Detailed design phase

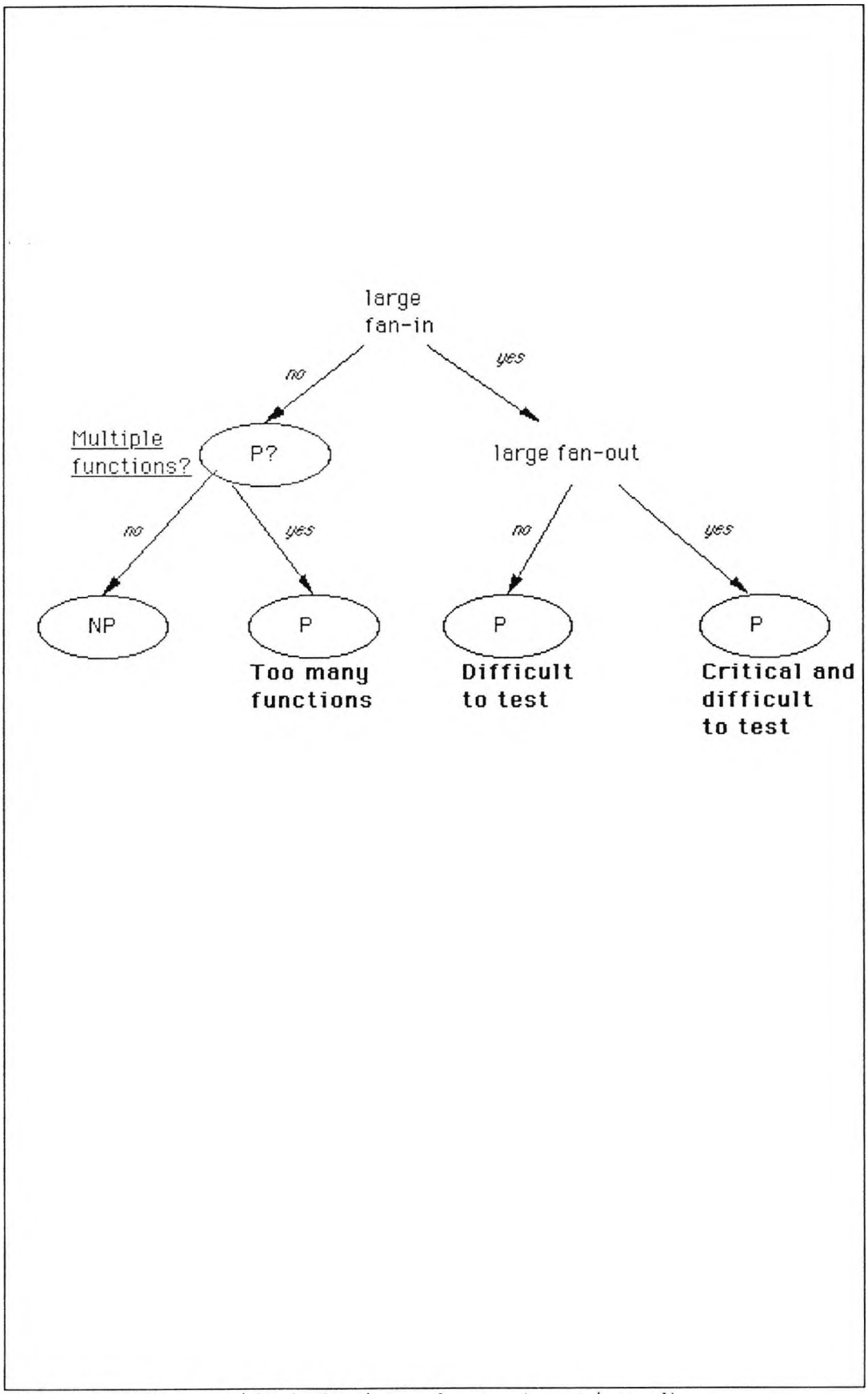


Figure 7.2 Detailed design phase (continued)

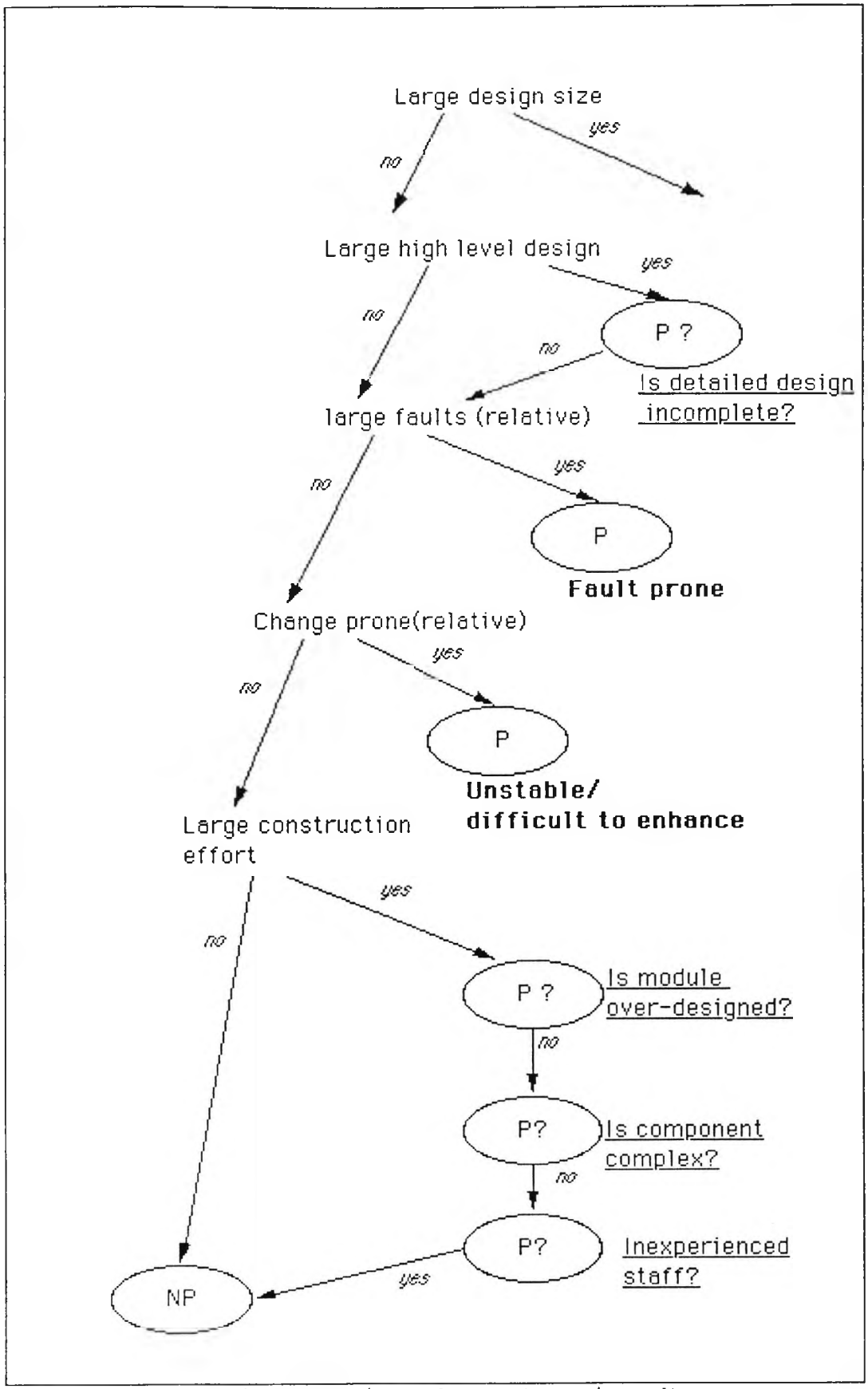


Figure 7.2 Detailed design phase (continued)

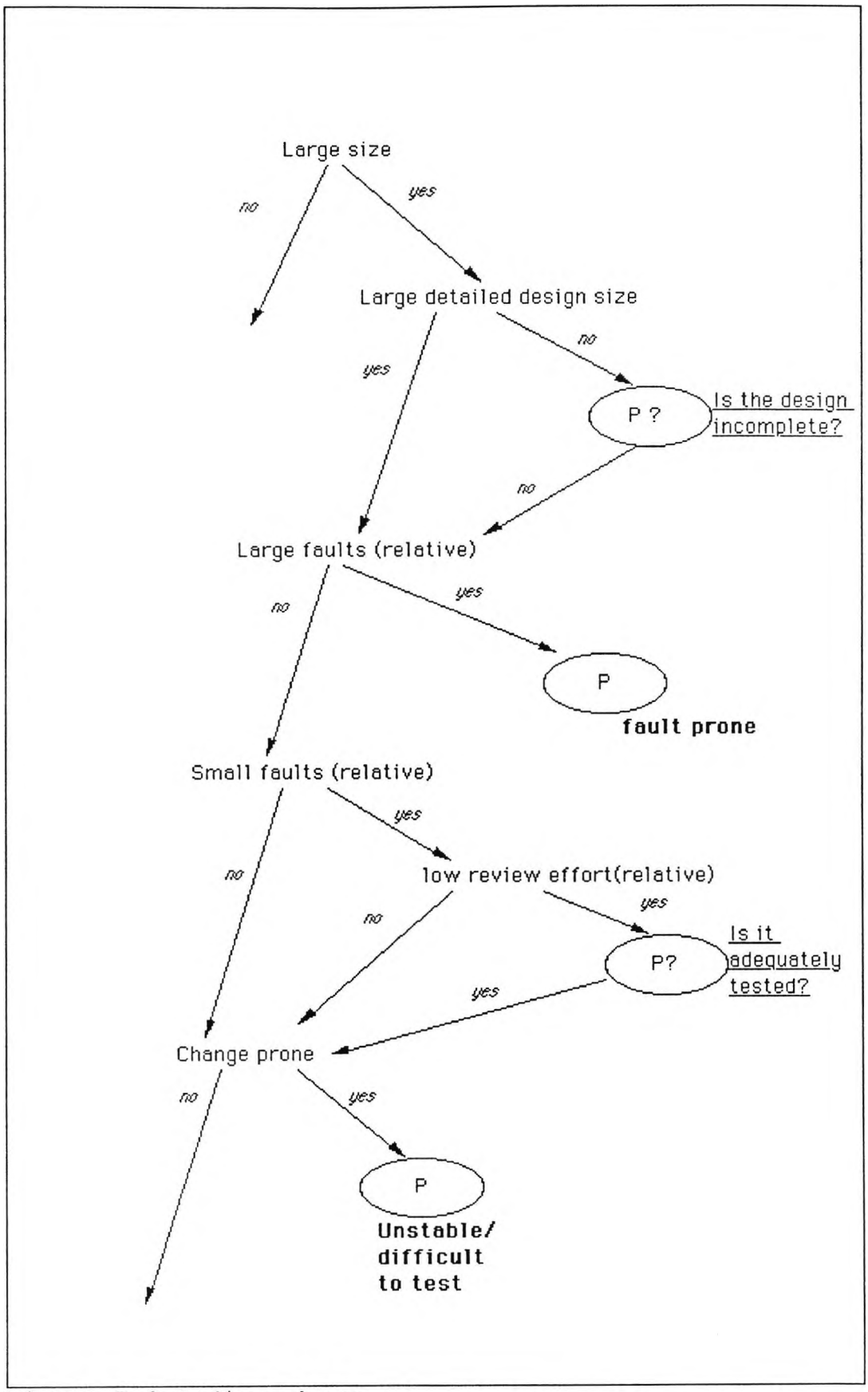


Figure 7.3 Coding phase

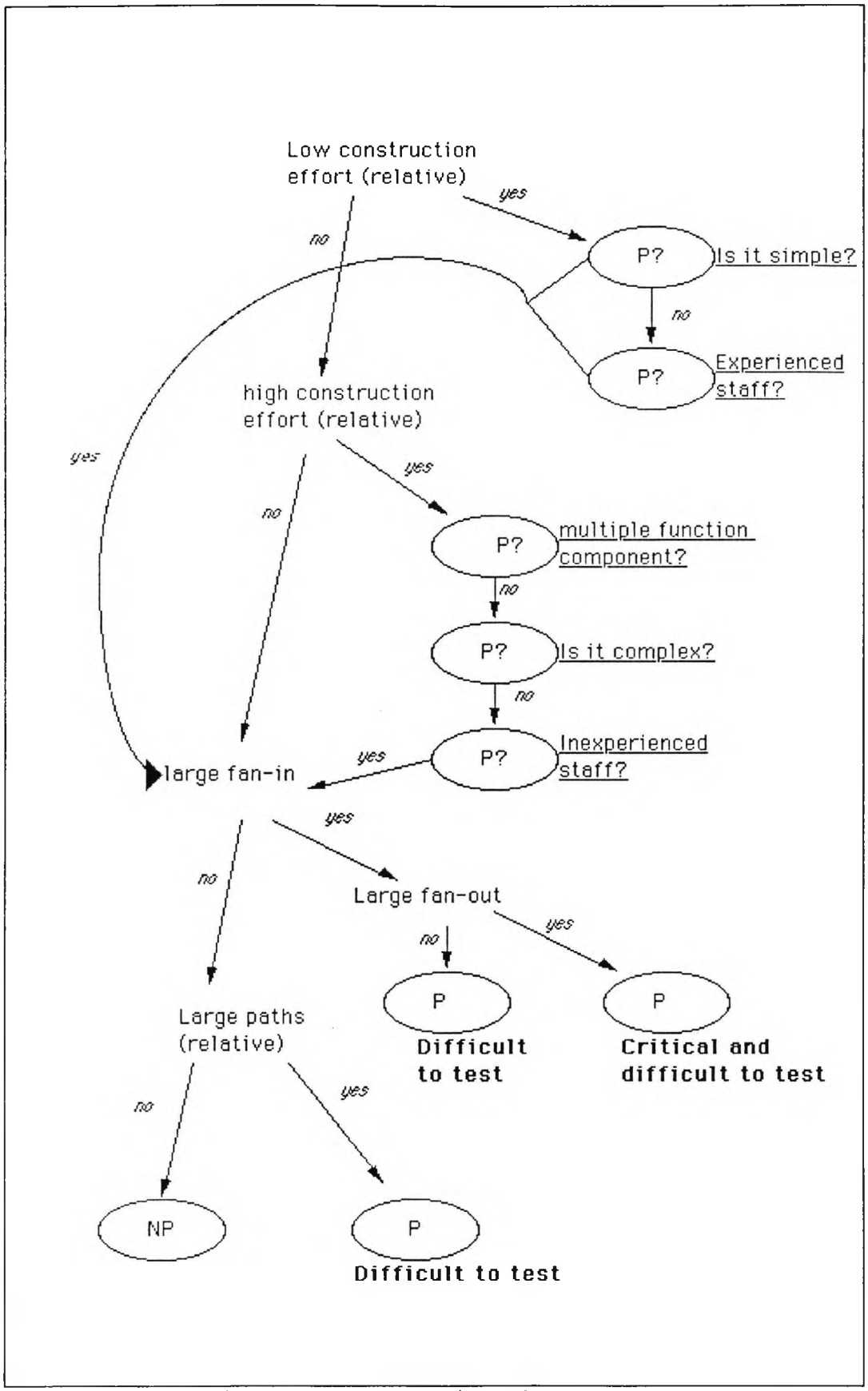


Figure 7.3 Coding phase (continued)

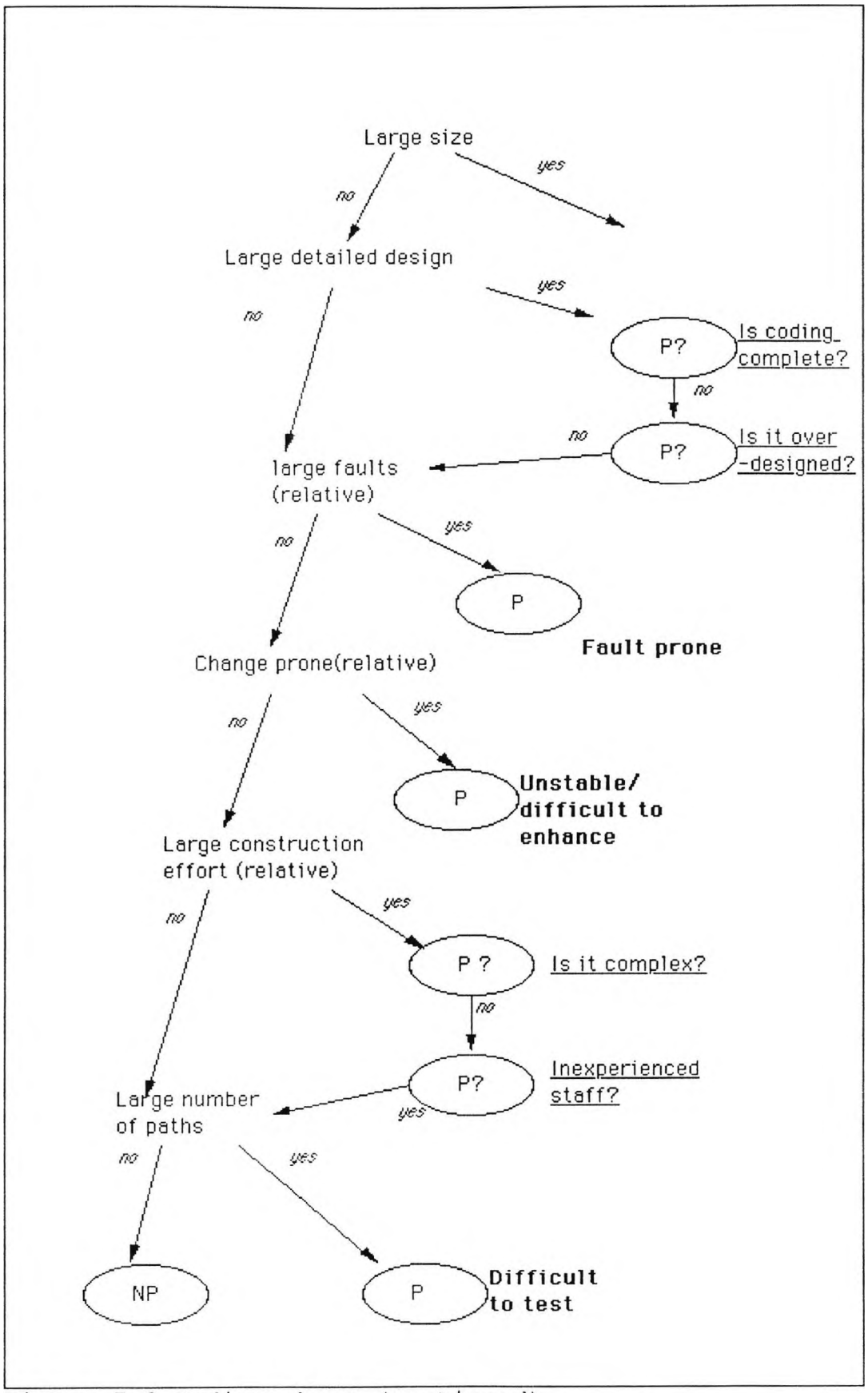


Figure 7.3 Coding phase (continued)

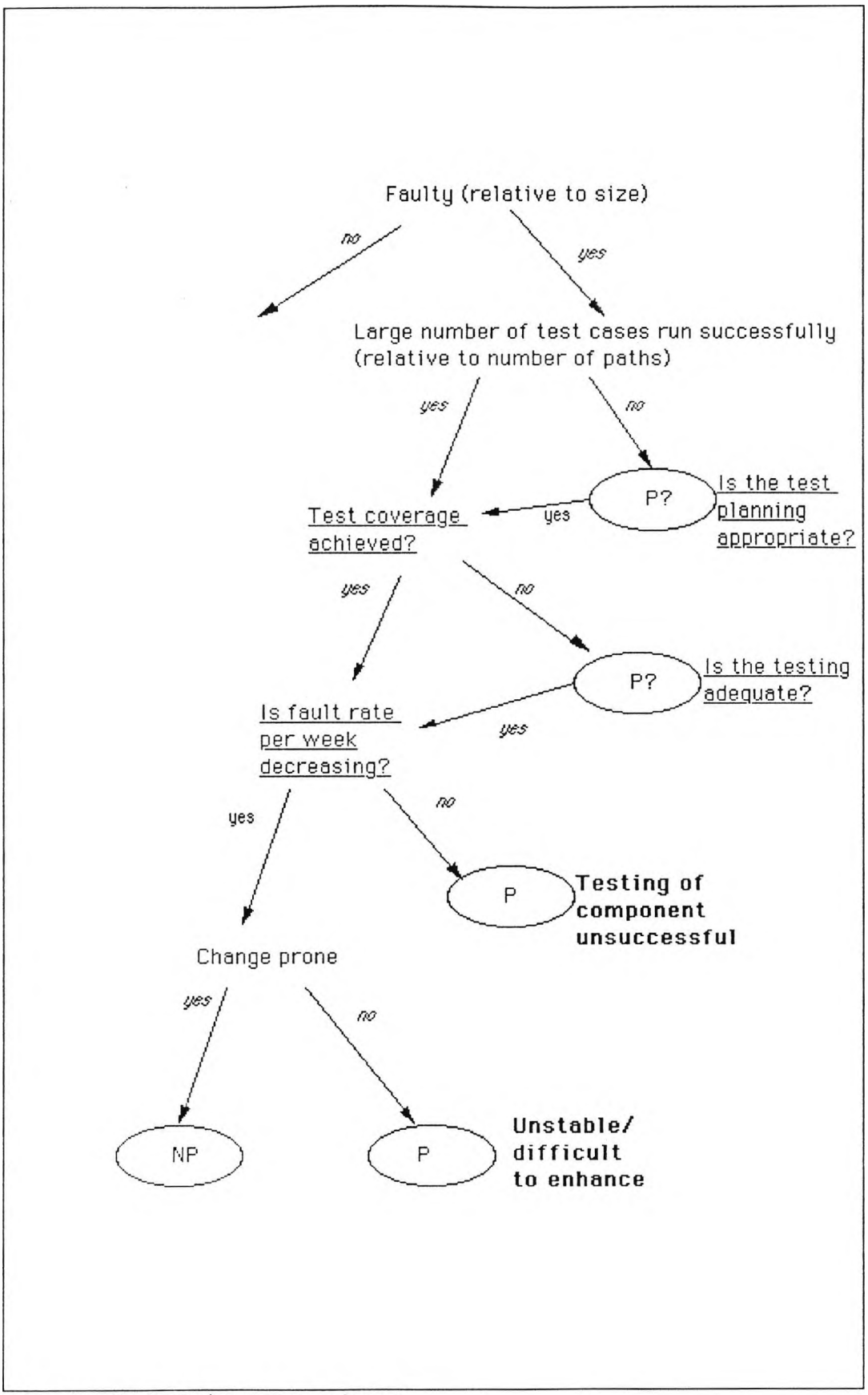


Figure 7.4 Unit test phase

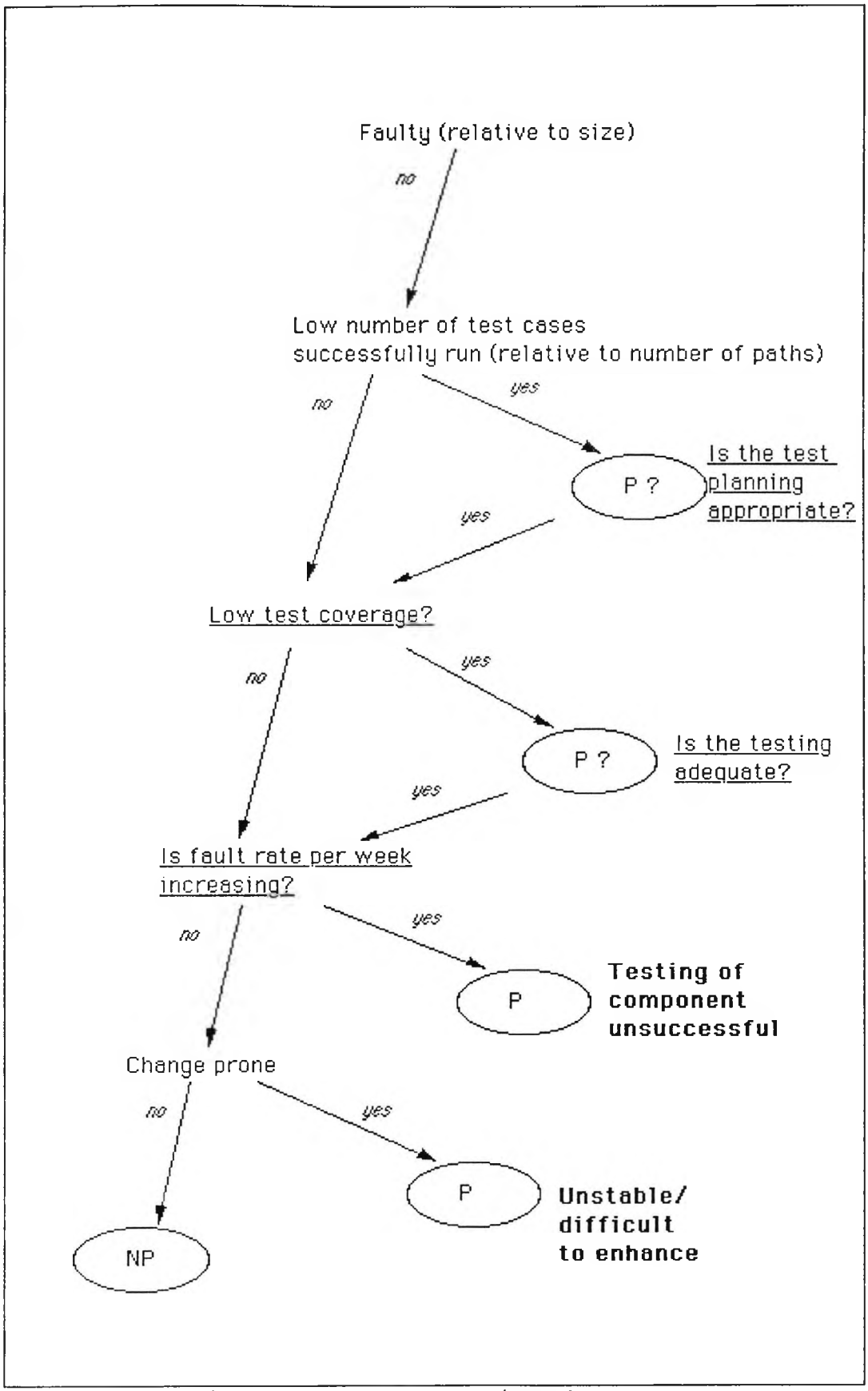


Figure 7.4 Unit test phase (continued)

7.3 Rules and Expert System Shell

The author implemented the interpretation trees as a file of Prolog rules for each phase. The expert system shell used to verify the rules is called "go", a proprietary STL product which was specially written for the this expert system. It is a primitive expert system shell written in Prolog and uses backward chaining logic. Figure 7.5 shows the architecture of the system.

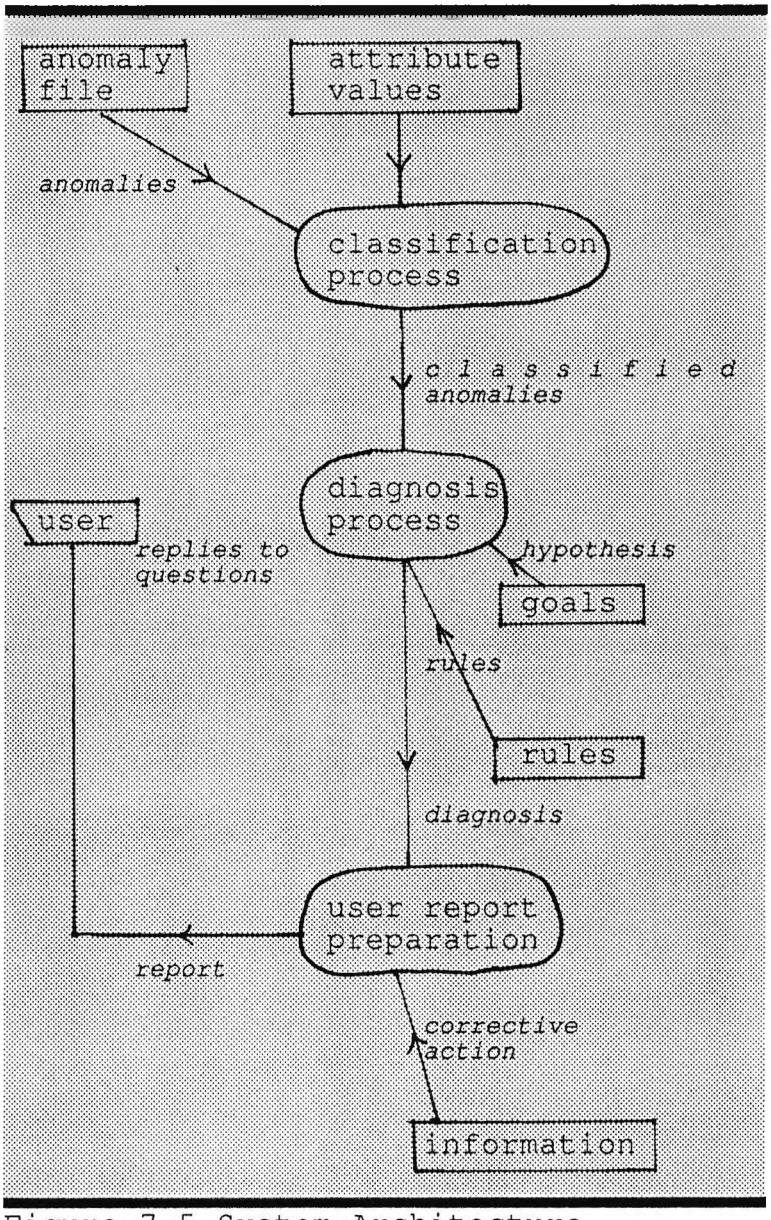


Figure 7.5 System Architecture

Since the expert system was developed using Prolog, all the names used in the rules must be valid Prolog atoms and each rule must end with a full stop. The system expects possible hypotheses, in this case "likely causes", and its rule-base must be supplied on

file. The attribute values can either be entered in a file or will be prompted for by the system. The system can save the conclusions from each run into a file if required.

The hypotheses are supplied in a file called "goals" and can either be explicitly affirmed, or denied, or voted on. Hypotheses which are affirmed when the system is run are reported in alphabetic order if the term "find_all_conclusion" is included in the rule-base. If this term is not included then only the first conclusion or likely cause is identified. If no hypothesis can be affirmed then the votes cast for all hypotheses are reported in numerical order. The format of the hypothesis is:

conclusion(<name of hypothesis>,<name of information file>)

At present, the interpretation system does not use the voting procedure since the rules are designed to generate one affirmed conclusion only. The voting system has been included because of the likelihood of enhancement to the rule-base (possible enhancements are described later in this chapter). The "information file" is used to store the corrective actions for the likely causes. The rule-base is supplied in a file called "rules". The rule-base contains both the knowledge base and the meta rules that allow the evaluation to be re-started. The knowledge-based rules are expressed in the form:

show(<name of intermediate hypothesis>,true/false) if <condition>.

or

conclude(<name of hypothesis>,true/false) if <condition>.

The attribute values are supplied in a file called "metrics". The system can deal with both absolute values and ordinal scale values as input since the classification routine converts the absolute values to ordinal scale. If any attribute value is not available the file then the user is prompted to input the value. As seen in the trees some of the questions asked of the user do not require any quantitative information they require a yes/no reply.

7.4 The Interpretation System Evaluation

The data set which was used to validate the anomaly detection procedure was used to validate the Interpretation system (dataset 4). This was also the dataset used for the evaluation of design metrics which was described in Chapter 3. The evaluator, who owned the data set, had not revealed what the problems were with the project prior to viewing the automatic system. He was asked to use the automatic system to identify potential problems, inputting his own data, and compare the results with what he had found during the development.

The results were very encouraging. The data set had two sub-system with a total of 56 components altogether. Of these, in the high level and detailed design, 8 components were detected as being a problem by the system. The user himself had detected 7 out of the 8 but had detected them at least one phase later in the project than the system had. Also, all the likely causes, except one, were correctly identified by the system. The one likely cause which was not identified was found to be the result of a minor error in the rules which, when corrected, resulted in the correct identification of the likely cause.

There was one potential problem component which the system identified but which the evaluator felt was not a problem. However, when the relevant component was re-used in another project, it caused many problems. The cause of these problems had been correctly identified by the system in the original project.

The evaluation of the implementation and unit test phases was completed after the work on REQUEST was stopped. This meant that the author did not receive a formal evaluation report from the evaluator. Informally, the major problems were detected and diagnosed correctly but many of the other identified components were diagnosed as "no problem". Therefore, it appears that the advice system is working correctly but the detection system was identifying too many components at the implementation and unit testing phases as anomalous to allow a detailed examination of whether the components were correctly diagnosed as no problem. The main difficulty appears to be that the detection threshold of the anomaly detection routine is too low for the implementation and unit test phases, resulting in too many components being classed as anomalous and

being passed to the advice system for diagnosis. It appears that the automatic monitoring system can correctly identify the majority of the project's major problems and can do so earlier than the project manager. However, further testing is required, using completely new data sets from different environments, before it can be inferred that the results are generally valid.

7.5 Future Enhancements

Two enhancements to the automatic interpretation system are envisaged. The first one is the addition of a "don't know" category for replying to questions. At present the worst case is assumed, for example, if the project manager is asked if a component is complex and does not know whether it is or not, the system assumes it is complex, and infers that this is the likely cause of the anomaly.

The second enhancement would only be undertaken if interest is shown in the system in its current state. The enhancement is to remove the assumption that an anomaly has a single underlying cause. This requires the advice to be enhanced to consider the situation when one cause has been identified but the others still have to be checked. At present, when a specific cause is identified the system exits from the diagnosis procedure. If the diagnosis were to check for coincident causes, the system would need to revert to processing subsequent parts of the tree. In practice, multiple causes are not uncommon [38], so this could be a useful enhancement.

8. Conclusions

The aim of my thesis was to identify statistical techniques appropriate for the analysis of software development metrics, and to investigate how they might be used to support quality management procedures.

The approach taken to achieve this aim was to:

- identify consistent/general relationships between software measurements collected during the development, and the subsequent fault or change-proneness of the product;
- identify measurements and methods for detection of atypical software components;
- construct stable, predictive models.

The results reported in Chapter 3 indicated that my first objective was not achievable. I was unable to identify any common relationships between datasets 1 and 2 when considering module relationships. Therefore identifying a consistent or general model between module attributes and product quality attributes seemed unlikely to be possible.

However, I did find that there were techniques capable of identifying outlier/anomalous components irrespective of the particular attribute. This led to the research goals being changed to:

- identify methods to automate anomaly detection;
- set up an advice system to help diagnose problems, that is, automate the interpretation of the detected anomalies.

Chapters 5 and 6 show that it is possible to use software measurements and statistical techniques to detect anomalies automatically. Chapter 5 also demonstrates that setting

targets or identifying expected values for particular software measurements can help a project manager control a project. Some default target values were suggested together with an acceptable range of actual values. Values outside the acceptable range would be recorded as a potential problem.

Chapter 6 described the statistical technique used to identify atypical values in the REQUEST prototype. The technique is a simple adaptation of the scatterplot principle that includes a measure of density to identifying atypical values objectively and automatically. The verification and validation exercise showed that, in this environment, the atypical values detected using this technique correspond to those that an expert consultant also identified as atypical.

The survey described in chapter 4 aimed to investigate how expert managers identify anomalous modules. The survey did not have the desired results because REQUEST incorrectly assumed that managers were familiar with the use of metrics for controlling their projects. The survey indicated that this assumption was not correct. However, it did produce some useful results since it identified the need for an advice system to interpret the reason for the atypical value and identify the problem components. It was clear that managers would not benefit from anomaly detection unless they were given some indication of what an anomaly meant.

Chapter 7 demonstrated that it was possible to develop an automatic anomaly interpretation tool. In general, use of a simple rule-based advice system should be capable of being calibrated to any environment by the input of the relevant advice in the rules.

Although the work reported in this thesis was incorporated into the COQUAMO-2 prototype, the prototype was never upgraded to a commercial tool. This is probably because project managers were not familiar enough with the use of software metrics for the control of projects for a tool to have been commercially viable at the end of REQUEST. However, consultants do appear to be aware of the benefits of measurement and are using measurements to identify problems during development. Therefore, it is likely that the prototype and the analysis techniques will be useful in the future but only after project mangers become more familiar with the concept of software measurements

and have had some training in its potential benefits.

The work since the thesis appears to be concentrating on the detection of unusual components. There is still a debate about whether intensive statistical analysis using sophisticated tools is better than simple non-parametric techniques. There seems to be a case for both, starting with the simple techniques and then, if necessary, applying the relevant sophisticated techniques.

There is still a major problem in moving between the different levels of the system (i.e. from component to system). Most of the current work on metric validation is being done at the module level, using faultcounts as a surrogate for reliability and number of changes for maintainability. There appears to be no evidence of any research addressing the problem of how to move between the levels. In fact, there has even been a step back in the standards arena, ISO 9126 is promoting the McCall et al's principle of a fixed number of general quality factors with a hierarchical decomposition in spite of the REQUEST criticisms. The ISO standard identifies six factors but does not suggest how these factors can be measured directly.

On a more positive note, although the COQUAMO-2 prototype has not been commercialised it is currently being used in a European funded research project called SQUID. In addition, the idea of using quantitative measurements for anomaly detection is generally accepted as a useful technique and is used in current text books [34].

9. References

[1] Frewin G D, Hamer P, Kitchenham B A, Ross N and Wood L M
Quality measurement and modelling - state of the art report
ESPRIT REQUEST Project Report R1.1.1, 1985

[2] Petersen P G
The COnstructive QUality MOdelling System
ESPRIT REQUEST Project Report R1.8.0, 1986

[3] Boehm B W
Software Engineering Economics
Prentice-Hall, 1986

[4] Gilb T
Design by Objectives
North Holland, 1985

[5] McCall J A, Richards P K and Walters G F
Factors in Software Quality, Vols I-III
Rome Aire Development Centre, 1977

[6] Boehm B W, Brown J R, Kasper H, Lipow M, Macleod G J and Merritt M J
Characteristics of Software Quality
North-Holland Publishing Company, 1978

[7] Bowen T P, Wigle G B and Tsai J
Specification of Software Quality Attributes
Final Technical Report, Vols I-III
Boeing Aerospace Company Engineering Technology, 1985

[8] Kitchenham B A, Wood L M and Davies S P
Quality factors, criteria and metrics
ESPRIT REQUEST Project Report R1.6.1, 1986

[9] Kitchenham B A
Towards a constructive quality model,
part 1: Software quality modelling, measurement and prediction
IEEE Software Engineering Journal, July 1987

[10] Petersen P G, Andersen O, Heilesen J H, Klim S and Schmidt J
Software Quality Drivers and Indicators
Proceedings from the 22nd Hawaii International conference on System Science,
Jan 1989

[11] Musa J D, Iannino A and Okumota K
Software Reliability: Measurement, Prediction and Application
McGraw-Hill International Editions, 1987

[12] Mellor P
Modelling Software Support Cost
ICL Technical Journal, Nov 1983

[13] Halstead M H
Elements of Software Science
Elsevier North-Holland, 1977

[14] McCabe T J
A Complexity Measure
IEEE Trans. Software Engineering, Vol SE-2, No.4, 1976

[15] DeMarco T
Controlling Software Projects
New York: Yourdon Press, 1982

[16] Garvin D A
What does 'Product Quality' really mean?
Sloan Management Review, 1984

[17] Brown P G
QFD: echoing the voice of the customer
AT&T Technical Journal 70(2):18-32, 1991

[18] Kogure M and Akao Y
Quality Function Deployment and CWQC in Japan
Quality Progress (October): 25-29, 1983

[19] Kitchenham B A and Walker J W
A Quantitative Approach to Software Development
Software Engineering Journal, 4, (1), 1989

[20] Doerflinger C W and Basili V R
Monitoring Software Development through Dynamic Variables
Proc IEEE International Conference on Computer Software and Applications, Nov 1983

[21] Basili V R and Ramsey C L
Arrowsmith-P - A Prototype Expert System for Software Engineering Management
Proc. Symposium on Expert Systems in Government, 1985

[22] Huff K E, Stroka J V and Strubble D D
Quantitative Models for Managing Software Development Processes
Software Engineering Journal, Jan 1986

[23] Selby R W and Porter A A
Learning from Examples: Generation and Evaluation of Decision Trees for Software Resource Analysis
IEEE Trans. Software Engineering, vol 14, No.12, Dec 1988

[24] Munson J C and Khoshgoftaar T M
Detection of Fault-Prone Programs
IEEE Trans. Software Engineering, Vol 18, No.5, May 1992

[25] Briand L C, Basili V R and Hetmanski C J
Developing Interpretable Models with Optimized Set Reduction for Identifying High Risk Software Components
IEEE Trans. Software Engineering, Vol 19, No.11, Nov 1993

[26] Briand L C, Basili V R and Hetmanski C J
A Pattern Recognition Approach for Software Engineering Data Analysis
IEEE Trans. Software Engineering, Vol 18, No.11, Nov 1992

[27] Selby R W and Basili V R
Analyzing Error-prone System Structures
IEEE Trans. Software Engineering, Vol 17, No.2, Feb 1991

[28] Agresti W W and Evanco W M
Projecting Software Defects from Analyzing Ada Programs
IEEE Trans. Software Engineering, Vol 18, No.11, Nov 1992

[29] Courtney R E and Gustafson D A
Shotgun Correlation in Software Measures
Software Engineering Journal, Vol 8, No.1, Jan 1993

[30] Schneidewind N F
Methodology for Validating Software Metrics
IEEE Trans. Software Engineering, Vol 18, No.5, May 1992

[31] Khoshgoftaar T M, Munson J C, Bhattacharya and Richardson G D
Predictive Modelling techniques for Software Quality from Software Measures
IEEE Trans. Software Engineering, Vol 18, No.11, Nov 1992

[32] Henry S and Lattanzi M
Measurement of Software Maintenance and Reliability in the Object Oriented Paradigm
Process and Metrics Workshop, OOPSLA 1993

[33] Rising L
An Information Hiding Metric
Process and Metrics Workshop, OOPSLA 1993

[34] Brooks I
Object Oriented Metrics Collection and Evaluation with a Software Process
Process and Metrics Workshop, OOPSLA 1993

[35] Anderson O
The Use of Software Engineering Data in Support of Project Management
Software Engineering Journal, Vol 5, No.6, Nov 1990

[36] Fenton N E
Software Metrics: A Rigorous Approach
Chapman and Hall, 1991

[37] Siegal S
Nonparametric Statistics for the Behavioural Sciences
McGraw-Hill, 1956

[38] Hoaglin D C, Mosteller F and Tukey J W
Understanding Robust and Exploratory Data Analysis
J Wiley & Sons Inc, 1982

[39] Kafura D and Canning J
A Validation of Software Metrics using many Metrics and Two Resources
Proc 8th International Conference on Software Engineering, 1985

[40] Meyer A & Sykes A
Probability Model for Analysing Complexity Metrics Data
Software Engineering Journal, Vol 4, No. 5, 1989

[41] Snedor G W and Cochran W G
Statistical Methods (7th Edition)
The Iowa State University Press, 1980

[42] Sprent P
Applied Nonparametric Statistical Methods
Chapman and Hall, 1989

[43] Dunn G and Everitt B S
An Introduction to Mathematical Taxonomy
Cambridge University Press, 1982

[44] Henry S and Kafura D
Software Structure Metrics based on Information Flow
IEEE Trans Software Engineering, SE7, 5, pp510-518, 1981

[45] Kitchenham B A, Pickard L M and Linkman S J
An evaluation of some design metrics
Software Engineering Journal, Jan 1990

[46] Yourdon E and Constantine L
Structured Design
Prentice-Hall Inc, 1979

[47] Myers G J
Composite Structured Design
Van Nostrand Reinhold Company Inc, 1978

[48] Henry S and Kafura D
The evaluation of software systems structure using quantitative software metrics
Software Pract. Exp. 14 (6) pp561-573, 1984

[49] Kafura D and Henry S
Software quality metrics based on interconnectivity
Journal Systems Software 2, pp121-131, 1982

[50] Troy D A and Zweben S H
Measuring the quality of structured designs
Journal System Software, 1982

[51] Kitchenham B and Pickard L
Part II: Statistical Techniques for Modelling Software Quality in the ESPRIT REQUEST Project
Software Engineering Journal, Vol 2 No.4, July 1987

[52] Pickard L M
Analysis of Software Metrics
Measurement for Software Control and Assurance (B A Kitchenham and B Littlewood eds.)
Elsevier Applied Science, 1989

[53] Kitchenham B A
A COQUAMO-2 Evaluation Study
ESPRIT REQUEST Project Report R1.18.13, Mar 1988

[54] Kitchenham B A
Specification of COQUAMO-2 Report Format
ESPRIT REQUEST Project Report R1.7.8, Dec 1988

[55] Everitt B
Cluster Analysis
Halstead Press, 2nd Ed., 1980

[56] Linkman S G
Description of the Implementation of Report Component
ESPRIT REQUEST Project Working Paper, May 1989

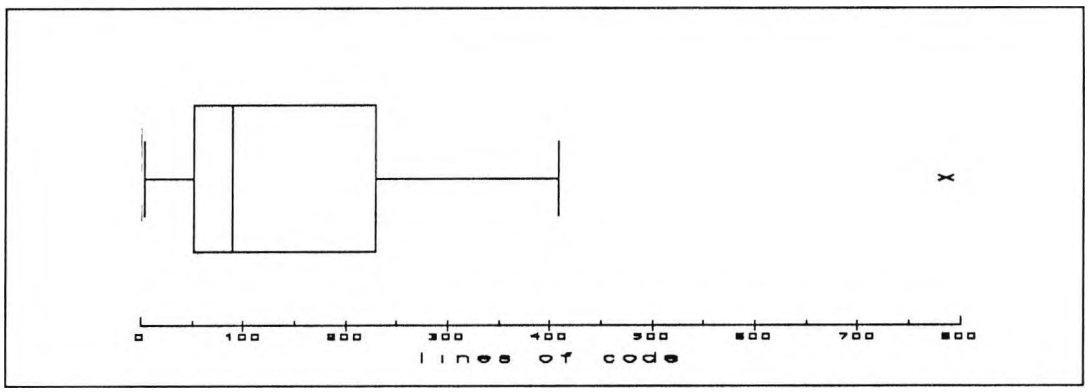
[57] Kitchenham B A, Andersen O and Klim s
A COQUAMO-2 Analysis of the DK-001/COM Data Set
ESPRIT REQUEST Project Report R1.10.1, Nov 1988

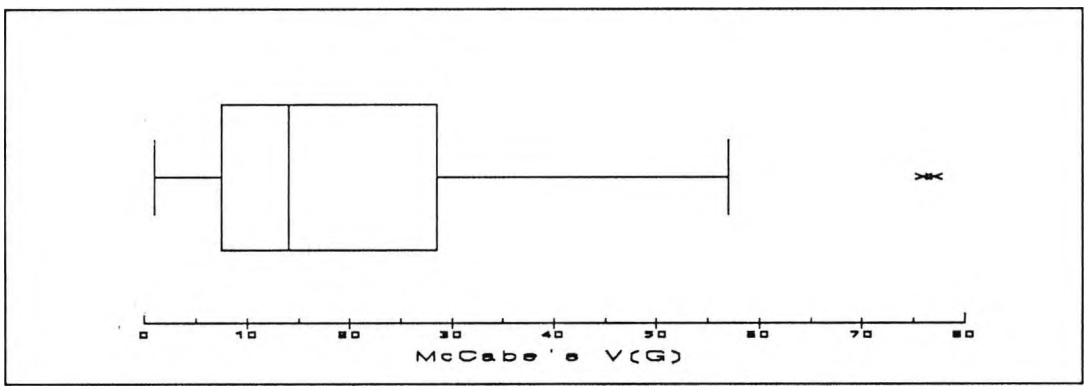
[58] Heilesen J H
Possible Scenario for COQUAMO-2
ESPRIT REQUEST Project Report R1.7.4, June 1988

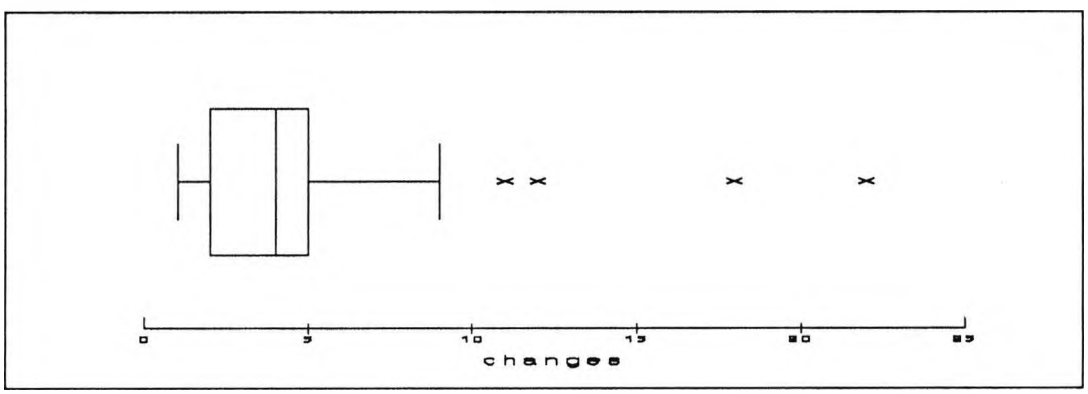
[59] Walker J G and Kitchenham B A
Quality Requirements Specification and Evaluation
Measurement for Software Control and Assurance (B A Kitchenham and B Littlewood eds.)
Elsevier Applied Science, 1989

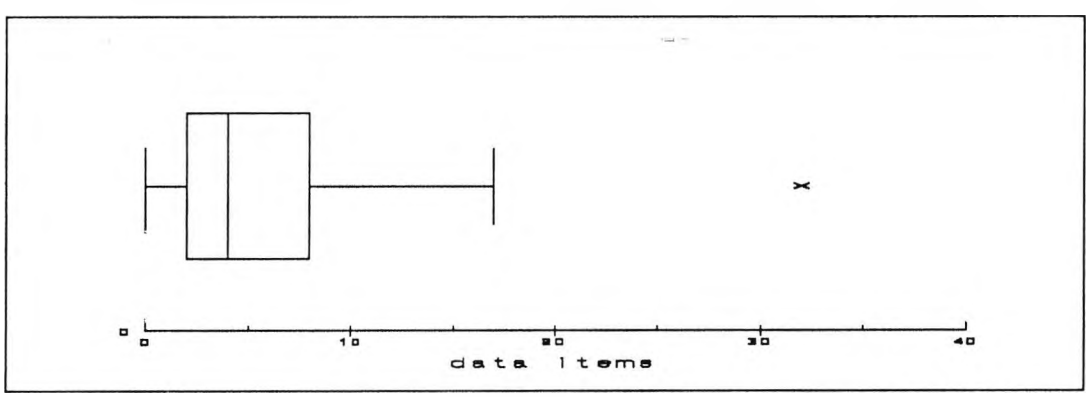
Appendices

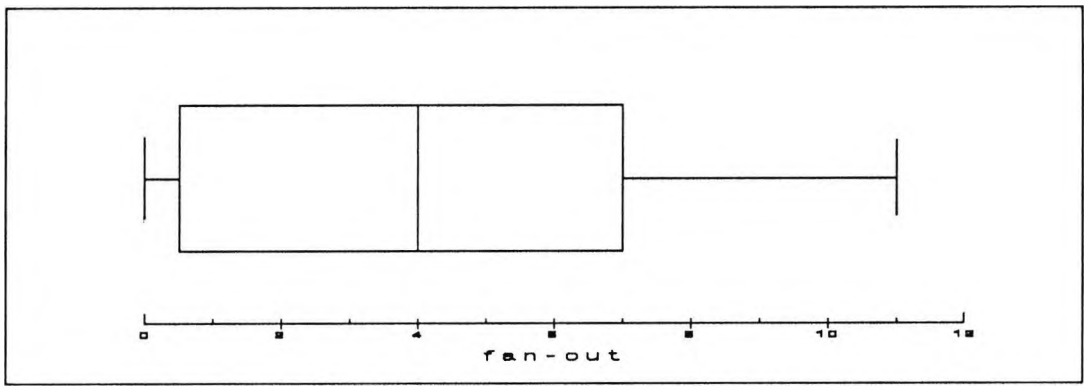
Appendix A: Boxplots for Dataset 1

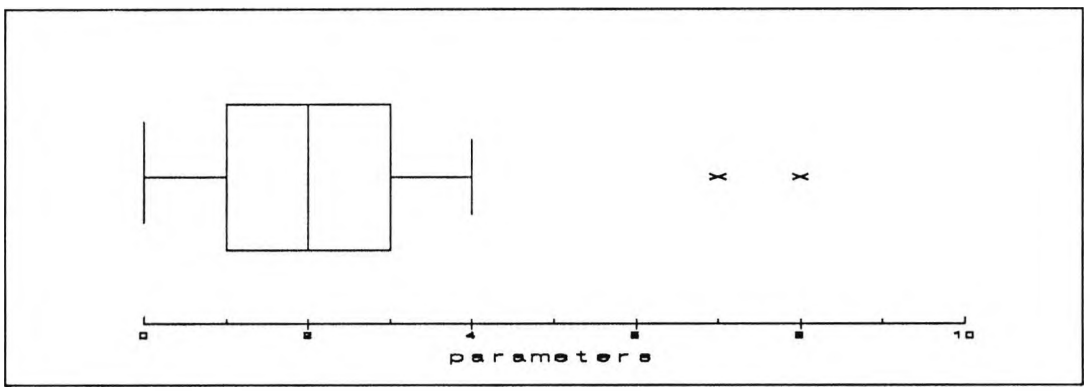

lines of code

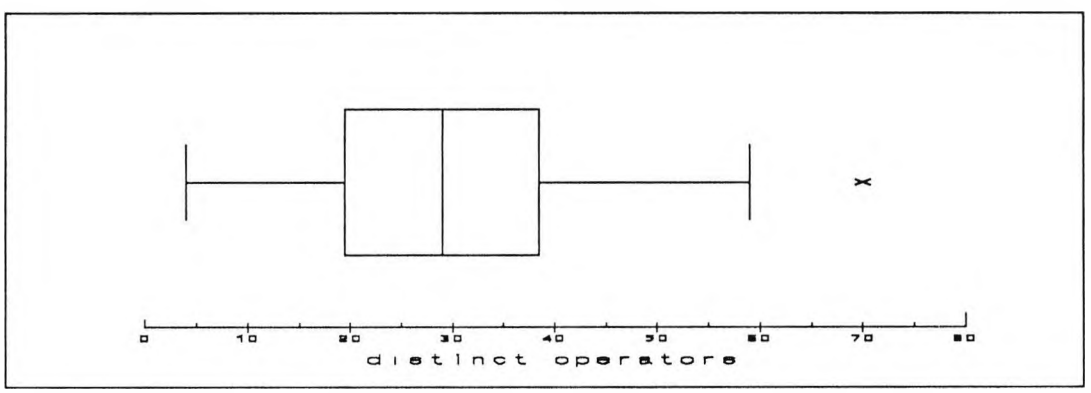

McCabe's V(G)

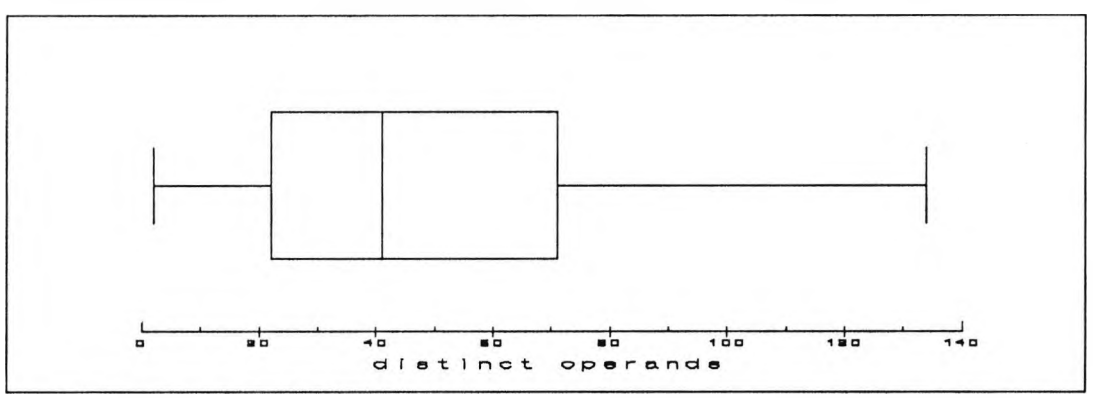

changes

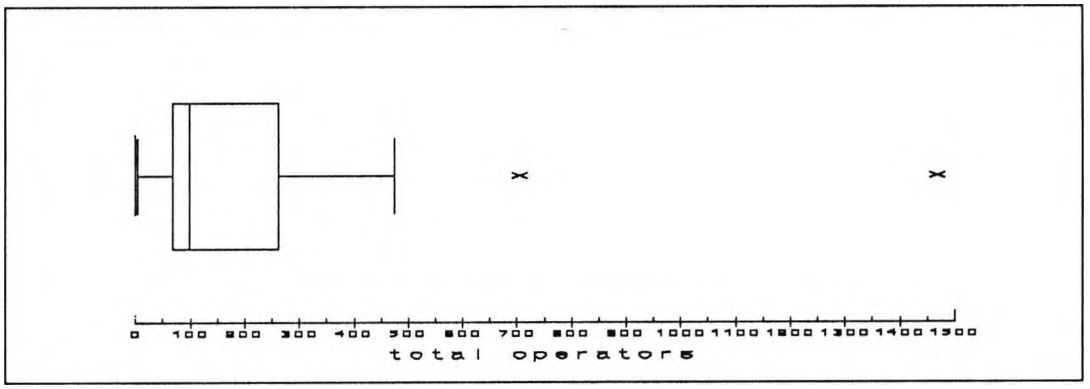

data items

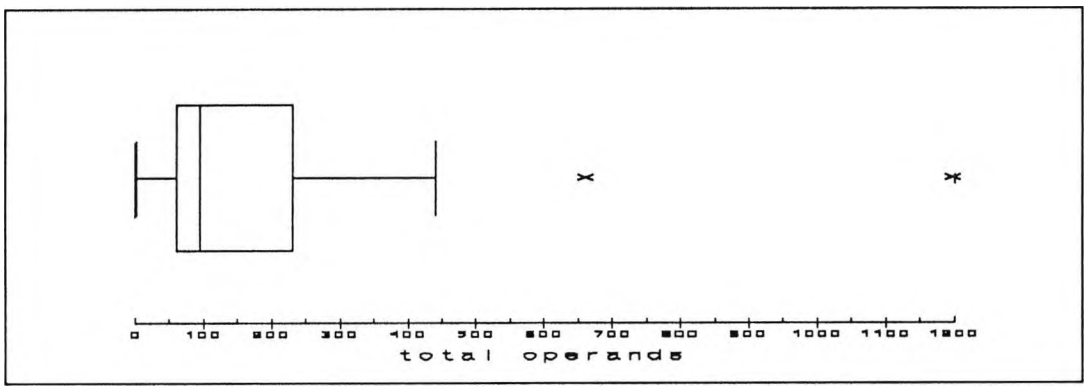

fan-out


parameters

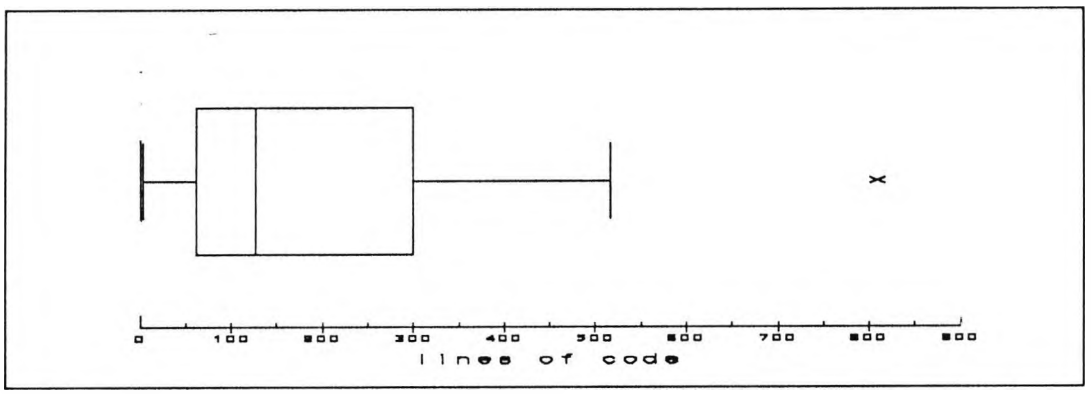

distinct operators

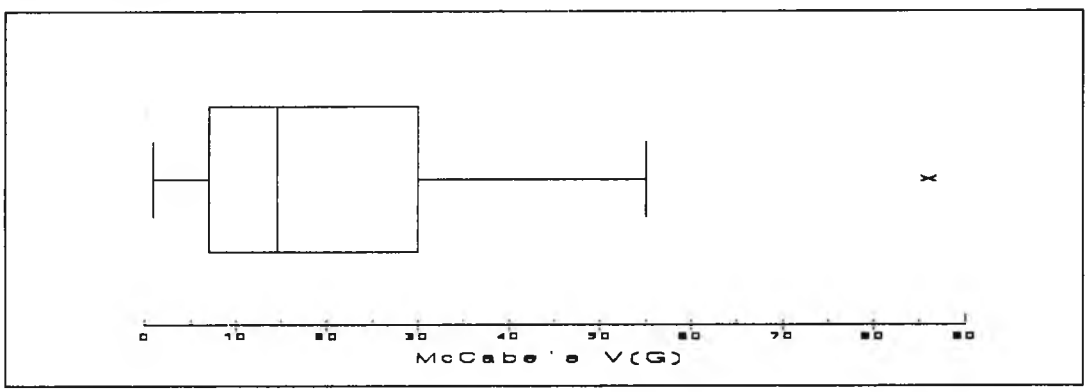

distinct operands

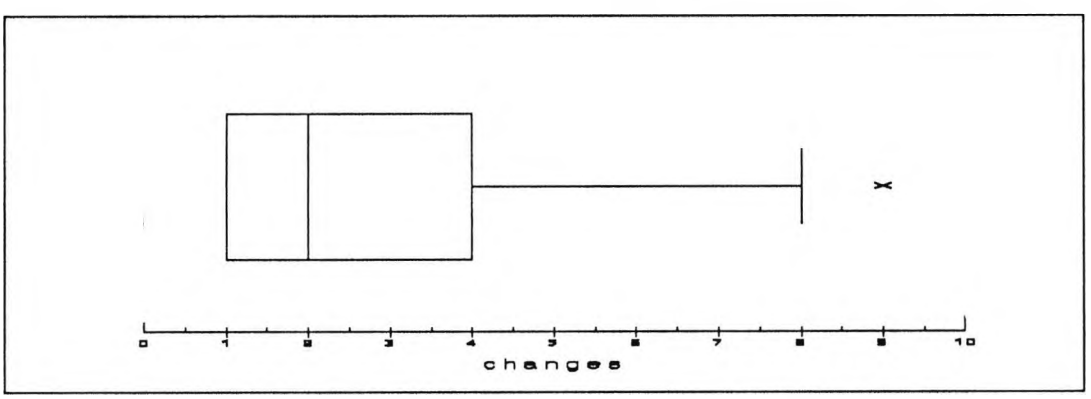

0 100 200 300 400 500 600 700 800 900 1000 1100 1200 1300 1400 1500
total operators

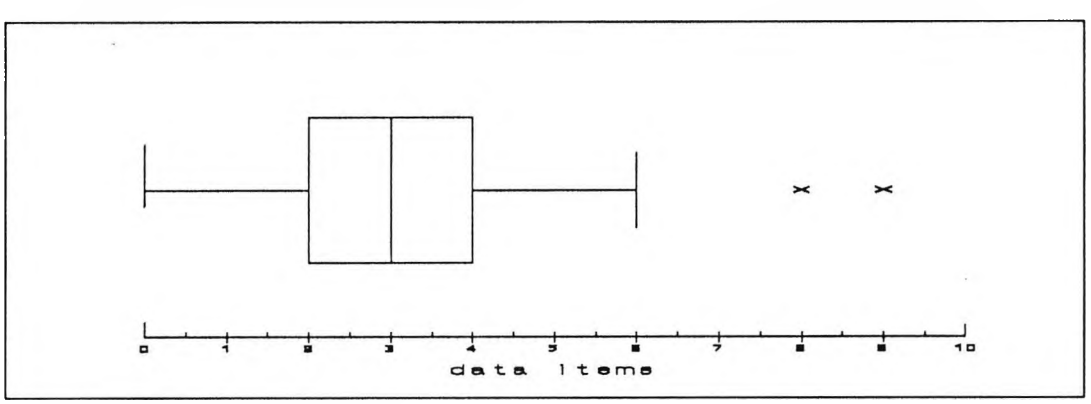

0 100 200 300 400 500 600 700 800 900 1000 1100 1200
total operands

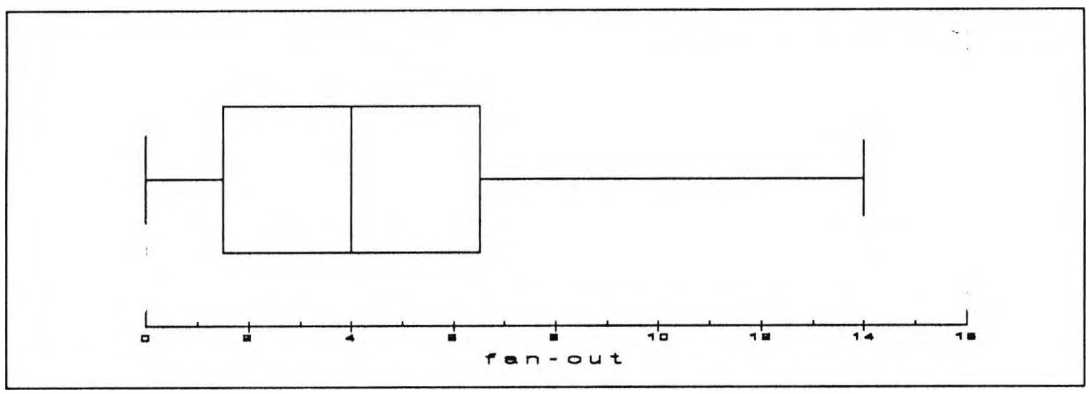
Appendix B: Boxplots for Dataset 2

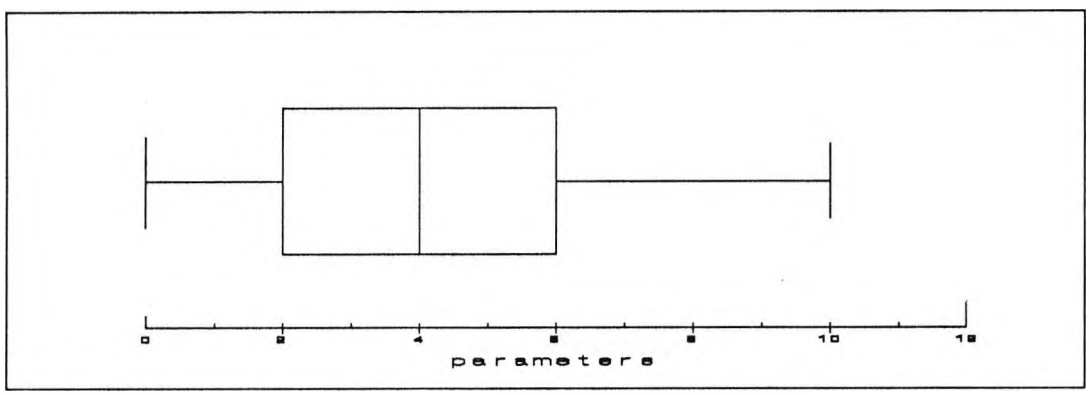

lines of code

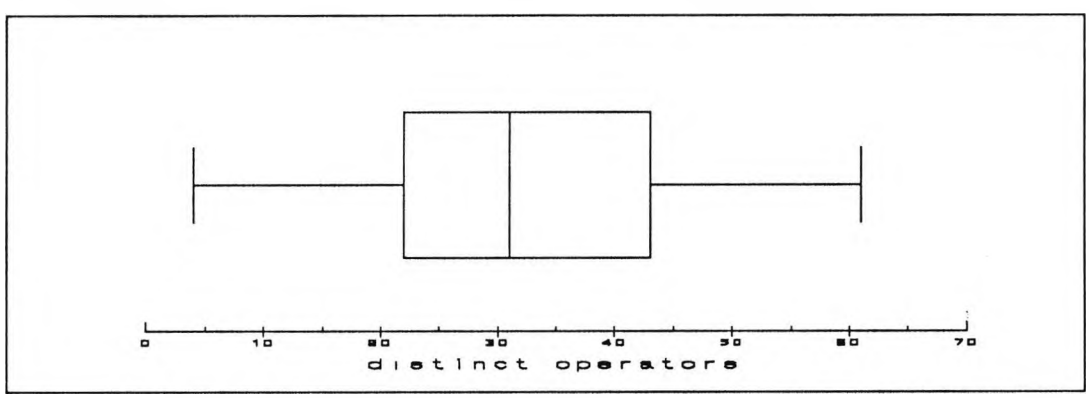

McCabe's V(G)

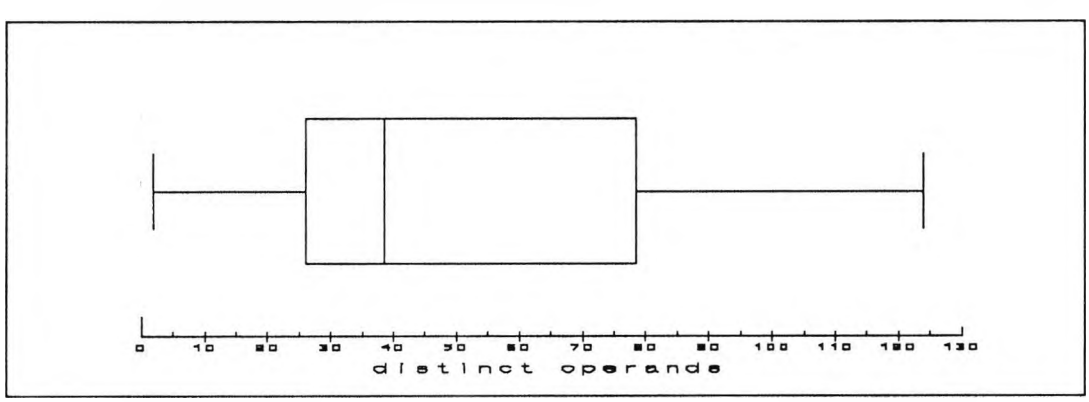

changes

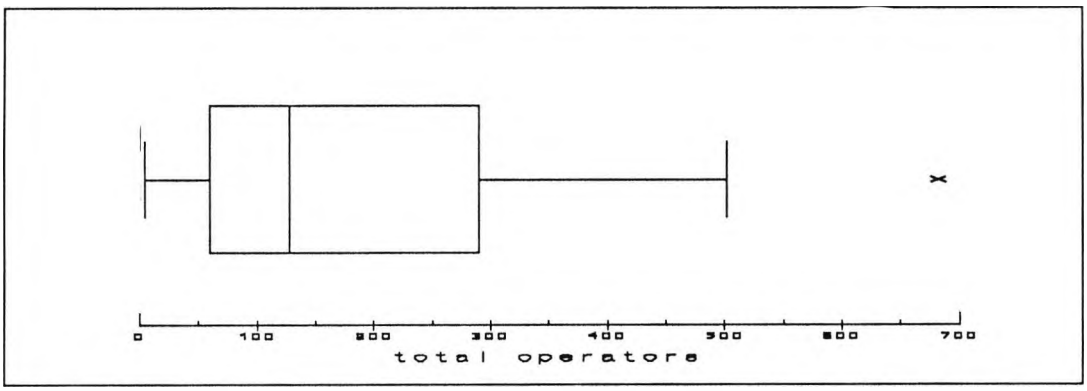

data items

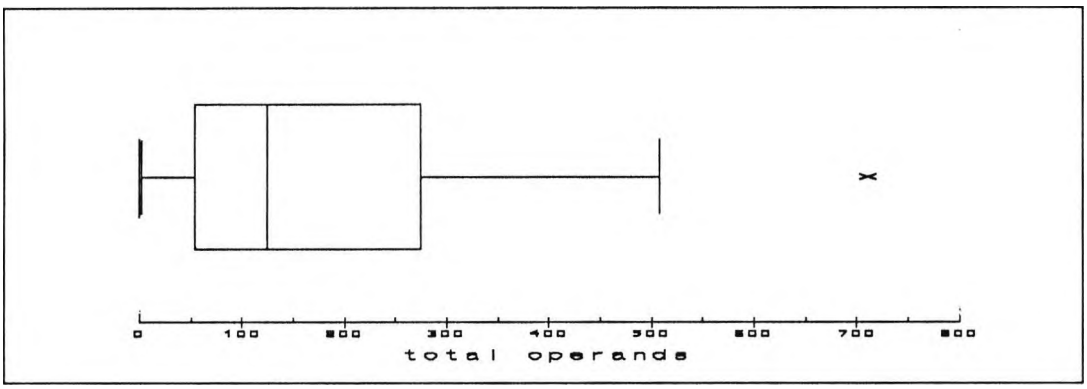

0
2
4
6
8
10
12
14
16
fan-out


0
2
4
6
8
10
12
parameters


0
10
20
30
40
50
60
70
distinct operators

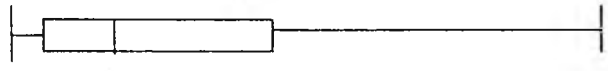

0
10
20
30
40
50
60
70
80
90
100
110
120
130
distinct operands


x
0
100
200
300
400
500
600
700
total operatore

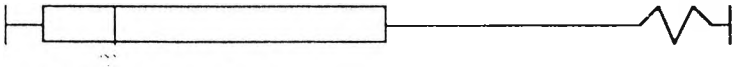

x
0
100
200
300
400
500
600
700
800
total operande

Appendix C: Boxplots for Dataset 4

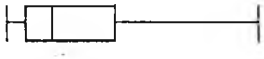
SCSI



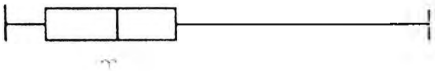
comments



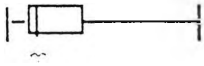
paths



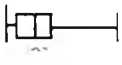
returns



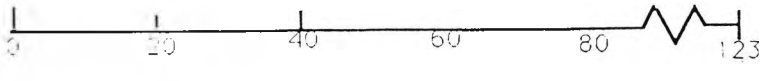
instruction boxes

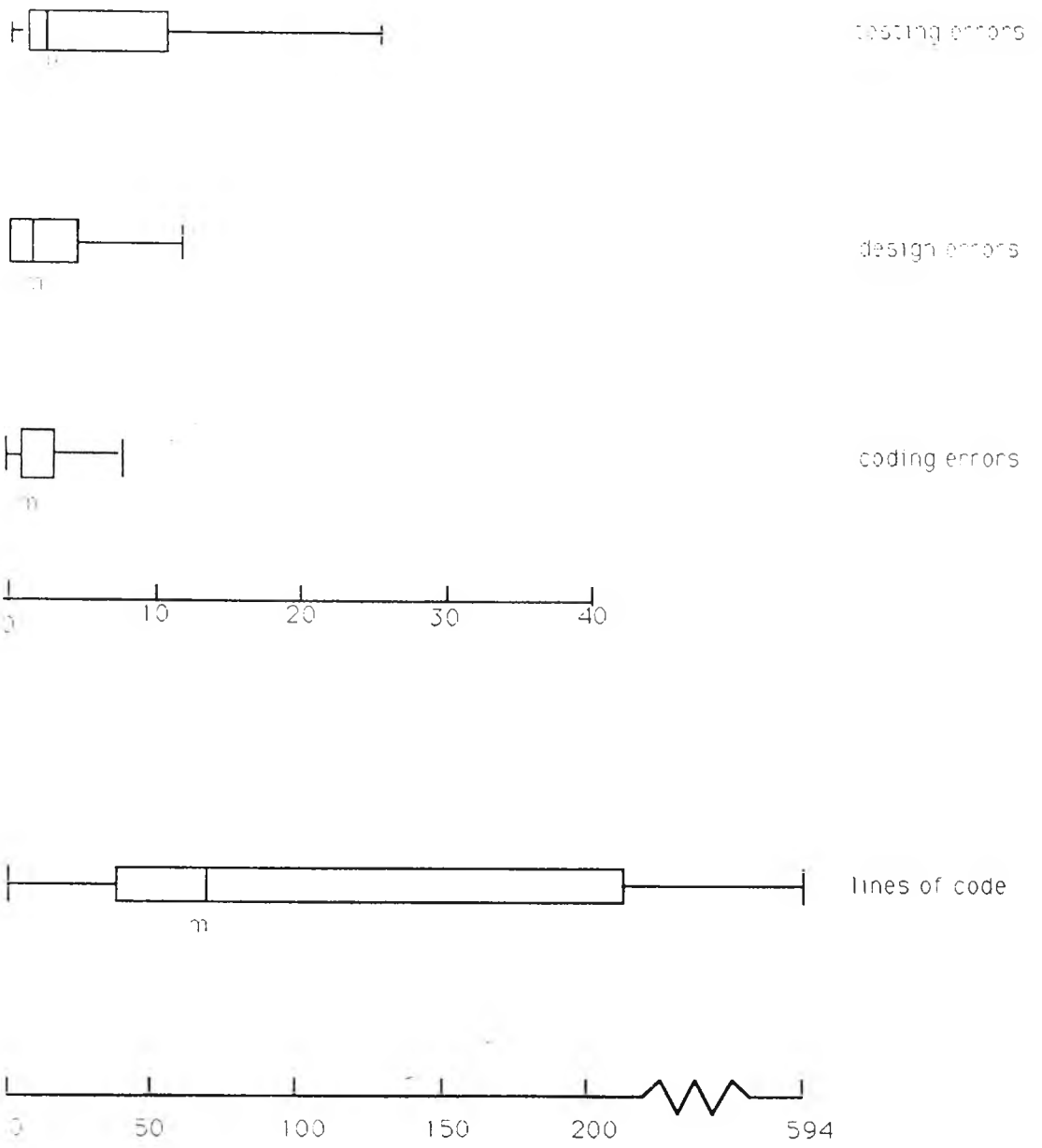


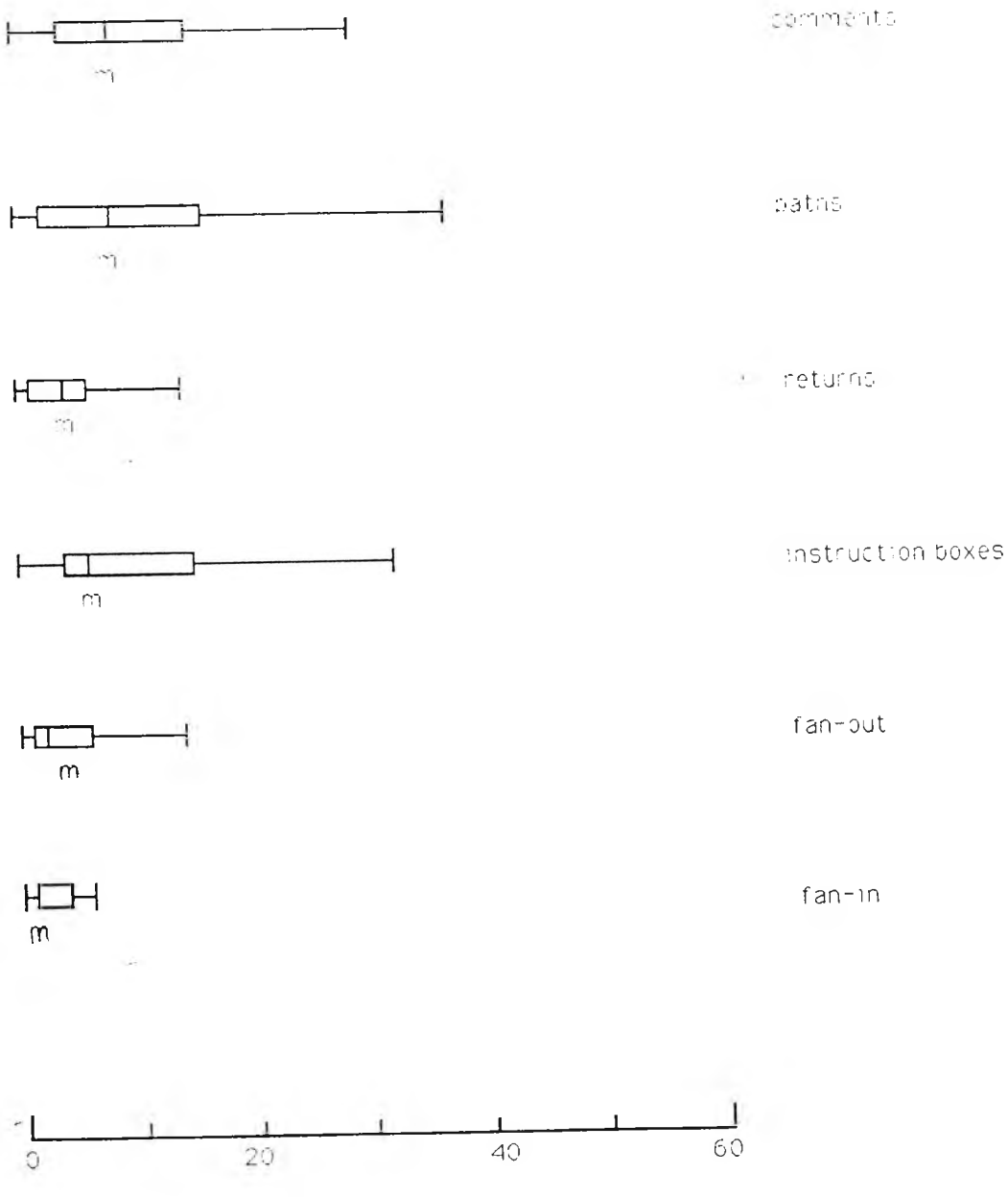
fan-in

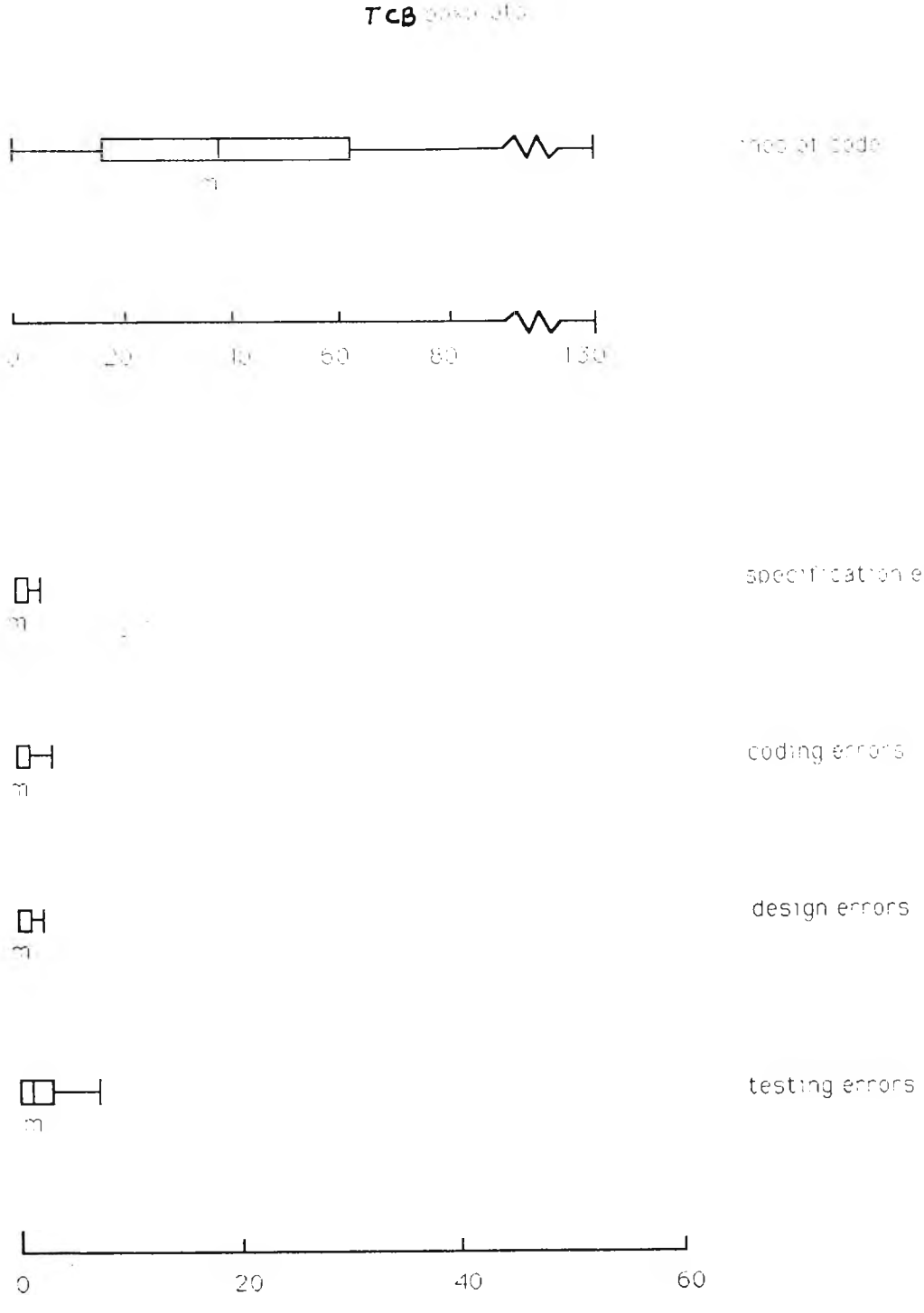


fan-out




testing errors
design errors
coding errors
0
10
20
30
40
lines of code
0
50
100
150
200
594


comments
m
paths
m
returns
m
instruction boxes
m
fan-out
m
fan-in
m
0
20
40
60


specification errors
coding errors
design errors
testing errors
0
20
40
60

Appendix D: Questionnaire

Automated Quality Control Experiment

Thank you for agreeing to take part in this experiment.

One method of quality control is to use software metrics to help detect when there is something unusual about an individual program. This can be done by studying the metric values and comparing the metric value for each program with all other programs. However, more information can be obtained by comparing two different metrics i.e. a scatter plot can be used to assess the relationship between two metrics. If the majority of the programs show a direct relationship (i.e. both metrics have either high values or low values for the same program) then the program that has one high metric value and one low one will be considered as unusual.

The aim of this experiment is to investigate what type of program, represented by points on a two dimensional scatter plot of various software metrics, are identified as unusual by experienced project managers. In some cases when the scatter plot could provide no information due to a large number of overlapping data points, a density plot has been used to represent the data instead.

The results will be used to identify a technique to automate the detection of unusual programs.

Background to the Experiment

This work forms part of an ESPRIT project, called REQUEST, which is led by STL at Newcastle-under-Lyme. The project is split into three sub-projects:

- development of a quality model
- development of reliability and ultra-high reliability/ fault-tolerant models
- provision of a database to support the other sub-projects

STL and ElektronikCentralen are primarily involved with the development of the quality model, although we have access to the results of the rest of the project.

The development of a Constructive Quality Model (COQUAMO) was inspired by the Constructive Cost Model developed by B. W. Boehm.The aim of the model is to give help to the quality and project managers in quality related issues. It has been recognised that the idea of one

predictive model, covering all phases of the development, is not feasible since strong predictive metrics are not available throughout the complete development. Therefore, it is now intended that the model will have three modes which relate to the three stages of a software development; planning, monitoring and assessment. EC has the responsibilty of the planning mode prototype and STL has the responsibilty of the monitoring mode prototype and the assessment mode prototype (the latter has mainly been sub-contracted to City University).

Planning mode

The planning mode will be used during the early stages of system planing and feasibilty assessment and is intended

- to help the user to identify and specify quality requirements.
- to predict final product qualities from the values of measures (quality drivers) available at the start of a project from plans and constraints.

This mode is the closest to COCOMO 's ideas

Monitoring Mode

The monitoring/steering mode is intended to assist the project manager to monitor and control the development, with quantitative measures that can be collected during development. This mode falls into two areas :

- A range of statistical techniques (from simple summary information to more sophisticated techniques), which will be used to identify unusual values.
- An advisory system, which will provide some possible interpretations of the cause of the unusual value.

This mode will be active throughout the development period from the requirements phase until the project has been completed.

Assessment mode

The assessment mode is invoked near the end of integration testing. It is intended to assist final product assessment in determining whether the product characteristics observed

conform to those initially specified during the planning mode. These results will be of interest in decisions on product release and in planning support for the operational phase.

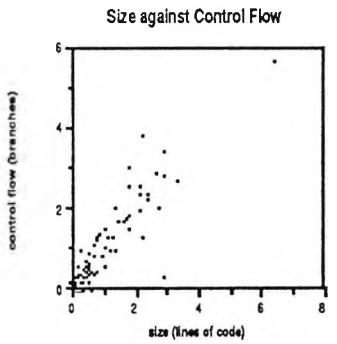
This experiment is concerned with the statistical techniques section in the monitoring mode. The results will be used to try and automate the procedure of identification of unusual metric values. The resulting information from an automatic detection can then be automatically fed into the advisory system which is concerned with the interpretation of the unusual values.

Procedure for Completing Questionnaire

The experiment consists of two sections:

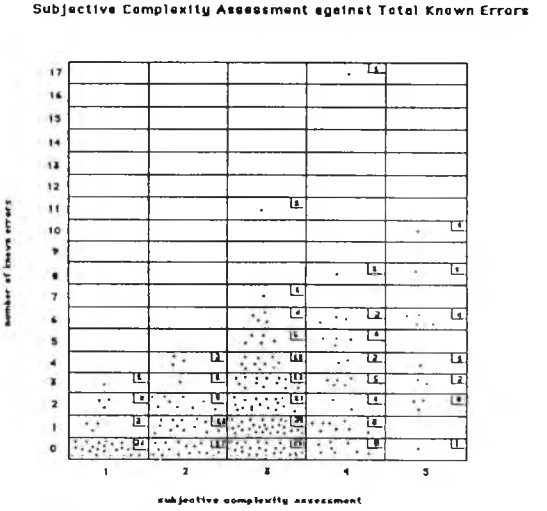
- Section one is a brief questionnaire which you should fill in at the start of the experiment to give information about your background and experience.

- Section two consists of bi-variate scatter plots and some density plots. The procedure is to simply circle the values you regard as unusual (if any), and worth further investigation, on each plot individually. Please add a minus sign beside the marked points which you regard as potentially problematic and a plus sign beside those marked which you regard as favourable (add nothing if you cannot determine which). If you feel that you require more information than that provided on an individual plot, please state the information you feel you require beside the relevant plot.



The density plot shows the number of points in each cell. Within each cell, the points are randomly scattered to show the density but they all have the same value and would overlap if plotted on a scatter plot. Apart from remembering that the position of any point

within a cell is random, and therefore no emphasis should be put on where the points lie, a density plot can be treated in a similar way as a scatter plot.



It is intended that the experiment should follow the normal practices of the manager, as far as possible. Therefore if another member of your team is normally responsible for quality control, this questionnaire should be completed by that person.

Each plot is intended to be treated individually. Do not attempt to compare them.

After completion of both sections please send them to the following address:

Lesley Pickard
STC Technology Ltd
Copthall House
Nelson Place
Newcastle-under-Lyme
ST5 1EZ

If you would like to receive a report on the results of this experiment, and/or further information about REQUEST, please attach your name and address to the Questionnaire when you return it. Please indicate if you would like to receive only the results of the experiment or information about REQUEST as well.

Description of Data Used

The data used for this experiment are measures, taken on a per program basis, from a sub-system of a large mainframe operating system. It consists of 226 programs. For each program nine measures were collected.

1. the number of programs which call a specified program (i.e. Fan-in)

2. the number of programs called by a specified program (i.e. Fan-out)

3. the number of total parameters on the program interface.

4. the number of data items used by the program.

5. program size in lines of code, i.e. non-comment, non-blank lines, in program

6. program control flow measured in terms of the number of branches.

   Notional branches were included so that IF-THEN-ELSE and IF-THEN-ELSE-IF were both counted as two branches. The number of branches for loops with a single control structure (i.e. FOR, WHILE, or UNTIL) was counted as two, and for loops with a dual control structure (i.e. FOR and WHILE, or FOR and UNTIL) was counted as three. The compiler evaluated compound booleans lazily, so each AND and OR in a conditional statement or loop control was counted separately.

7. program enhancements, i.e. the number of times the program was amended excluding changes for fault clearance.

   This information was obtained from formatted comments in each program which recorded each change to the program during its development and subsequent evolution.

8. the number of known errors, i.e. the number of faults corrected in the program.

   This information was obtained from formatted comments in each program, recording each fault cleared during its development and subsequent maintenance.

9. subjective complexity, i.e. an assessment of the complexity of the program on a scale of 1 (very simple) to 5 (very complex) provided by a member of the development group

Section 1 - Questionnaire

Reference Number _____

1. Number of years of computer experience?
   - a 0 - 2
   - b 2 - 4
   - c 4 - 6
   - d 6 - 10
   - e 10+

2. Experience domain?
   - a software
   - b hardware
   - c mixture of both

3. Formal Education

   (i)Level
   - a Post graduate
   - b Degree
   - c HND / HNC
   - d School qualifications
   - e No formal qualifications

   (ii)Subject
   - a computer science
   - b science / mathematics / engineering
   - c arts
   - d other

4. Type of job (predominately)? Reference Number _____

   a project manager
   b quality assurance manager
   c member of QA group
   d team leader
   e developer
   f other (please specify)

5. Are you a current or past manager?

   a current
   b past

   5(ii) If past - number of years since?

   a 0 - 2
   b 2 - 6
   c 6 - 10
   d 10+

   Please state current position _____

6. Do you usually use software metrics to detect problems?

   a usually
   b seldom
   c never

7. Are you familiar with the use of scatter plots for software quality assurance ?

   a no
   b partially
   c completely

8. Are you used to using metrics during development?

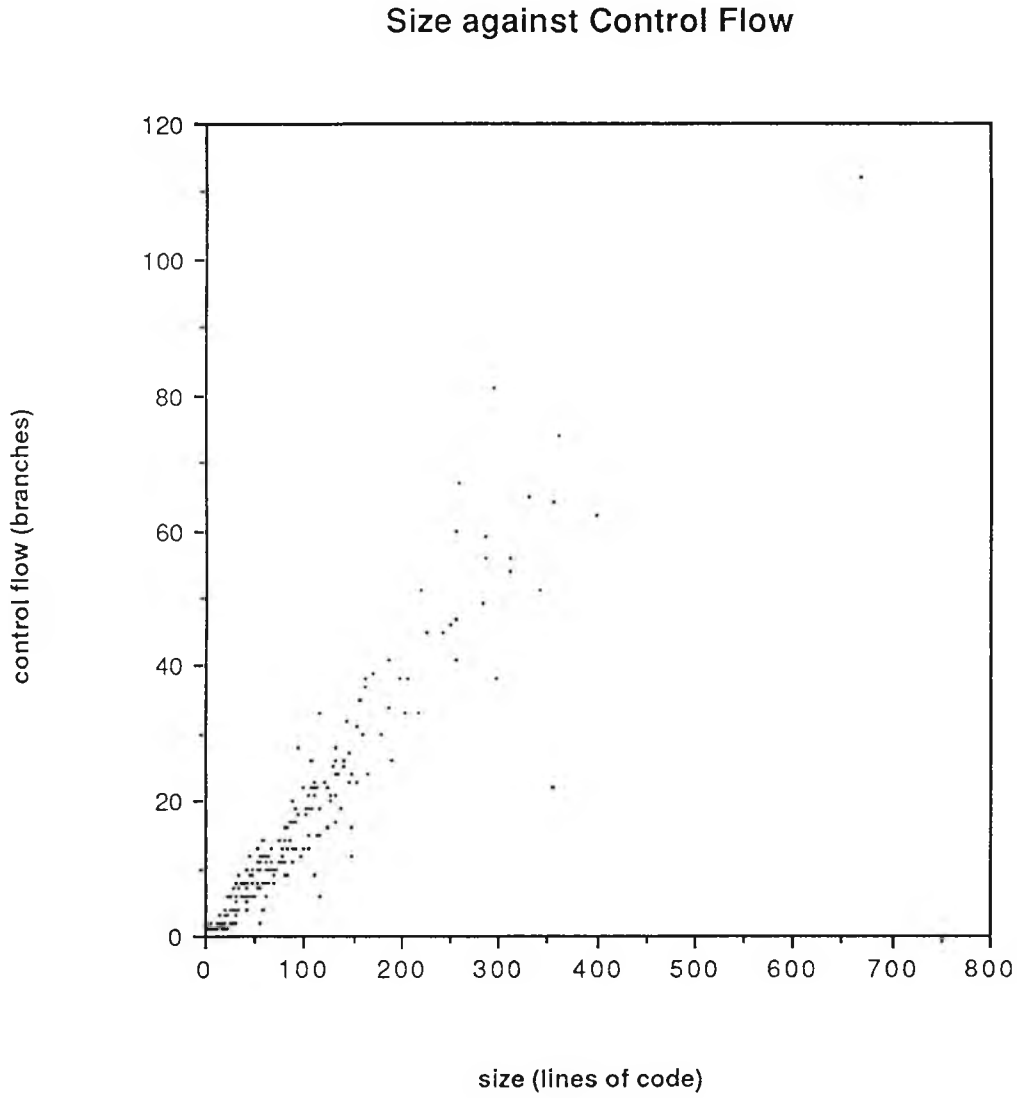
   a yes
   b no

9. Would you be willing to take part in a more detailed study concerning interpretation of software metrics?

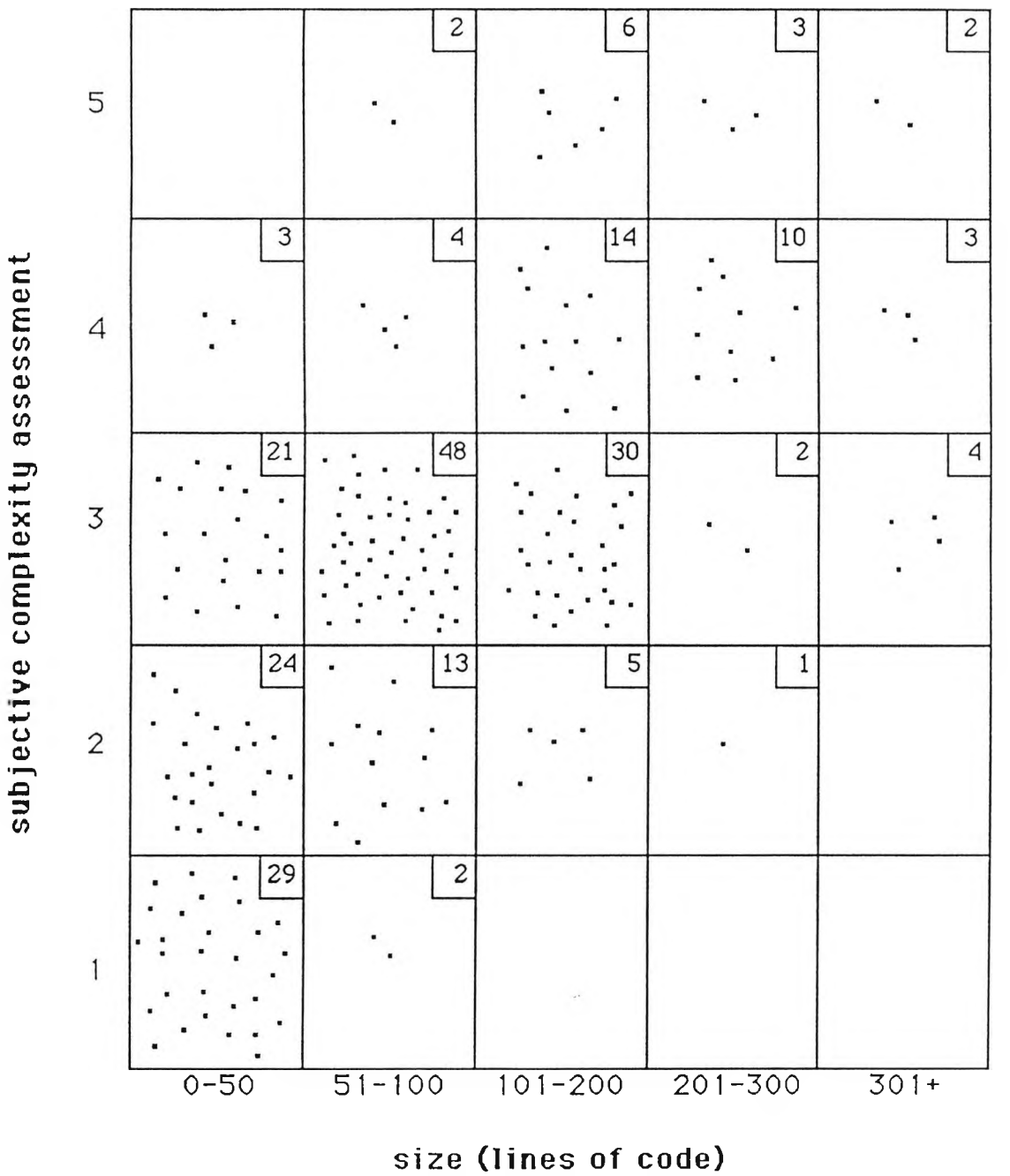
   a yes
   b no

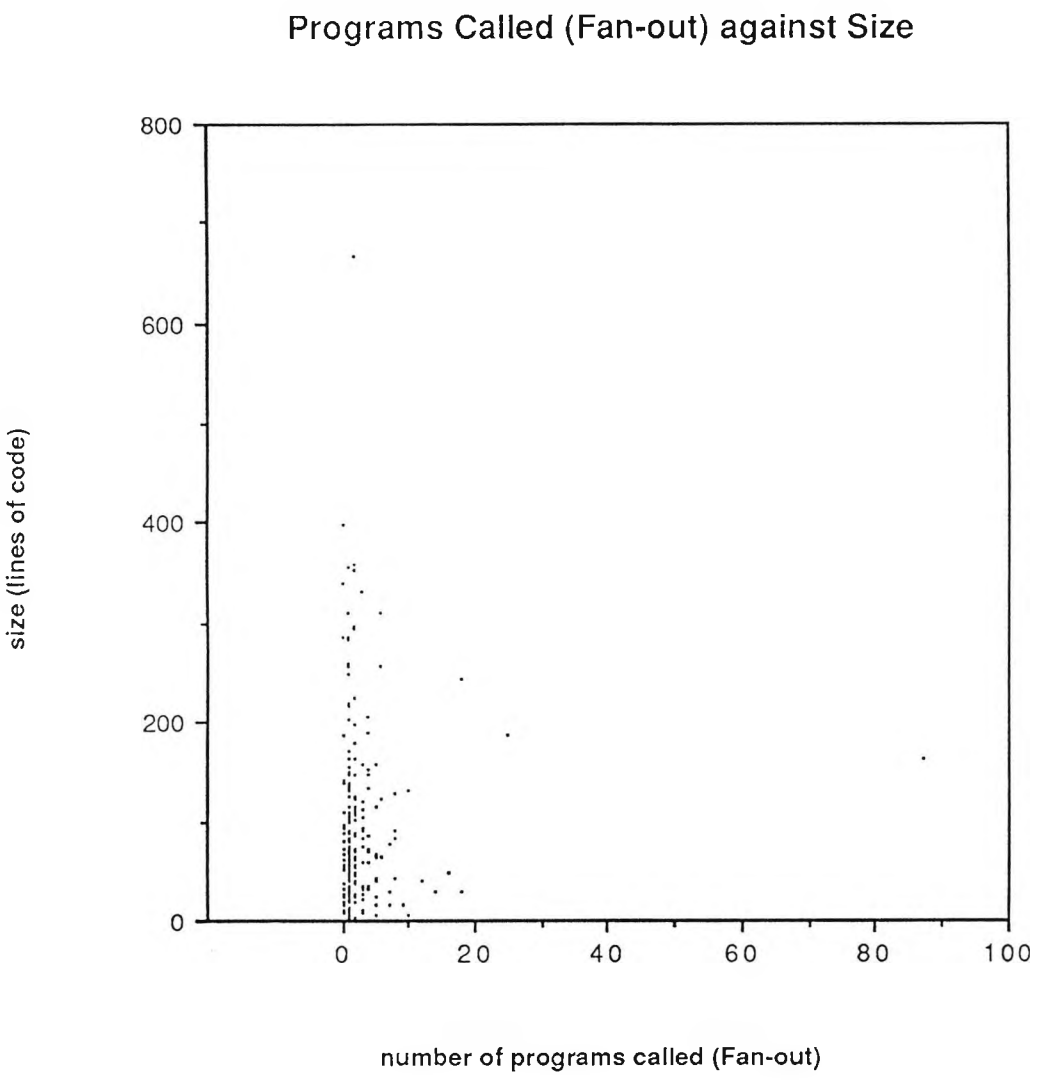
   If yes, please attach your name and address to the Questionnaire when you return it

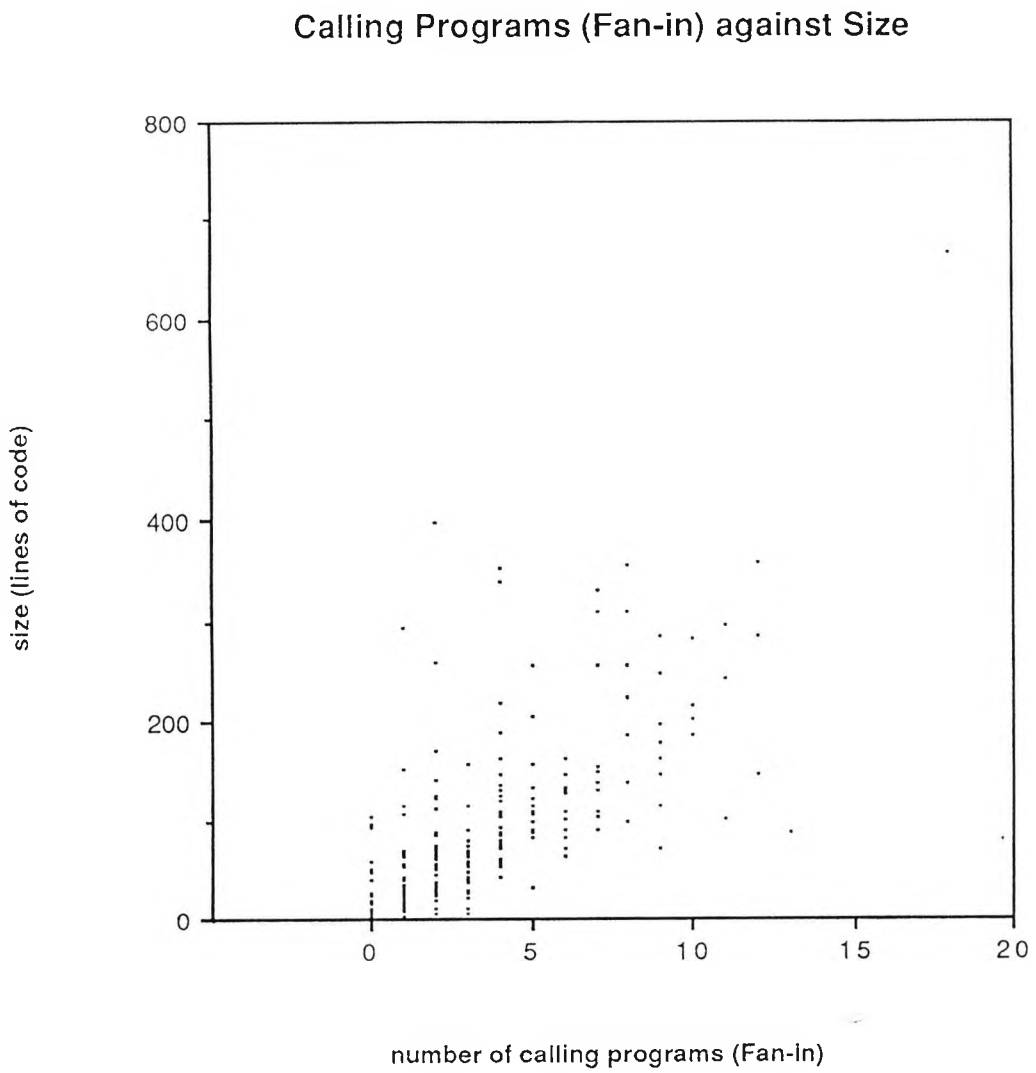
Contents List of Plots


Size against Control Flow
120
100
80
60
40
20
0
control flow (branches)
0
100
200
300
400
500
600
700
800
size (lines of code)

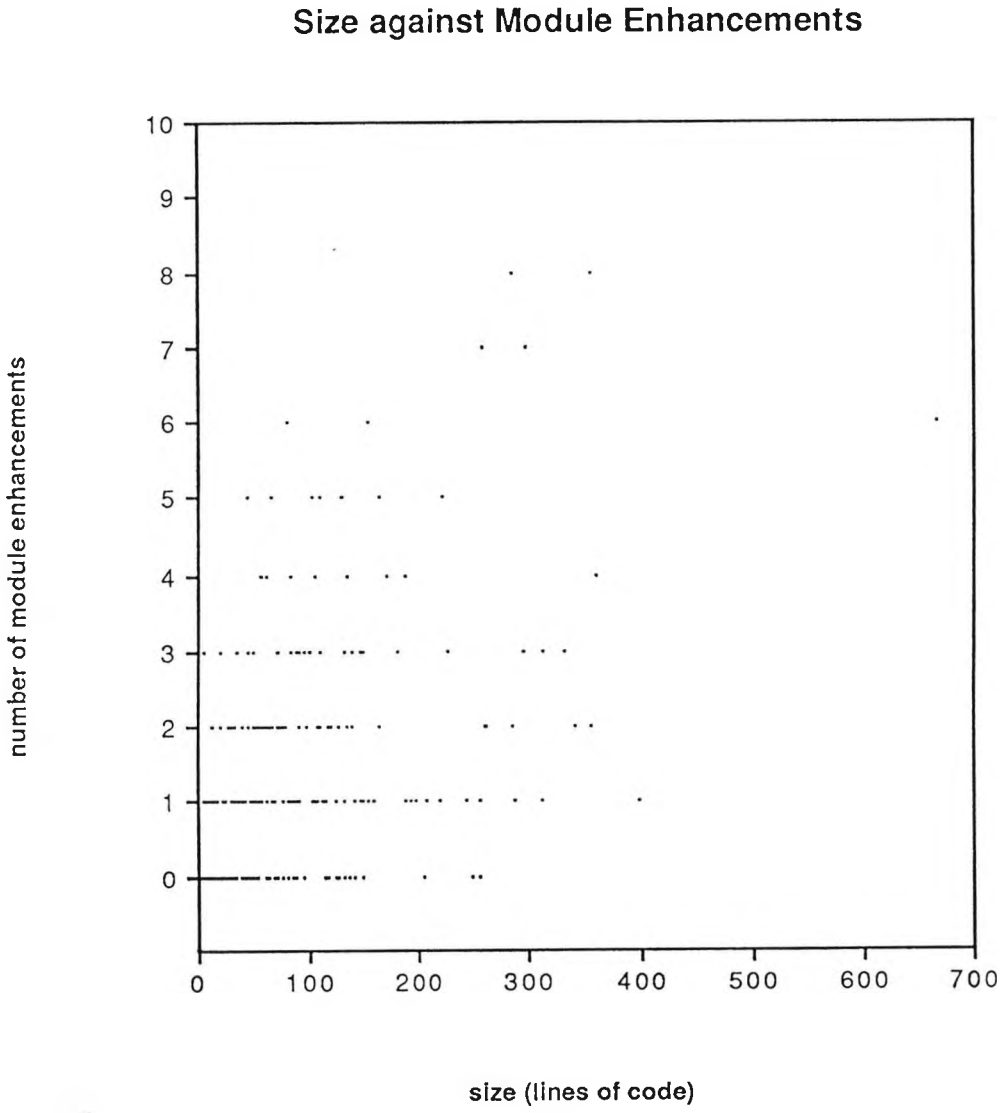
Size against Subjective Complexity Assessment

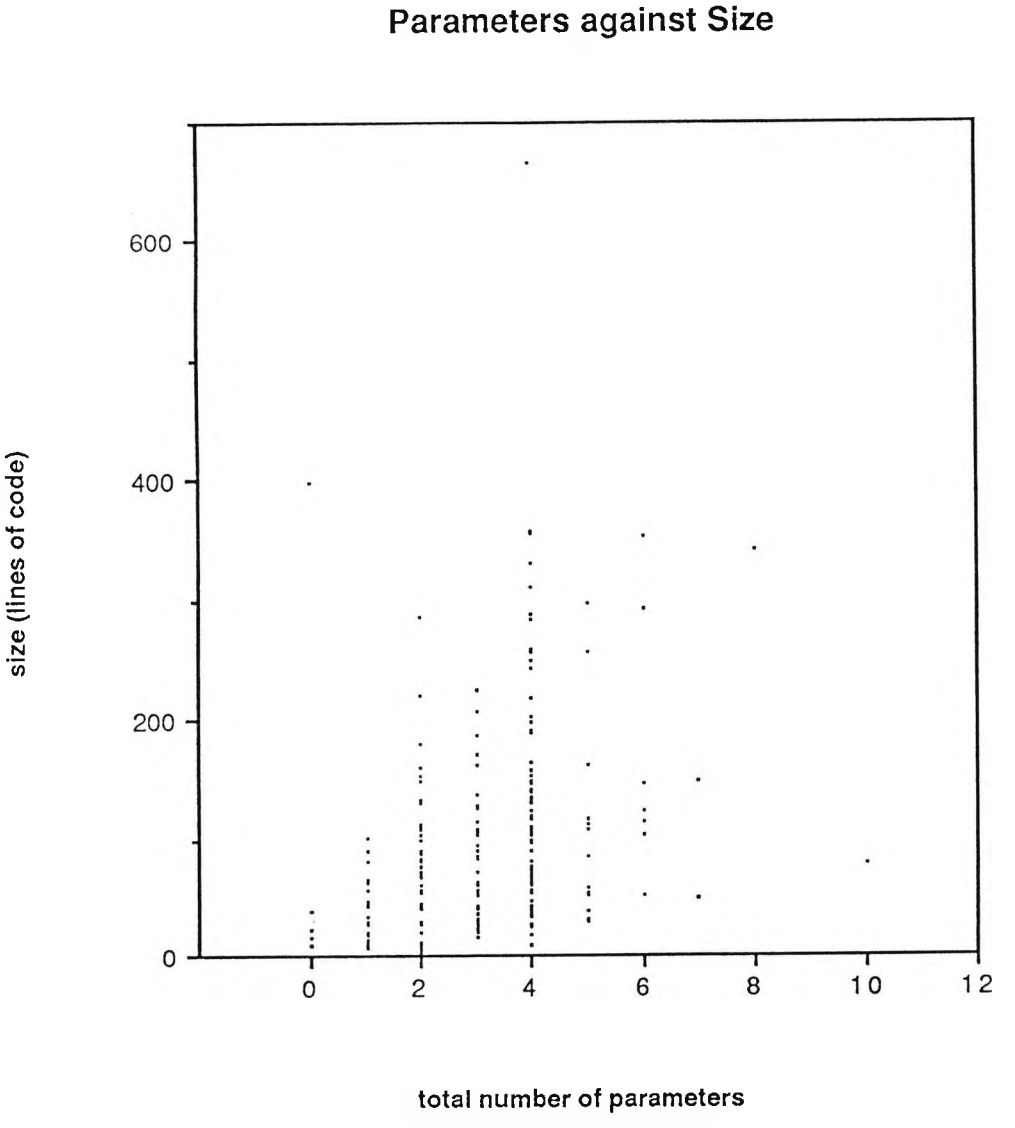


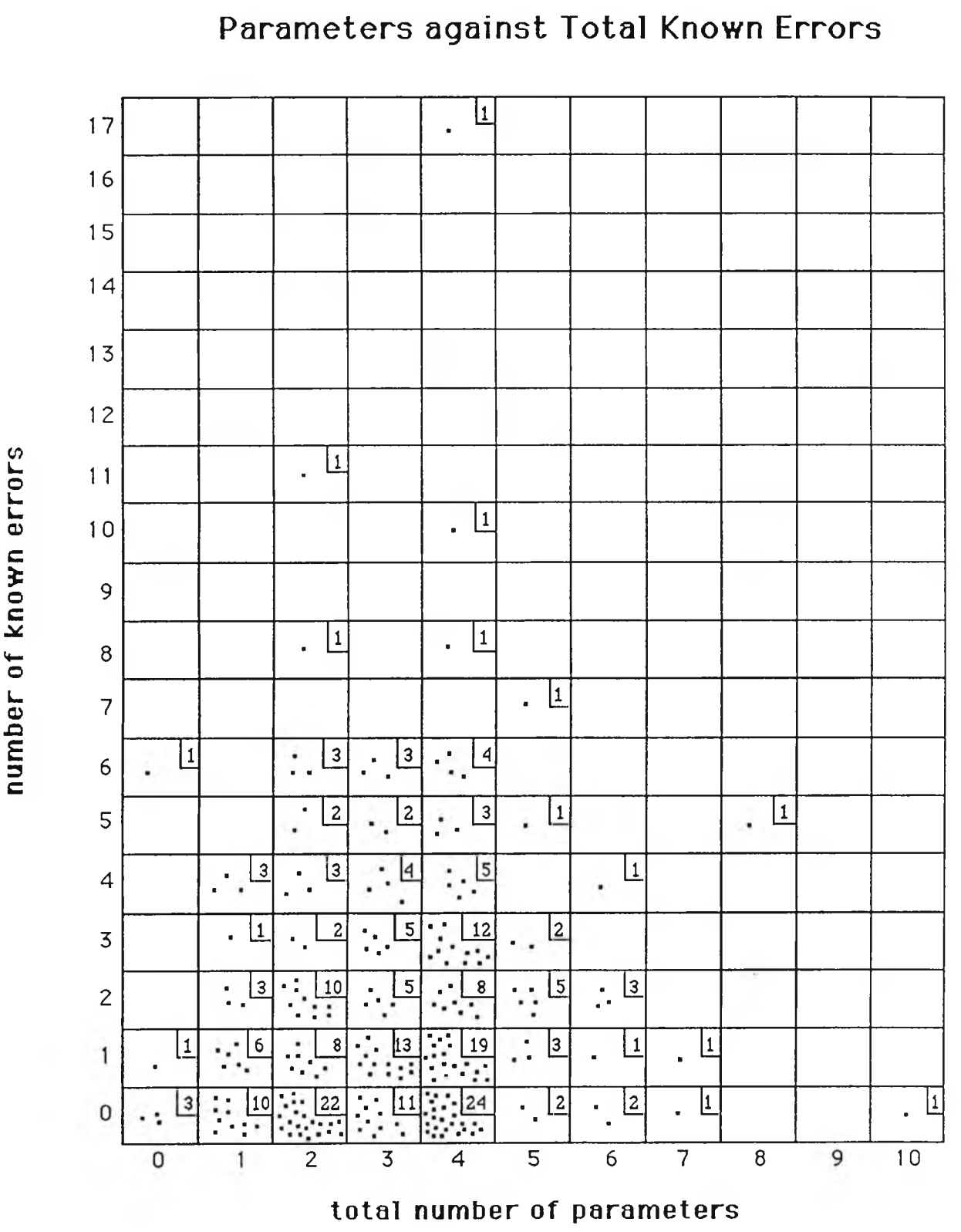

Programs Called (Fan-out) against Size
800
600
400
200
0
size (lines of code)
0
20
40
60
80
100
number of programs called (Fan-out)


Calling Programs (Fan-in) against Size
size (lines of code)
800
600
400
200
0
0
5
10
15
20
number of calling programs (Fan-in)

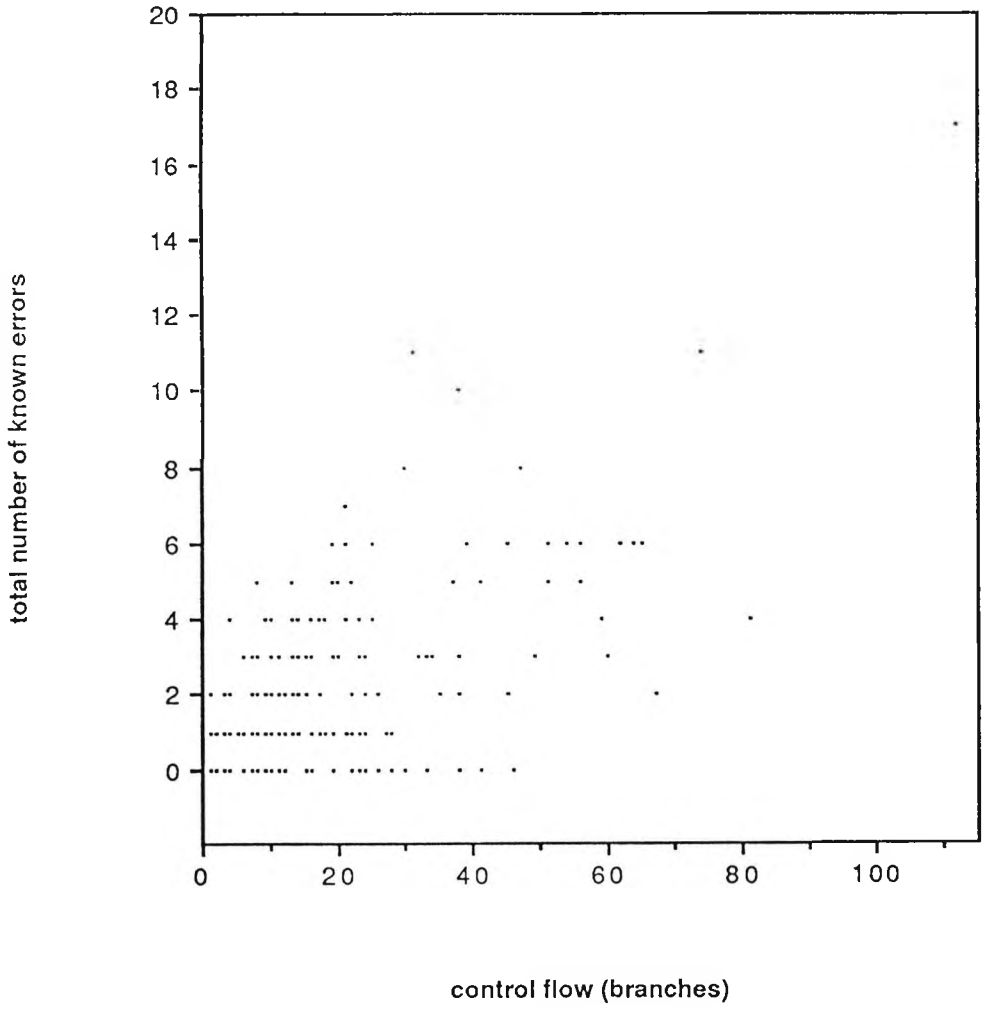

Size against Total Known Errors
number of known errors
20
18
16
14
12
10
8
6
4
2
0
0
200
400
600
800
size (lines of code)

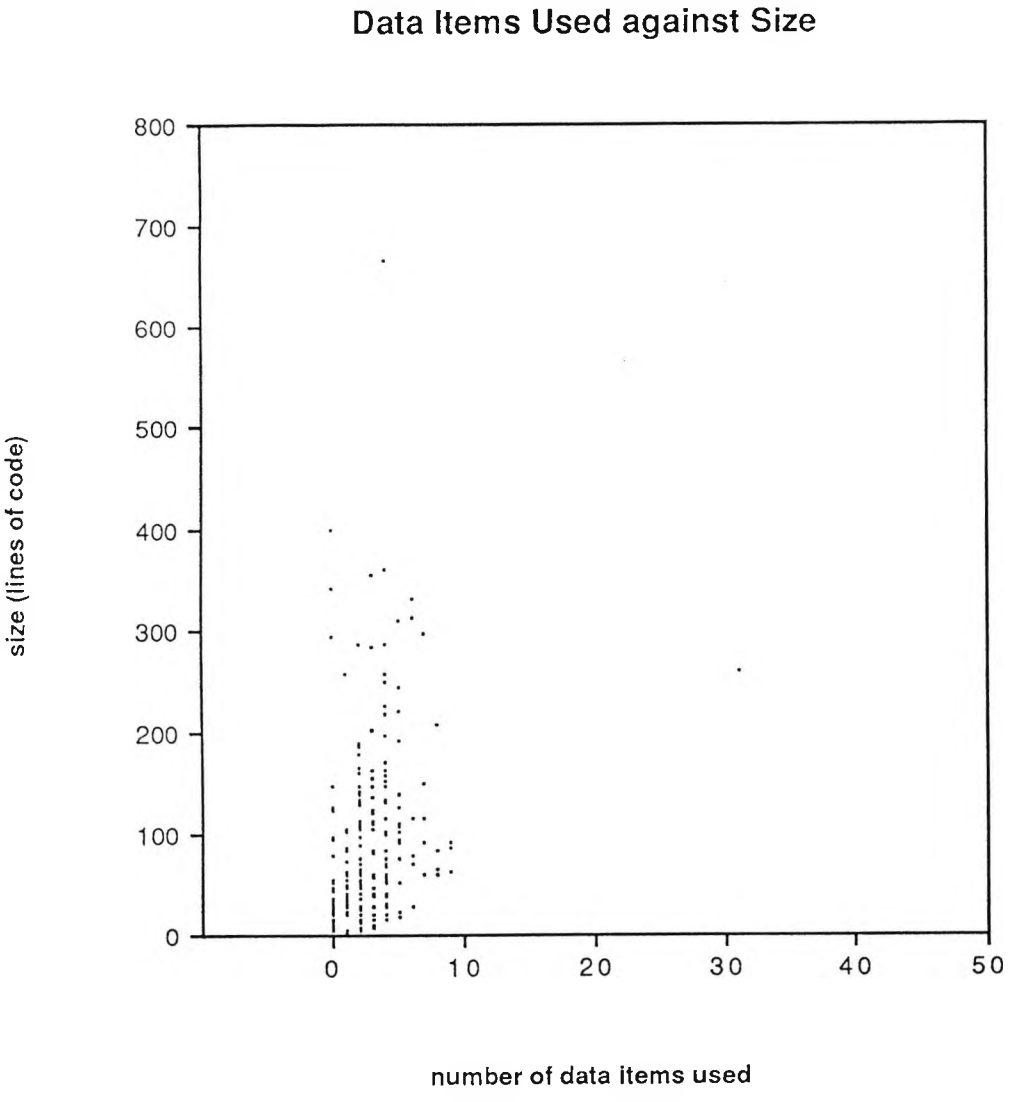

Size against Module Enhancements
number of module enhancements
10
9
8
7
6
5
4
3
2
1
0
0
100
200
300
400
500
600
700
size (lines of code)

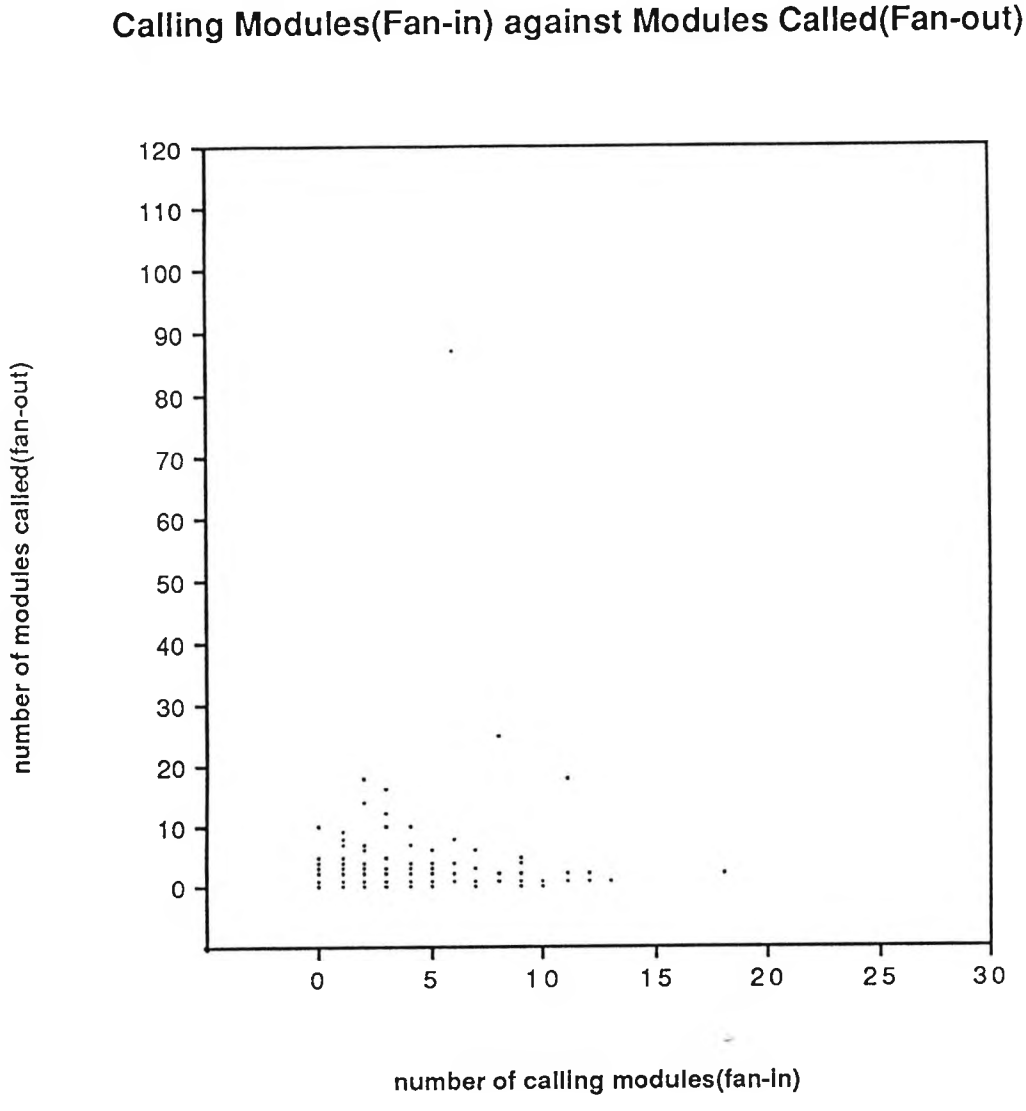

Parameters against Size
size (lines of code)
600
400
200
0
0
2
4
6
8
10
12
total number of parameters


Parameters against Total Known Errors
number of known errors
total number of parameters
0 1 2 3 4 5 6 7 8 9 10
17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

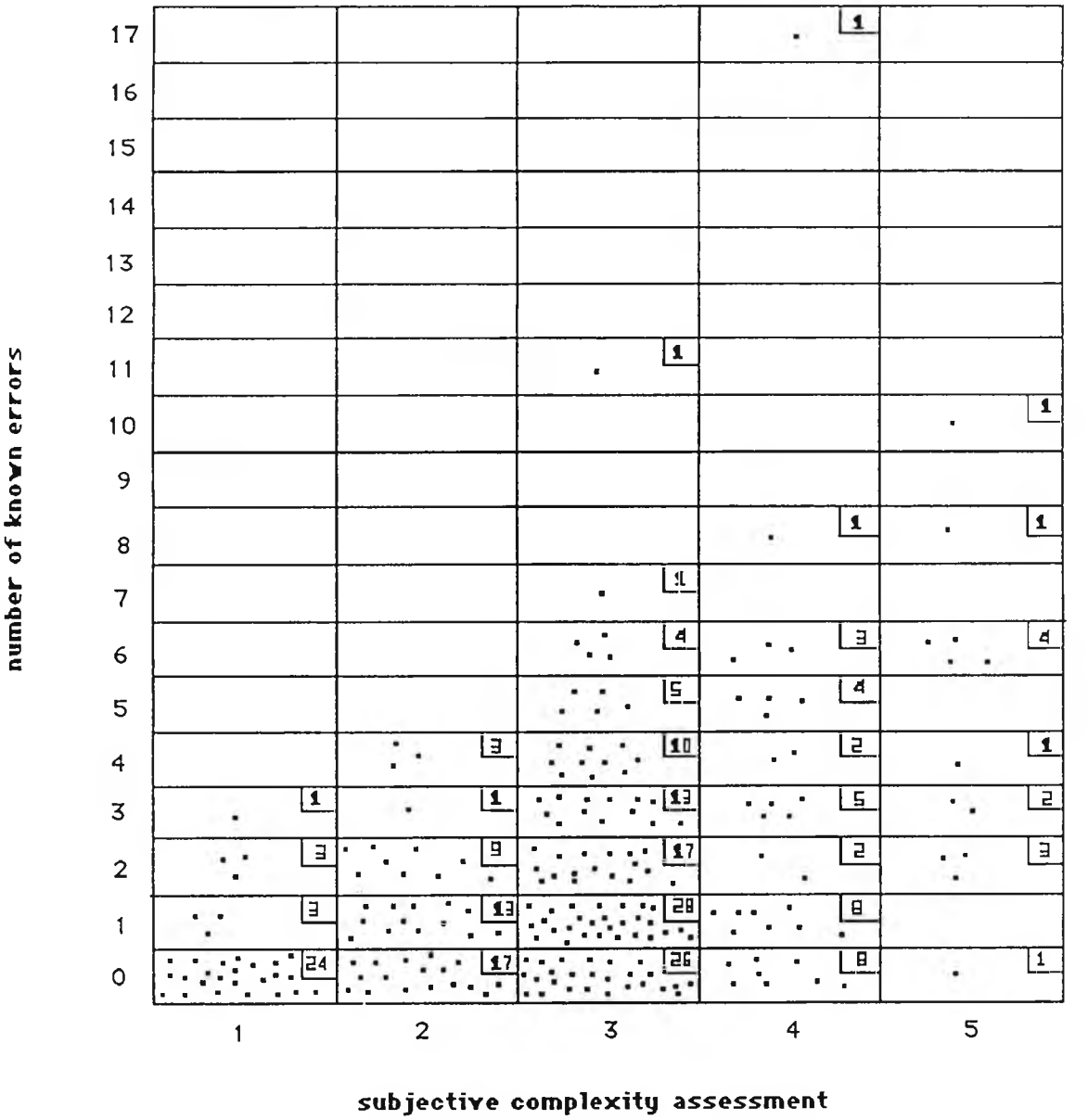
Control Flow against Total Known Errors

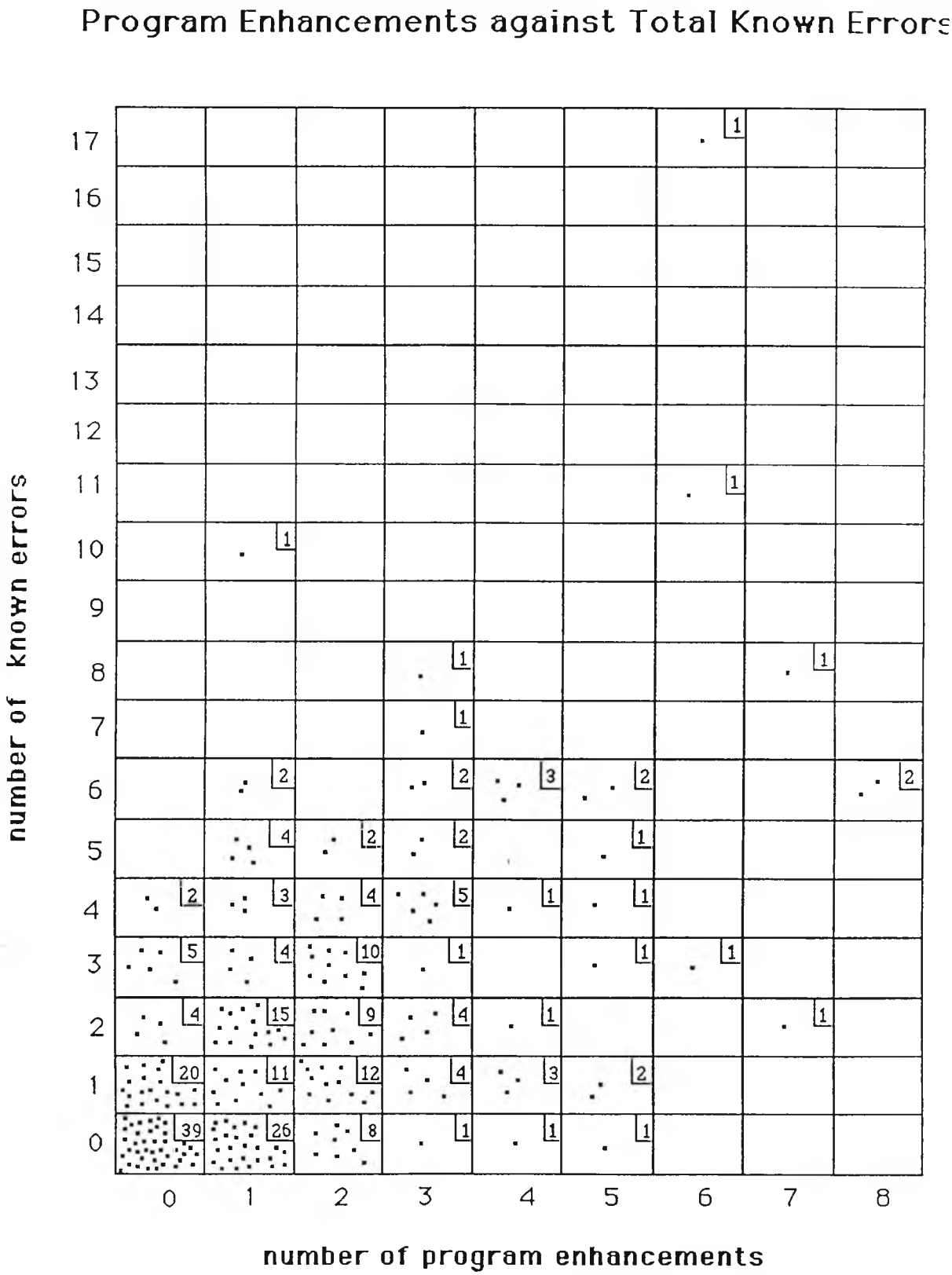



Data Items Used against Size
size (lines of code)
800
700
600
500
400
300
200
100
0
0
10
20
30
40
50
number of data items used


Calling Modules(Fan-in) against Modules Called(Fan-out)
number of modules called(fan-out)
120
110
100
90
80
70
60
50
40
30
20
10
0
0
5
10
15
20
25
30
number of calling modules(fan-in)

Subjective Complexity Assessment against Total Known Errors




Program Enhancements against Total Known Errors
number of known errors
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8
number of program enhancements