The Application of Software Metrics to the area of Formal Specification

by

Kathleen Margaret Finney

This thesis is submitted to the Centre for Software Reliability and the Higher Degrees Committee of City University in partial fulfilment of the requirements for the degree of Doctor of Philosophy.

1998

Table of Contents

List of Tables

List of Figures

Acknowledgements

I am very grateful to so many people who have supported, encouraged, and guided me in this work. My family and friends for unfailing support, my colleagues at work for the encouragement to start and finish a PhD and my supervisor, Professor Norman Fenton, for his patient, wise and reassuring guidance throughout the three years.

Library Declaration

Abstract

In this work our aim has been to investigate the application of software metrics to the area of formal specification. We first look at the nature of Formal Methods and those most commonly in use. This suggests the scope for the type of specifications that should come under investigation. Metrics and the nature of measurement become an important consideration as we discuss how to assess Formal Methods themselves and their impact on software development. We look at the work already done in software metrics to see if parallels can be drawn with the proposed work on formal specification. To establish current knowledge about the use and effectiveness of Formal Methods we investigate to see how their benefits had been assessed by two major surveys. Further information on the impact of the methods is obtained by analysing the results from a series of case studies. We build on work done in a previous study to investigate and compare attributes of three formal specification notations. This gives valuable insights into possible attributes, the way measurements might be made and possible metrics to establish. Drawing together the knowledge and experiences gained from the background research we design three experiments to look at metrics for comprehensibility in formal specifications. The experiments together with their results and statistical analysis are described. The main findings show that the variable names, comment levels and structure of Z specifications have an effect on their comprehensibility.

Dedication

I dedicate this work to my husband Roger who has always believed in me and to God who has sustained me.

Who hath measured the waters in the hollow of his hand, and meted out heaven with the span, and comprehended the dust of the earth in a measure, and weighed the mountains in scales, and the hills in a balance?

ISAIAH 40 12

CHAPTER ONE

1. INTRODUCTION

In this chapter we give the background and aims of the thesis, its main and subsidiary hypotheses and an overview of the succeeding chapters.

1.1 Background

The search for a software development method which would give assurance about the capture of users' requirements, accuracy in their realisation into code and all at a reduced cost, has been almost as long as the history of computing itself. The early proponents of Formal Methods were convinced that they had found part, if not all, of the solution to this search. However in the last 20 years these methods have not been widely adopted by the computing industry as a whole.

We believe that practitioners of software development have yet to be convinced of their value because so much of the 'evidence' about the benefits of their adoption is anecdotal or based on academic examples with little industrial relevance. To make the case for Formal Methods or to destroy the myths surrounding them we must produce hard evidence both from empirical work that can be replicated, and also from industrial sized case studies which have been conducted rigorously.

1.2 Hypotheses

The purpose of this work is to consider the application of metrics in the area of formal specification of software. Our aim is to examine a number of hypotheses concerned with the usefulness and effectiveness of Formal Methods. We will propose measures of attributes such as comprehensibility (of specifications), quality (of programs) and measures of the structuredness of the formal specifications themselves so that we can test the following hypotheses:

1. Formal Methods can be understood by any intelligent software developer with reasonable training,
2. the use of Formal Methods leads to improved software quality,
3. the structure of formal specifications impacts on their comprehensibility.

From a practical viewpoint we also propose the following hypotheses:

4. Formal Methods have not been used extensively in industry in realistic applications,

5. widespread take up of Formal Methods will occur only after the results from large scale case studies are published.

1.3 Aims

Formal Methods are an attempt to introduce mathematical rigour into the software development process. They are mostly used as specification techniques in the critical initial stage of the software lifecycle when the clients' requirements are stated and interpreted. They are used with the objective of helping to model the natural language statements of a specification using a mathematical form of notation, a formal specification, which may be expressed and validated using proof and logic. The final implementation in code, produced after a formal approach has been used in the initial stages, should reflect the thoroughness and precision of this approach. A discussion of the terms used and the background to Formal Methods is given at the beginning of Chapter 2.

We are aiming to quantify properties of formal specifications with the objective of considering their bearing on software quality. The criteria of interest for our research hypotheses are

- improved product quality as indicated by fewer faults or failures, improved reuse and easily understandable specifications,
- improved processes as indicated by the lower cost of overall software development and a greater predictability of both production effort and fault or failures.

It will be a consistent theme of this thesis that claims and counterclaims must be backed by scientifically valid studies and quantitative data. We argue that an attempt should be made to apply measurements to assess the benefits of using formal specifications with three objectives:

1. To measure the effectiveness of the methods themselves, for example whether using Formal Methods produces better quality software, lower re-engineering costs or reduced defect density. If a case is to be made for the use of formal specification techniques within the software engineering community then there is a need to quantify the effectiveness of their use. To provide evidence for this case measurements, both on the specification itself and the resulting software, are vital.
2. To measure directly attributes of the specifications themselves, for example size, structure, and modularity. The attributes of the specification need to be quantified so

that we will be able to move towards improving the quality of this stage of the software development. For example, it is strongly conjectured that the structure of a specification directly impacts on its comprehensibility. We will examine this particular hypothesis in depth. We should be able to make judgements about what constitutes a 'good' formal specification by measuring key parts of the documentation. Internal properties of the formal notation which have a bearing on its usefulness as the initial step in bridging the gap between client and code should be investigated to help these judgements.

3. To build predictive models based on these measures, for example development effort based on size and structure. We should be able to focus on those aspects of the specification which we can use to predict attributes of the resulting software. We need to establish if there is a direct link from some of the metrics taken from the formal specification and those obtained from the resulting software. We must try and establish the manner in which internal attributes measured from the requirements specification impact further down the software development lifecycle. In this way we are using the metrics obtained from the formal specification both for assessment and prediction.

1.4 Approach

The research approach used to address these issues is a combination of:

- novel analysis of independent studies where we look in detail at the basis of the study, the results presented and their interpretation to test hypotheses 2, 4 and 5,
- new controlled experiments to test hypotheses 1 and 3 concerning some of the fundamental qualities of a formal specification.

The benefits and limitations of formal specification are discussed in Chapter 2 along with a summary of several of the most common notations used.

To tackle all five hypotheses above we need to establish a rigorous framework of measurement and experimentation. In Chapter 3 we review those metrics which have been applied to software products and processes and look at how they might be related to measurements of formal specifications. In most cases we find that the advantages of sophisticated metrics over simple measurements like lines of code are not proven. We also see that the predictive power of some metrics is not well established.

To examine evidence of the extent and effect of the use of formal specification (hypotheses 2 and 4) we have analysed the findings of two major surveys into the use of formal methods in Chapter 4. In Chapter 5 we look in some detail at four large case studies where formal methods have been used in commercial settings and question some of the published conclusions about their impact. In particular we find the claims for one of the most celebrated 'validations' of Formal Methods to be based on dubious statistics and assumptions. Chapter 5 also includes an overview of the main case studies available in the public domain which address hypotheses 4 and 5.

Our consideration of the direct measurement of specification attributes has shown how little work has been done in this area. There are very few software metrics developed for use in the requirements specification stage of the development lifecycle. Some small examples are included in Chapter 6 along with our own comparative work looking at the same specification written under different formal notations. These latter studies were undertaken to investigate some of the internal attributes that might be measured in formal specifications (hypothesis 3).

The main experimental work to test hypothesis 1 is described in Chapters 7 and 8. These chapters focus on the attributes of specifications written in Z which we believe have a bearing on comprehensibility. Chapter 7 looks at the impact of natural language comments, helpful variable identifiers and structure on the comprehension of a specification. Here we found that the careful choice of identifiers for variables did have an impact on the comprehensibility and that the comment level helped to improve correct reading of the specifications. The aspect of structure (hypothesis 3) was further investigated in the experiment reported in Chapter 8 where the 'modularity' of a Z specification was altered to see what effect the size of the schemas had on several aspects of its interpretation. The best results from this experiment were obtained using a medium size schema whilst reducing the size down to less than 10 lines gave no measurable improvement.

The experiments show the effect of changing some of the attributes of several specifications under controlled conditions. As part of this experimental work other more general conclusions can be drawn about readability of this notation as perceived by its users. Full statistical analysis is given of the results of the three experiments conducted.

1.5 Conclusions and suggestions

Conclusions and suggestions for future work are given in Chapter 9. Broadly these concentrate on the 5 issues:

- the extent of the use of Formal Methods today,
- the parallels there are from work in software metrics,
- what can be learnt from industrial case studies,
- what we have shown by our experimental work,
- what further work can be done.

Appendix P contains papers that have been published so far as a result of this work.

The surveys reported here demonstrate a lack of evidence for or against the use of Formal Methods. In the research reported here we believe we demonstrate the value of empirical work and point to ways in which evidence may be obtained.

CHAPTER TWO

2. FORMAL METHODS

In this chapter we consider the nature of Formal Methods and give an overview of some of the most popular ones. We also consider the myths that surround them, some of the positive and negative aspects of their use and challenge some of the basic assumptions about them.

2.1 Background to Formal Methods

Formal methods arise from the search for software assurance. Problems in the manufacture of software include late delivery, expensive over-budget development and a product that does not fulfil the original requirement. As Rushby reminds us in comparison with the traditional engineering disciplines, based on the laws of science and the rigors of mathematics, software engineering seems more a craft activity based on trial and error rather than calculation and prediction [Rushby 1995].

Software systems are complex and in many systems, particularly safety critical ones, there is little margin for error. The relationship between the inputs and outputs of software is essentially a cumulative effect of many discrete decisions and we need to be concerned with correctness rather than limits of tolerance. To check properties of the software we need to apply modelling and proving techniques. Rushby states that the *formal* part of Formal Methods derives from *formal* logic where intuition and judgement are replaced by assumptions theorems and proofs written in a restricted language with very precise rules about what constitutes an acceptable statement or a valid proof. [Rushby 1993].

2.2 Definitions and origins

In journals and articles the terms formal methods, formal specification and formal notation are sometimes used interchangeably. Cohen argues that there are really only formal notations [Cohen 1989]. These are ways of expressing natural language statements using a mathematically based notation. Brinksma distinguishes between the formal notation and calculi used to describe and analyse models of a system and the methods which enable the user to take the model and obtain a working product [Brinksma 1992]. Many of the formal notations have had developed around them systematic ways of setting out their mathematical statements, together with rules defining the permissible set of manipulations possible within that notation. Some of these notation and rule systems are then collectively known as Formal Methods. Ehrig et al. give a historical perspective to the development of Formal Methods by connecting the rise of algebraic specification to the advent in the 1960's and 1970's of programming languages with higher levels of abstraction. [Ehrig et al.1992]. Bjørner et al. in their overview drawn from 20 years work in Formal Methods give the following definition which we shall adopt:

By a formal method we mean a method some of whose techniques and notations are formal, that is: can be mathematically expressed and mechanically supported [Bjørner et al. 1992].

Formal methods are distinguished from other design methods by the fact that the semantics are defined to such a high degree of rigour that it is possible to use a machine to verify proofs about conjectures relating to the specifications. In our studies the notation used for this mathematical expression is a very important factor with implications for comprehensibility, however we recognise that this may be a grey area when considering Formal Methods which allow an ASCII form of expression as an alternative.

Formal notations have their roots in mathematics and are an extension of the ideas of elementary algebra which can be used to express natural language statements. Given the statement

there are two quantities one of which is three times bigger than the other,

this can be modelled as an algebraic expression of the type

$$\mathbf{x = 3y}$$

For a fuller discussion of related notions of modelling see [Fitzgerald and Larson 1998]

We start to use letters for quantities and develop semantics so that the symbol = is used for "is equal to" and 3y implies a quantity 3 times bigger than y.

With natural language statements such as

all the nice girls love a sailor, (1) problems of ambiguity, vagueness, incompleteness and contradiction become apparent. Some of these problems are highlighted by the questions:

- do all the girls love the same unique sailor? (ambiguity)
- do the girls have a distinct single sailor each? (ambiguity)
- if a girl does not love a sailor is she not nice? (incompleteness)
- can a girl who is not nice still love a sailor? (incompleteness)
- is the sailor the sole object of their love or can others be included as well? (ambiguity)
- what is meant by nice? (vagueness)
- what is meant by love? (vagueness)

If statement (1) is coupled with

Jane is a lovely girl but dislikes sailors,

then we may even have a contradiction.

It is claimed that formal specifications, as part of the software engineer's range of techniques, should be incorporated from the start of the development lifecycle to try and capture the requirements specification precisely and overcome some of these problems of natural language. It is proposed that formalism be used in the initial stages of the specification to form a mathematically based bridge between the requirements documentation and the design and implementation of the software. As with any good specification, the formal specification describes what the functional requirements of the system are at an abstract level but not how they are to be implemented in code. Recently the border between functional requirement and design has become less clear and some Formal Methods are seen as an aid to design. Hayes and Jones argue strongly that specifications should be kept at an abstract level and should not be restricted by specifiers aiming for executable constructs [Hayes and Jones 1989], although an alternative viewpoint was presented by Fuchs [Fuchs 1992].

Returning to the phrase (1) above, by 'translating', it that is re-expressing it in a formal notation, with suitable sets, functions and logic, the mathematical basis of the notation should provide clear answers to all questions about the statement or at least highlight areas where further information is needed to clarify the true meaning. Unfortunately the original easily read natural language phrase could now well resemble an expression of the type

$$\forall x \exists s ((x \in G \wedge x \in N) \Rightarrow (s \in S \wedge (x,s) \in L)) \quad (2)$$

2.3 The problems of understanding formal specifications.

Comparing versions (1) and (2) of the information we can see that in order to try and express our statement clearly and unambiguously in the language of first order logic as in equation (2), we have lost the 'user friendliness' of natural language. Moreover, even the formal notation does not avoid the more serious problems inherent in any specification. In particular we may still have the problem of vagueness if we cannot interpret the sets N and L clearly. Validation by the customer becomes difficult once their requirements are translated in this mathematical way and the notation becomes a barrier to the reading of the specification.

Large sections of the population, particularly in the U.K, have difficulty with concepts of basic school algebra where numbers in concrete situations are replaced by abstract symbols and equations [Burghes 1992, Roy 1992]. This has implications for the level of comprehension which can be expected from clients, designers and programmers when presented with a formal method which can contain mathematical symbols, expressions of logic and a notation which is specialised to that particular method. We need to understand the properties of formal specifications better and measure some of the key attributes in order to understand whether these have had an impact on the adoption of Formal Methods. Traditionally most Formal Methods exponents are among the small minority of the population who have a good advanced mathematics background and training; they may therefore fail to appreciate the difficulties a lot of people have becoming at ease with mathematical notations. Later we shall see this theme developed in Chapters 6-8 as we look at the comprehensibility of formal specifications in more detail.

So far reaction to raising these difficulties (after initial disbelief that anyone could find the mathematics a barrier) has been to take the view that only a few specialised groups within a company would therefore be trained and familiar with these Formal Methods [Rushby 1995]. This does conflict with the philosophy of aiming for widespread use of formal specification as a solution to problems of software quality. We return to this issue in Chapter 4.

2.4 The procedure from specification to code

A formal notation should support abstraction in the sense that it should not favour any particular design or method of implementation. A lot of decisions are postponed until later in the development so that the most appropriate representation of the data, functions and operations can be chosen by the design and/or programming team.

Hall [Hall 1990] defines the main activities where Formal Methods can be used as:

- writing formal specifications,
- proving properties about a specification (e.g. error conditions and invariants),
- constructing a program by mathematically manipulating the specification (refinement),
- verifying that a program satisfies its specification.

Hayes and Jones suggest that a specification written in a formal notation should not be compiled and executed like a program and Wordsworth states that

> *Formal methods are not methods for developing software without facing up to difficult decisions, but methods for recording those decisions once made* [Wordsworth 1992].

Some methods have set procedures laid out for each of the stages from specification to code with a mathematical basis for refining the specification repeatedly until coding is finished and these could therefore be described as 'mature' methods. Other notations do not have such well-documented procedures or have not been developed to a similar level and often are only employed in the first of the specification stages to capture the requirements from the client. In many projects the use of formal notations has been confined to these early stages where their main benefit has been as a means of clarifying requirements and design decisions. No attempt has then been made to continue with the rigorous demands of refinement.

For those formal notations that are part of a mature Formal Method, the notation supports a development method which leads from the specification towards an implementation through all the lifecycle stages to the final production of the software itself. Woodcock gives the mathematical basis and definition of algorithmic refinement, the transformation from specification to code [Woodcock 1992]. The process of changing from a formal specification to an executable program should incorporate the stages of verification, refinement and translation.

At the verification stage the design of the formal specification is checked against the clients' requirements and is shown to satisfy them. During refinement details can be added to bring the specification nearer to a design for coding and decisions about implementation details like data structures and storage can now be incorporated. Each time a change is made the Formal Method usually has a requirement that it should be possible to prove that these additions have not introduced errors and that this new, less abstract form of the specification still satisfies the original requirements. (We note that what is referred to as verification here is often referred to as validation elsewhere)

Wordsworth defines a refinement, Ref, of a specification, Spec, by the two relations given below [Wordsworth 1992]. These involve the pre-conditions and post-conditions of the specification and its refinement. Pre-conditions are predicates that must be satisfied by the inputs to a specification whereas post-conditions are predicates which are concerned with the final state as well.

1. pre Spec $\vdash$ pre Ref

2. (pre Spec) $\wedge$ Ref $\vdash$ Spec

1. is the safety condition where the preconditions of the original specification are weakened in the refinement.

2. is the liveness condition where the post conditions are strengthened.

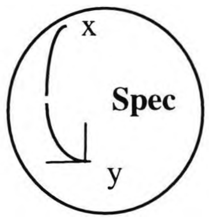
By weakness what is meant is that we allow a wider choice of inputs or less stringent conditions applying to the input state. In contrast the conditions on output are strengthened and become more rigorous and restrictive.

As an example

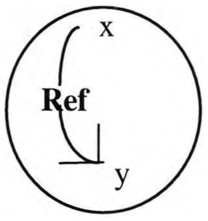
Spec is defined on the set R^+ (positive real numbers) with the operation OP_S to output the square root of an input number.

Ref is defined on the set R (all real numbers) with the operation OP_R to output the square root of the absolute value of an input number .

OP_S



OP_R



$y = \sqrt{x}$

$y = \sqrt{|x|}$

i.e. $x \geq 0 \quad y = \sqrt{x} \qquad$ or $x < 0 \quad y = \sqrt{-x}$

Refinement may be done in a series of stepwise operations rather than a single transformation. What is produced after the final refinement is known as a concrete specification.

The last step of translation moves from the concrete specification to code, this step is also sometimes called implementation.

In practise we have seen that despite valiant efforts on the part of academics and others there are very few examples where any attempt has been made to use Formal Methods for the complete procedure from requirements to code and no real evidence from case studies for their use across the whole lifecycle. Most published applications of Formal Methods show that even when used with tool support, they have been restricted to the specification part of the software lifecycle with little refinement or proof.

2.5 Classification of formal methods

The classification of Formal Methods has parallels with that of programming languages. Both can be based on the style, generation or on how widespread their use has been. Formal Methods have been classified into types in a number of ways; Barroca and McDermid divide them by style into 5 categories [Barroca and McDermid 1992].

Following from Barroca and McDermid's classification we can divide Formal Methods by type.

- Model based methods which give explicit definitions of the state (system) and operations which transform the state. No attempt is made to deal with concurrency. Examples are Z [Spivey 1992a], B Method [Neilson and Sørensen 1994,Abrial 1996] and VDM [Jones 1990].

- Process algebras which use explicit models of concurrent processes and represent behaviour of the system by the constraints on allowable communication between processes. Examples are CSP [Hoare 1985] and CCS [Milner 1989].
- Algebraic approaches which use implicit definitions of operations by relating the behaviour of different operations without defining the state, again with no concurrency. Examples are OBJ [Gougen and Tardo 1979] and general ADT [Martin 1986].
- Logic based approaches, these are a wide classification which covers approaches such as use of temporal and interval logics; useful where the timing behaviour of a system needs to be captured [Galton 1987][Kroger 1987].
- Net based approaches which concentrate on the data flow through the system and include conditions under which the data can pass from one node to the next. Examples are Petri nets [Peterson 1977] and Predicate Transition nets [Voss 1980].

In addition to the McDermid classification, there are some methods which are 'hybrid'. For these there has been an explicit attempt to merge two or more different approaches. In this class we have examples like Raise [Havelund et al 1992].

Within categories there can also be a classification by scale and maturity. Some methods have been adopted for large-scale industrial use, for example Z, others are small individually designed examples that have only ever been used in a single textbook or as an academic paper for a 'toy' application. There are a large number of this latter class as researchers have designed their own specification notation to suit a particular problem domain.

2.6 Overview of the main notations

In order for readers to understand the basis for the empirical studies in this thesis it is necessary to provide an overview of the main methods that arise in the case studies and experiments. We have concentrated on methods from the first three types as these are more commonly cited in the surveys of use of formal methods (see Chapter 4).

2.6.1 VDM (model based)

VDM was developed in IBM's Vienna Laboratory and grew from its predecessor VDL, the Vienna Definition Language [Wegner 1972]. VDL was the semantic definition of the programming language PL/I constructed by the 8 strong team under Heinz Zemenack at the laboratory in the 1960's and it was not conceived as a general notation. However out of this experience the more general Vienna development method, VDM, was born. The notation system used within it to express definitions and operations was referred to as Meta- IV, but it was only used in the IBM laboratory in Vienna from 1973-1978. Its use began to spread as VDM was applied to the formal definitions of programming languages used in the Telecommunications industry and people saw it being used to validate Ada compilers. By the mid 1980's VDM Europe was established and a variety of VDM dialects were in use. A definitive textbook was produced outlining the notation and methods used by VDM [Jones 1990]. The BSI and ISO developed a standard using as a basis the books of Bjørner and Jones [Bjørner and Jones 1978,1982] which describe the formal semantics of Meta- IV. [VDM 1991] later superseded by the final ISO standard of the VDM Specification Language now called VDM –SL [Andrews 1996]

TOP()e:Item

ext rd s:Item*

pre s $\neq$ []

post e = **hd** s

The example of VDM given above shows a stack specified using a sequence structure and the operation TOP which reads the top item by taking the head of the sequence given it is non-empty.

2.6.2 Z (model based)

Z developed out of a need to improve on one of the perceived difficulties of VDM, that of independence of operations. The interdependence of the component parts of a VDM specification is not explicit in the specification's structure, whereas Z permits quoting and reuse of specification text at various points through its schema calculus.

We see in this example (a schema which describes the check for a non-empty stack) that all information about the stack is included by the statement in the first line.

Ismt

Ξ Stack

b! : Boolean

---

b! $\Leftrightarrow$ s = <>

The concepts and framework were first proposed around 1981 and developed through pilot projects and case studies undertaken by the Programming Research Group at Oxford. A large part of the early work was carried out by Spivey following the pioneering work of Abrial and published from 1985 onwards. [Spivey 1985,1992] The book on Z by Spivey has become the classic reference manual. As was also the case with VDM, variations began to spring up as practitioners adapted the notation and added constructions to suit their needs. The first publicly available version of the proposed ISO Z standard has been produced [Brien and Nicholls 1992] .

2.6.3 B Method (model based)

One of the developers of Z, Abrial, tried to extend the scope of Z to provide

- a formal development lifecycle from specification to code,
- tool support for all stages to realise the potential of formal methods in large-scale projects. These tools would show the correctness of the specification and its refinements.

In 1984 having worked on the method and tools in parallel he presented a unified foundation for predicate logic, set theory and program construction [Abrial 1984] and a proof assistant which was the forerunner of the B-Tool. He developed the work from 1985 together with the B.P. Research Centre Sunbury and the Oxford Programming Group.

The B-Method consists of 3 stages:

1. the system requirements are captured using formal specification in AMN (Abstract Machine Notation);

2. this specification is gradually refined until it reaches a final refinement which is in a simple imperative style programming language;

3. code is automatically generated from this final refinement.

The example is taken from the specification of the abstract machine for a sensor which is picking up information and storing it in an array [Storey 1992].

A feature of the method and tool are the large number of proof obligations it generates

```
MACHINE
        Sensor(noreads)
CONSTRAINT
        noreads:NAT
SEES
        Info
VARIABLES
        darray
INVARIANT
        darray : DTYPE+->seq(REAL) &
        !(xx).(xx:ran(darray) => size(xx) = noreads)
INITIALISATION
        BEGIN
                darray :={}
        END
OPERATIONS
        assign_darray(dd.vv)=
                PRE
                        dd:DTYPE &
                        vv:seq(REAL) &
                        size(vv) = noreads
                THEN
                        darray(dd):=vv
                END
END
```

which must be discharged both to prove the initial specification internally consistent and to show the refinements are correct representations. The tool draws on a large library of proof rules to discharge these obligations and other rules can be added by the user.

Claims made for the tool are that it incorporates:

Configuration management	- for design
Static analysis	-syntax and type checking
Logical consistency analysis	-generating proof obligations for consistency and correctness
Proof Tools	-to discharge the proof obligations
Dynamic validation	-prototyping and animation for validation testing

Code Production	-automatic translation from low level design to code
Built in reuse	-a library of reusable modules
Automatic code generation	-generating from entity relationship models
Rapid prototyping	
Document preparation tools	-indexed cross referenced documents using LaTeX
Automatic rebuilding	-remaking facilities when source files are altered

2.6.4 CCS (process algebra)

The development of CCS was a major breakthrough in the modelling of concurrency and synchronisation and originated with Milner [Milner 1980,1989]. His objective was to develop a framework for constructing and comparing different models in various stages of abstraction. He gives a basic syntax to denote processes and also a system of relationships between expressions including strong equivalence, observational equivalence and observational congruence.

$$\mathrm{CELL}_1(y) = \alpha x.\mathrm{CELL}_2(x,y) + \overline{\gamma}\, y.\mathrm{CELL}_0$$

$$\mathrm{CELL}_2(x,y) = \overline{\beta}\, y.\mathrm{CELL}_1(x)$$

$$\mathrm{CELL}_0 = \delta x.(\textit{if } x = \$ \textit{ then } \mathrm{ENDCELL} \textit{ else } \mathrm{CELL}_1(x))$$

$$\mathrm{ENDCELL} = \alpha x.(\mathrm{CELL}_1(x) \frown \mathrm{ENDCELL}) + \gamma \$.\mathrm{NIL}$$

The example is taken from [Milner 1980] and shows the construction of a pushdown store from a series of cells.

2.6.5 CSP (process algebra)

The standard work on CSP (Communicating Sequential Processes) has long been the book which developed from Hoare's work and lectures on the problems of concurrency [Hoare 1985]. His ideas deal with the problems caused when a system continually acts and interacts with its environment. CSP sets down formally the behaviour of a process in terms of traces of the sequence of actions in which it engages. Hoare gives a full mathematical basis for the method with algebraic laws to underpin the theory. Nondeterministic processes are dealt with and proof rules are given for showing that a design satisfies its specification.

This example shows a phase encoder which is a process T which inputs a stream of bits and outputs <0,1>for each 0 input and <1,0> for every 1 input. The inputs are denoted by having a "?" attached to the channel and the outputs are given a "!" following the left or right channel. A decoder, R, reverses this translation. It is taken from [Hoare 1985].

T = left?x→right!x→right!(1-x)→T

R = left?x→left?y→**if** y=x **then** *FAIL* **else** (right!x→R)

Hoare describes attempts at implementation in Pascal plus, Simula and Ada. However his preference was to convert the specification into OCCAM [INMOS 1984] a simple programming language chosen to make the transition as straightforward as possible. OCCAM syntax was designed to be composed directly on screen with the help of a syntax checker and implemented with static storage allocation on a fixed number of processors.

2.6.6 ADT (algebraic)

The particular method of Abstract Data Types described here is taken largely from Martin's book [Martin 1986]. It developed from the ideas of applying an algebraic technique for the specification of data types as described in [Guttag 1977] and a good introduction to their use is given in [Ehrig et al. 1992]. The method concentrates on the behaviour of operations on data types and describes the data types axiomatically. Large sections of Martin's book concentrate on developing a good understanding of some of the common data types, for example sets, sequences and trees. He demonstrates how to implement the resulting structures and expressions in Pascal but points out that other programming languages could be used equally well.

As an example the following expression specifies the result of looking for the top element of a stack when you have previously put the element e onto the stack s.

TOP(PUSH (e,s)) ::= e

2.6.7 RAISE (hybrid)

Raise (Rigorous Approach to Industrial Software Engineering) was the name of a CEC funded ESPRIT project which ran from 1985 - 1990. The Raise specification language

RSL [Havelund et al 1992] was based on VDM but was designed to incorporate features it lacked. The particular deficiencies of VDM it addresses are property oriented specification, structuring and concurrency. Raise is an all embracing notation with features borrowed from OBJ [Goguen and Tardo 1979], Standard ML [Milner et al 1990], CSP and CCS and is therefore able to handle a wide range of specification domains. The notation is incorporated into a method defining the rules for refinement (the implementation relations) and has a tool set to help with checking, proving and translating.

It claims to cope with a full range of specification features such as parameterisable abstract data types, modularity, concurrency, nondeterminism and subtypes for full development from initial requirements to programs in Ada or C++. As part of the method there are facilities for formal development and correctness proofs.

```
LIST =
        class
                type
                        List
                value
                        empty : List
                        add : Int ×List→List
                        head: List $\xrightarrow{\sim}$ Int
                        tail: List $\xrightarrow{\sim}$ List
                axiom forall i:Int, l : List •
                        [head-add]
                                head (add(i,l)) ≡ i,
                        [tail-add]
                                tail (add(i,l)) ≡ l
        end
```

The example here shows, in the specification of a list some of the elements of classes from object oriented languages, type declarations similar to VDM and axiomatic definitions that have much in common with ADT. It is taken from Hauvelund's book.

An extension of the original project from 1990 -1994 was funded with the intention of improving the industrial potential of RAISE and transferring formal methods into various LaCoS (Large Correct Systems) partners.

2.7 Tool support

Many software engineers who have attempted to use Formal Methods have noted that tool support is essential if anything other than toy size specifications are to be attempted [Computer 1996]. The need for tools is one of the conclusions drawn from the surveys discussed in Chapter 4, [Craigen et al 1993, Austin and Parker 1992], the need is especially pertinent for anyone attempting any proofs or checking. As a typical comment Holloway and Butler regard inadequate tools as one of the serious impediments to the industrial use of Formal Methods [Holloway and Butler 1996]. Lafontaine et al. discuss the difference between general tools and those designed to support a particular Formal Method as they try to apply the B-Tool to a VDM specification [Lafontaine et al. 1991].

Without tools, the claims that a formal specification satisfies certain conditions and invariants, or that a refinement from an initial specification has been carried out correctly, will be almost impossible to check by hand for any reasonably sized specification. Even small specifications can give rise to a large number of statements to check and many proofs to undertake. The validity of the specification and resultant code cannot be claimed under the Formal Method in question unless these are completed.

Even with tools in place if the tools themselves (particularly proof generators and checkers) are not themselves rigorously designed we will be doing the equivalent of checking the slope of a brick wall with a faulty spirit level. Holloway and Butler balance their argument for tools by warning against relying on those in their first stages of development which might contain errors themselves.

Developers of proof assistants must also be aware of the dangers of 'hiding' the rule bases of their proofs to make the tool more user friendly. To have confidence in the proofs the operator of the tool must know how the proof obligations are being discharged and whether the underlying assumptions are valid. It is recognised that carrying out checking and proofs by hand for all but the smallest specifications is a very laborious procedure. Woodcock gives some small examples of proof obligations and their discharge in Z which give a flavour of the detail and work involved [Woodcock 1989]. Work done on the proof theory for VDM-SL and its applications can also be found in [Bicarregui et al. 1994, Bicarregui 1998]

However, relying on an automatic prover provided by a toolkit can also lead to difficulties. The tool automatically generates large numbers of proof obligations because of the problem that, as a machine, no intuition or common sense can be used. Unless the rule base the prover works from is thorough and sophisticated, much time and effort can be wasted while the tool is discharging 'obvious' proofs or 'trivial' cases. Even those interactive tools which enable the user to modify the rule base 'on the fly' as proof obligations are generated can leave the operator having to make too many tedious interventions.

Tool support has been slow to develop although a range of proving and checking tools in various stages of development are available for the more popular methods. As some formal notations have matured to become well-established methods, an attempt has been made for tool support to be incorporated at every stage to supply:

- syntax checking,
- type and scope checking,
- proof assistants,
- translators from notation into code.

Some tools also have a systematic way of dealing with issues of consistency (ensuring that the specification contains no contradictory statements) and completeness (all possible conditions are covered).

The development and sophistication of the tool sets which have been developed not only vary considerably from one notation to another but also between rival products for the same notation. Some companies are now trying to produce more generic tools that can be adapted by the user to incorporate their own grammars, libraries of proof rules and notations - for example Logica's Formaliser [Formaliser 1994]. Most are still some way from being able to generate a formal specification at one end of the process and, through rigorous refinement, produce code at the other although some like the B-Tool make these claims [B-Toolkit] although there is no published evidence of the evaluation of them. A very comprehensive list of current tools with cross references to their sources and suppliers is given as an appendix in the NASA guidebook [Covington 1995].

2.8 Test Generation

Once the software is written Formal Methods can still be useful as part of the testing process. By highlighting those cases giving rise to error conditions, they can be used to generate test cases. There is some work published on how formal specifications can be used for the automatic generation of test [Dick and Faivre 1993, Hörcher and Peleska 1995, Gaudel 1995]. As an illustration if we return to our expression (2)

$$\forall x \exists s ((x \in G \wedge x \in N) \Rightarrow (s \in S \wedge (x,s) \in L))$$

obvious error conditions to investigate would be (with possible interpretations):

$x \notin G$	(What about boys? do they love a sailor?)
$x \in G \wedge x \notin N$	(What about girls that is not nice?)
$s \in S \wedge (x,s) \notin L$	(What about the sailors that the nice girls don't love?)
$s \notin S \wedge (x,s) \in L$	(What about the non- sailors that the nice girls love?)

In general by encouraging the writing of pre-conditions to operations or changes of state explicitly, many Formal Methods force the specifier to consider the conditions which will give rise to errors or will lead to classes of test cases. This can be particularly true of the more tabular methods where columns or rows must be completed for each case showing that all possible conditions have been covered. However de Neumann reminds us that only certain error types will be found by applying formality and also states that to carry out the formal proofs of any realistic system is intractable [de Neumann 1989].

2.9 Users of Formal Methods

In their comprehensive survey Austin and Parkin [Austin and Parkin 1993] sent questionnaires on what formal methods were being used to those who responded to a mailshot of 3000 or to an invitation they posted on an electronic bulletin board. Of the 800 requests for the questionnaire 444 returned it completed and the majority were using formal methods in some form. From the forms that they analysed (just over 100) they found that industrial practise of Formal Methods was concentrated on 5 main notations although a large number of others were in use. The five were VDM, Z, (each being used by 55% of the participants analysed) LOTOS, CSP and CCS (which together were used by a further 18%). They did make the point that the latter three were used for situations where concurrency was important and, as in general that constitutes a smaller proportion of software development, there would be a correspondingly reduced percentage of use.

In the same survey they also found that 89% of participants would use formal methods for the specification phase of software development but this dwindled to 17% for refinement and only 5% for verification.

This survey is discussed in greater detail in Chapter 4 together with the International survey carried out by Craigen, Gerhart and Ralston.

2.10 Myths, supporters and detractors

In his often quoted article Hall addresses 7 of the myths about formal methods [Hall 1990]. His arguments are summarised here:

- Formal methods guarantee that software is perfect.

No, mistakes will be made and limits are also imposed by the coding language, operating system and hardware but some correctness can be demonstrated and some errors found earlier.

- Formal Methods are all about program proving.

No, the formal specification should be the basis for the program and the program should arise from the refinement of the formal specification.

- Formal Methods are only useful for safety critical systems.

No, they can help to ensure that the right software is built for any system.

- Formal Methods require highly trained mathematicians.

No, the mathematics required is easy. The more difficult aspect is the modelling. The inumerate need less than a weeks training in discrete mathematics and a two week course in notation. The majority of software engineers would not be able to do proofs easily.

- Formal methods increase the cost of development.

No, writing a formal specification decreases the cost of development. Proving each step is probably too expensive and in industrial examples you must be careful how the training one off costs are incorporated. The balance of resources used in the lifecycle changes and the increase front end costs may be recouped by lower maintenance.

- Formal Methods are unacceptable to users.

No, they help users understand what they are getting. However commenting and natural language explanations must be used to supplement the formal specification and some limited animation or prototyping is useful.

- Formal methods are not used on real large-scale software

No, they are used daily on such projects, for example IBM CICS, Praxis CDIS project, Rolls Royce and Associates on Nuclear reactor control software.

In this thesis we shall examine some of Hall's arguments, notably the last four concerned with the notation, user-friendliness, cost of development and their use in large scale projects. By looking for hard evidence we will try to see whether these 'myths' about the negative features of Formal Methods can in fact be so easily dismissed.

Alternative scenarios showing the failure to use Formal Methods properly have been the practise of 'reverse specification'. This occurs where there is a legal or contractual requirement to use formal methods, the software is written first using traditional methods and then the formal specification is derived 'backwards' from it.

The lack of published material could be the result of the sensitive nature of the projects. When we made requests to some companies for samples of Z specifications the sensitive nature of the work was given as the reason for it not being made available. In the USA if Formal Methods are used they tend to be for security critical projects rather than safety critical ones so publishing results or material might be difficult.

2.10.1 Claimed benefits

Software failures come to our notice when the results have an impact in our lives. If an aircraft crashes because of a computer fault, a missile system fails and the 'wrong' people are killed or a financial institution loses millions of pounds because of an undetected computer fraud we are alerted to the repercussion of poor software quality.

There are numerous well-documented cases of such failures [Peterson 1996] and the twin consequences of loss of life and revenue drive the search for improved quality. In the USA with its bias towards litigation these dual concerns have combined together in recent years with cases of companies being sued for incidents indirectly attributable to software that they have supplied. In these situations there are incentives for investment in methods that will claim to be rigorous in detecting errors before release and aim to show the program to be provably correct.

It has always been a feature of developments in computing that they are the subjects of hype and vested interests and Formal Methods are no exception. Defect free software is promised if you adopt Formal Methods [Sullo and Williams 1992] and Formal Methods require no more tools than a word processor that can handle mathematical symbols [Potter 1991].

It has been claimed that the earlier faults are detected in the software development lifecycle, the cheaper it is to fix them [Boehm 1981]. By using formal methods at the initial requirements specification stage more of these early faults should be detected. The most damaging types of fault, those caused by incorrect requirements capture, should become rare using a method that puts so much emphasis on a true translation of the clients specification.

A further claimed benefit derived from using the Formal Methods comes after delivery. With so many of the problems discovered during development, delivered software should be more reliable than that developed using traditional methods and with such rigorous, well documented origins it should also be easy to maintain [Hoare 86, Barber 1991].

2.10.2 Indications of limitations

'Oversold and underused' is the summary in Barroca and McDermids paper [Barroca and McDermid 1992] which tries to detail the arguments used for and against the inclusion of formal methods particularly in safety critical systems. In discussion of the strengths of formal methods they distinguish between the principle and practise. The theoretical claims that these specifications are clear, unambiguous, precise, abstract and concise are not always borne out in practise and they make the important point that what is clear between two practitioners used to reading formal specifications is not the same as saying that the notation has clarity. They also stress that the need for precision of the individual statements in formal notation can lead to a mushrooming effect that can make the formal specification far longer than the natural language description.

The first of the weaknesses that they highlight is to do with interpretation. This occurs at both ends of the bridge that formal specification makes between initial natural language requirements and the design and implementation. Here human error creeps in as the specifier must first interpret and translate their understanding of the client's statements. At the other end of the bridge the implementor must re-interpret the formal specification in terms of coding. We conjecture that the fact that the gap between the requirements and code is often filled with a formidable array of mathematical symbols and constructions can lead to more of these errors of interpretation.

Larson and Plat [Larson and Plat 1992] argue that specification languages allow a level of abstraction that executable programming language do not, but they also state that it is not possible to apply a coherent specification notation for all paradigms and applications. This echoes the earlier arguments by Hoare [Hoare 1987] who was refuting arguments for executable notations and for a universal formal notation system.

Similar conclusions are expressed by Naur who advocated a variety of part-specifications in any style that is helpful to the user [Naur 1982]. His conclusion was at the end of a much wider argument on the nature of formal and informal expressions, specifications and proofs. He also makes the point that analysis of a situation and its specification is essentially an intuitive process which can be helped by Formal Methods in appropriate cases and if it suits the user. He presents some damning evidence on the limitations of Formal Methods by comparing a fragment of the official description of Algol 60 [de Morgan et al 1976] with the description of the same part given in VDM [Henhapl and Jones 1978]. He states that the formal description is incomplete, contains numerous errors and inconsistencies and that the proof of consistency seemed to be impossible.

2.10.3 The jury is still out?

In a recent round table feature on Formal methods in Computer [Saiedian 1996] the two groups representing the two sides to the arguments about the adoption of formal methods seemed to emerge. One led by Hall, Bowen and Dill were on the whole advocating training, tool support and investment to spread the methods whereas others represented by Zave, Glass, Parnas and Holloway were expressing caution and wondering why after 25 years the technology transfer had not taken place. They were still waiting for the hard evidence that the methods brought tangible benefits. They wanted solutions to problems they did have, not elegant ways of dealing with problems that did not occur.

Caught in the middle were several proponents of formal methods, Wing and Jones among them, trying to reconcile these arguments either by diluting the rigor with which the methods were applied or restricting those who have to deal with them to a specialised few. An insight to the popular view of Formal Methods within the software engineering community is given by Sullo and Williams:

Formal Methods are like Latin: they will always be part of our high culture but you don't see them used a lot in everyday practice [Sullo and Williams 1992].

2.11 Summary

Formal Methods have been proposed as a means of assuring correctness of systems and they are especially relevant for critical systems. However a major problem is that the need to use formal notation because of the need of formal semantics means that formalism and understanding are sometimes in conflict. A major objective of this thesis is to show that the problems of comprehension are an impediment to the take up of Formal Methods.

We have looked at the origins and nature of Formal Methods and some of the main types. We have considered the importance of tool support and the role of the formal specification as a basis for the generation of test cases. Using arguments from both sides we have summarised the views of supporters and detractors of Formal Methods.

We believe that to ensure the future use of Formal Methods there must be

- clear reasons to use them :

measurable goals in terms of the improved quality of the resulting software,

- a more user friendly approach :

reducing the steep learning curve involved and the additional knowledge required of users,

- increased standardisation:

encouraging a small number to be widely adopted rather than a large number used rarely,

- hard evidence for their benefit:

industry needs studies which collect relevant data in a scientific way to put together a case for their use.

CHAPTER THREE

3. MEASUREMENT AND FORMAL SPECIFICATIONS

In this chapter we look at the need for measurement to establish the case for the use of Formal Methods. We look at the attributes of interest both in software and in formal specification and try to establish a scientific basis for possible metrics in Formal Methods.

3.1 Introduction

In the introductory chapter of the thesis three objectives were given for the measurements made on formal specifications:

1. to measure the effectiveness of the methods themselves,
2. to measure directly attributes of the specifications themselves,
3. to build predictive models based on these measurements.

We now consider the nature of the measurements needed for our objectives. If we aim to show that incorporating Formal Methods into software development leads to higher quality code, we must then decide how this quality is to be measured. So we must look at existing software metrics and decide what we can learn from them. We also need to consider whether all Formal Methods will affect the quality of the code produced equally or whether there are attributes of the formal specification themselves which have a bearing on the production of better software. This leads us to investigate whether we can measure the characteristics of particular formal specifications to find factors that might influence software quality.

3.2 Why measure at all?

Roche states in [Roche 1994]

> *The aims of objective software measurement, i.e. measurement that is independent of the collector, are both technical and managerial in nature and include characterisation and evaluation; control and improvement of software quality and productivity and comparison, prediction and estimation.*

Measurement has long been part of the scientific approach to many subject areas. Growth and progress in fields as diverse as agriculture and economics are measured by statistics which are widely recognised, for example yield/hectare or GNP. We use these statistics to monitor performance and to aid the selection of the best methods to ensure a quality result. If software engineering is to be accepted as a rigorous scientific profession then it must meet the standards applied by researchers and practitioners in other disciplines. Claims about improved performance or lower failures must be assessed by measurements and statistics that can be applied and interpreted universally.

When new methods or products are proposed they need to be measured against established standards. In 1992 the first international standard was proposed for software quality measurement, ISO 9126 [ISO 1991] based on earlier work by McCall developed for the US Air Force [McCall et al. 1977]. This standard was an attempt to define software quality in terms of six factors: functionality, reliability, efficiency, usability, maintainability and portability. Many companies now use this model as a framework for evaluating software quality despite the fact that many characteristics and subcharacteristics related to these six factors are not properly or unambiguously defined.

In an attempt to carry out quality assessment procedures a company may use measurement of a generally accepted level of operational quality like defects per thousand lines of code or the number of post release failures. We shall show in section 3.5 that this can be a problem if faults, failures and defects are not carefully defined. The use of company defined 'in house' terms or standards make comparisons between different products and processes more difficult.

If a business is considering a major overhaul of its software systems it needs hard evidence on which to base its decisions before planning what may be a large investment. It must weigh up the perceived improvement of the proposed new system against the costs and drawbacks of abandoning existing practice. The costs of changing over, with associated retraining and re-equipping overheads, must be balanced by the anticipated savings through lower maintenance or the predicted higher income from better performance. The availability of measurements taken from several products under the conditions most pertinent to the company will mean that an informed choice is made and a product suited to the company's needs and meeting its performance criteria is chosen.

The same is true for companies reviewing their process methods and in particular their attitude to Formal Methods. Incorporating them into software development could mean a complete change in the company processes and a different cultural approach to projects.

3.3 What is measurement?

If we are going to call for data on Formal Methods we need to understand the meaning of measurement. This is a vast topic and the discussions following represent a summary of the most relevant aspects of measurement as they relate to our thesis. The mathematical basis of measurement as given by representational theory is not covered here but can be found in [Krantz at al. 1971].

Measurement of an entity implies assigning to it an objective value which can then be conveyed in a meaningful way from the measurer to others. In one of its most common uses we can say that a particular person is 183.7 cm tall; this gives an objective standard measure well understood by anyone with knowledge of the centimetre. If instead we simply say that a person is tall we have also put a measurement to them but we now have to be sure that if we are to pass this information on, those receiving it have an understanding of what is meant by 'tall'. A comparative measure, as in 'he is bigger than your Uncle Ed', will only be useful if the standard (in this case Uncle Ed) is known.

In his book on software metrics Fenton gives the following definition of measurement;

> *measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules.*

He follows this with a definition of a measure;

> *a measure is an empirical objective assignment of a number (or symbol) to an entity to characterize a specific attribute* [Fenton 1991].

So a measure can be devised and refined as a result of observations of a particular attribute. Metrics are proposed measures which may characterise their attribute with varying degrees of success. Underlying any attempts to develop metrics should be an understanding of the theory of measurement and a framework for measuring attributes which has developed from our understanding of the empirical relations involved.

In scientific disciplines a consensus emerges from the experience of the practitioners of the way that attributes will be defined and that in turn leads to the basis of an accepted measurement. As an example, one of the classifications of rabbits may be 'long eared' and biologists could arrive at a consensus about the way that long eared will be defined which in turn leads to the basis of a measurement for ears.

In our first example we have considered the attribute of height by assigning a number in centimetres as a measurement but also taking a different approach we have given the label 'tall'. This leads us to consider the nature of the measurement and the descriptive types that can be used. The nature of the different ways of assigning values to the attribute gives the scale type for the attribute.

3.4 Scale types for measurement

There are five well-established scale types which we will illustrate using an example of the size of a person as it is usually applied to girth.

Nominal - a naming type of label - e.g. fat, a description which has no order about it but just characterizes that type of person. You cannot compare a fat person with an obese one-there is no underlying scale to tell you which is larger.

Ordinal - labels with an ordering e.g. L from the range S<M<L<XL. This is an ordered set of sizes where S is less in some sense than M.

Interval provides a notation of ordering and a notion of interval between entities. For example measure of clothes in sizes 10,12,14,16 where the difference between 10 and 12 is the same as the difference between 14 and 16

Ratio a measure relative to a known scale e.g. waist 91cm gives a comparative measure to the known scale of centimetres.

Absolute not applicable here but used for items which can be counted using natural numbers, e.g. calories.

The nature of the measurement determines the type of statistics that we can obtain and the conclusions we can draw from the results. If we use an inappropriate scale type or misuse the statistics this will affect the validity of any results we obtain and any conclusions that we try to draw.

3.5 What do we measure?

When we are considering the production of software there is an almost unlimited set of attributes connected to the final product and its production that can be considered. Which ones are given particular attention will depend on the goals and objectives of the measurer. We are interested in measuring attributes to try and quantify the impact of Formal Methods so we will be concerned with two products, the final delivered software and the formal specification. We will also need to consider the difference Formal Methods make to the processes involved in developing the software. Resource issues such as training needs and the use of Formal Methods experts may have a bearing on the arguments about their incorporation into projects.

As we look at how we will obtain some of these measurements we also need to classify what we mean by attributes. Fenton distinguishes between *internal* and *external* attributes to differentiate between the case when an attribute can be measured in terms of its self-contained properties and alternatively when it is measured in terms of its relation to the environment [Fenton 1991]. When we consider the different attributes of software and specifications we can investigate, the picture is complex and classifying them will help us to focus on the areas relevant to our study.

3.5.1 Classification of attributes

In all these discussions 'attributes' has been a rather loose term for the properties and characteristics of interest. Fenton in his book Software Metrics [Fenton 1991] puts all entities which have attributes we might want to measure into the 3 classes, processes, products and resources and gives examples of attributes. Table 3.1 is a summary of the information on software attributes and is reproduced from the book.

The first column lists the most common entities in software development under their three classes. The second column lists internal attributes which are an integral part of each entity. The final column gives the external attributes which are related to the environment and may be consequences of the internal attributes.

ENTITIES	EXAMPLE ATTRIBUTES	
	INTERNAL	EXTERNAL
Products		
Specifications	size, reuse, modularity, redundancy, functionality, syntactic correctness	comprehensibility, maintainability
Designs	size, reuse, modularity, coupling cohesiveness, functionality	quality, complexity maintainability
Code	size, reuse, modularity, coupling functionality, algorithmic complexity control flow structuredness.	reliability, usability, maintainability
Test Data	size, coverage level	quality
Processes		
Constructing Specification	time, effort number of requirements changes	quality, cost, stability
Detailed Design	time, effort number of specification faults found	cost effectiveness, cost
Testing	time, effort number of bugs found	cost effectiveness, stability, cost
Resources		
Personnel	age, price	productivity, experience, intelligence
Teams	size, communication level, structuredness	productivity, quality
Software	price, size	usability, reliability
Hardware	price, speed, memory size	reliability
Offices	size, temperature, light	comfort, quality

Table 3.1 Components of Software Measurement

In considering the attributes in relation to formal specifications the shaded rows of the table will be the most relevant.

Classification and measurement of attributes are not always straightforward, and we take the example of the attributes of a cake as an illustration.

The taste of a fruitcake will be a key external attribute affecting how attractive it is to customers. This external attribute may be very difficult to measure directly but we could look for internal attributes which can be measured and which might be related to the taste of the cake. The metric could be constructed from such internal attributes as:

- fruitiness -measured by the proportion of fruit in the mixture,
- sweetness- measured by the amount of sugar included,
- lightness- measured by the proportion of flour in the ingredients,
- texture- whether nuts and/or cherries were included.

The single metric proposed as an indicator for the external attribute taste could be made up from numbers relating to each of these four internal attributes.

Not only must we now consider the demarcation between internal and external attributes but we must also look at the division between software and specification attributes and the connections between them. We shall concentrate mainly on measurements based on the products.

3.5.2 External software attributes

External software attributes are characteristics of software quality such as reliability and maintainability which are apparent from the users point of view and may be requested as part of the specification. These may be measured by using surrogate quality measures such as failure rates, fault density data and MTTR (mean time to repair). To base the standards of software quality on such measures several terms must be carefully defined. It is common for the terms defects, failures, bugs, faults, anomalies, errors and crashes to be used with widely different interpretations. This problem is compounded when the measurements taken are given in relation to the code length so that rates are used rather than absolute values. The definition of lines of code can also be open to misunderstanding as we discuss later.

In Adams work at IBM [Adams 1984] he found that a range of major systems contained ‘large’ faults which caused failures infrequently and also ‘small’ faults leading to the most observed failures. For example one third of all faults caused a MTTF (mean time to failure) of over 5000 years. Reliability measures are based on failure data, not fault data, and this study suggests that this may not give an accurate reflection of the quality of the software.

3.5.3 Internal software attributes

In contrast to the external ones, internal attributes are not visible to the end user of the software. As examples of these we could include the complexity of the code, the data flow through the code or even something as simple as the size of the software given by total lines of code. These can often be calculated directly from the code fairly automatically. By investigation we may find that there is a relationship between some of the external attributes and certain internal attributes of the software. One of the most obvious links may be between the complexity of the code and the maintenance effort. In this respect McCabe's cyclomatic number [McCabe 1976] is often used as a guide to maintenance effort although by itself it does not give sufficient information about code to be a good indicator.

3.5.4 External specification attributes

In a similar way we can look at a formal specification of the software and consider its external attributes as they affect those who read and interpret it. The ease with which it can be read, its usability and its maintainability will come under this category but these are difficult to assess and as for the software attributes we try and find surrogates for them which can be more easily captured. So we could link the readability with metrics based on the richness of the notations and the structure of the specification.

3.5.5 Internal specification attributes

The internal attributes of a formal specification may be related to the external attributes but are properties of the specification itself which might be of interest in their own right. The structure of a specification may have a bearing on its comprehensibility (external) but also may need to be given a standard or boundary of its own to ensure good practice for writing specifications is established (internal).

3.6 *Linking attributes*

Figure 3.1 shows these four aspects of the software as interlinked and we need to investigate both the different types of attributes and their relationships to each other. We need to find the key optimal internal attributes in formal specifications as they are linked to the external ones. We also need to see if they connect with the desirable attributes of the resultant software which are a measure of its quality.

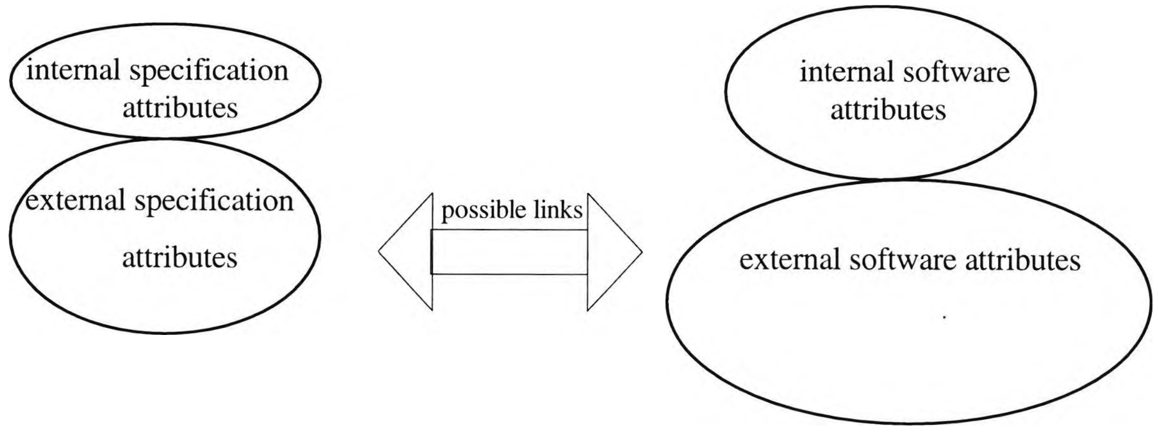

internal specification attributes
external specification attributes
possible links
internal software attributes
external software attributes

Figure 3.1 Showing various attribute areas of specifications and software.

Ideally we would want to define metrics related to the attributes of the specification which we could link to the established software measurements of quality.

If we return to our analogy of the cake representing the product of finished and delivered software, we could take the formal specifications to be analogous to the recipe. Figure 3.2 shows how our attempts to link the various attributes of software and specification are like trying to investigate the properties of the recipe and link them to the final cake.

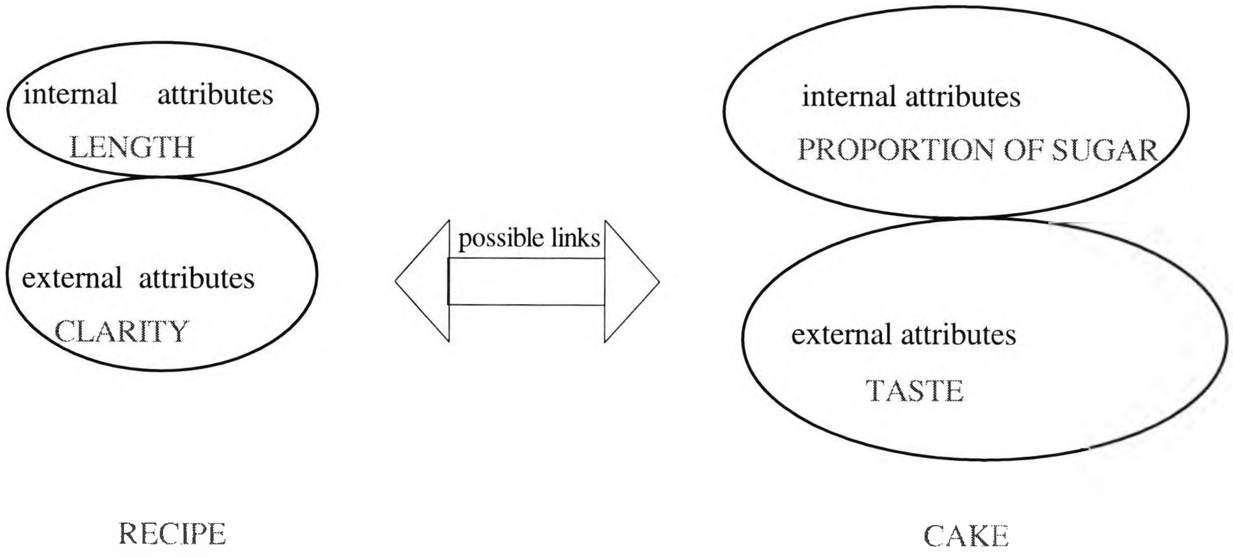


Figure 3.2 Showing various attribute areas in the cake analogy.

3.7 How do we measure?

In a word we must measure 'scientifically'. Measurements which are not carried out correctly under stringent and stated conditions cannot have much weight given to them. It is as if a measurement has been taken with a bent ruler placed at the wrong angle, it is worse than no measurement at all because it could be misleading.

There have been discussions on the nature of software measurement and claims and counter claims for different theories and frameworks for measuring. The scientific approach should underpin two interconnected aspects of proposed software metrics. First, as Fenton has argued, there needs to be an acknowledgement of the role of the representational theory of measurement as a basis for the proposed metrics if they are to be true measures [Fenton 1994]. As a second issue the rationale and methods of obtaining measurements are also to be justified scientifically. In their paper Fenton et al. give guidelines for those intending to carry out scientific evaluation of software which include emphasis on empirical evaluation and data, the design of experiment and the need to have measurements appropriate to the intended goals [Fenton et al. 1994].

The need to establish the circumstances under which the results of measurements were obtained is important; where possible the conditions should be replicated so that further supporting evidence is produced. This is a guard against freak statistics from favourable conditions and unscrupulous operators. Whilst bias can never be totally eliminated, repeated experiments under similar conditions minimise its impact.

3.8 The claims and limitations of measurement

The value of software metrics can be dependent on two aspects of their collection. The first aspect that needs to be examined is whether a particular measurement made on an attribute is an appropriate one for the intended purpose, for example does measuring the number of lines of code give a metric relevant to the cost of a project? The second aspect that must be considered is the scope of the interpretation of statistics collected for example does a metric devised from a small example scale up to an industrial sized project? Limitations to the claims for metrics can arise from failures in either of these two areas.

3.8.1 Internal and external validity

Fenton looks at the validation of software measures and classifies them as *internally and externally valid* with relation to measurement theory. Internally valid measures provide a numerical value for an attribute in such a way that there is an accepted underlying model of the attribute and the measure used preserves relations over the different software under consideration. Internal validity is a necessary requirement for a good measure.

As an illustration if we consider the classification of piano pieces in a fairly common way by the three difficulty levels beginner, intermediate and advanced, then we could propose an underlying model of difficulty. This model might consist of taking the average number of notes per bar, the number of sharps or flats in the key signature and an adjusted total number of bars. This could give us some sort of difficulty measure

$$D = \text{npb} \times \text{key} \times b/10$$

(where npb = average notes per bar, key = sharps/flats in key signature, b = total number of bars)

For D to be a valid internal measure it must be clear that the mapping between our D values and the levels beginner, intermediate and advanced preserves the ordering so that for example an advanced piece would be expected to have a larger D value than an intermediate one. We note that for our proposed metric the other characteristics of internal validity are present namely that it is a clear attribute we are proposing to capture (difficulty) and that there is a clear basis on which D is measured (the given formula).

Inappropriate metrics will make any conclusions virtually useless. It invalidates a series of highly respectable scientific trials if the attributes being measured do not have a demonstrable bearing on the intended area of concern.

When we consider *externally valid* software measures we are looking for metrics that will be used as predictors of a behavioural aspect of the software such as cost of maintenance or expected time to failure. Here it is more difficult to define measures that will map directly onto a numerical measure of the attribute concerned. Fenton draws attention to the difficulty of using LOC (lines of code) as a predictor of anything except the amount of paper used in printing a program out, although he notes that it is an internally valid measure of the length of a program. Incorrect inferences are made when we try to use LOC as a predictor for other attributes such as complexity or running time. Some researchers have, in our view, wrongly rejected internally valid measures on the basis that they cannot be proved to be externally valid [Schneidwind 1992]. One of the major objectives of measurement theory is to find internally valid measures of software attributes.

At present one of the difficulties of defining valid measures either internally or externally is the problem of agreed definitions of common terms. For example the terms faults, failures, bugs and errors are used in connection with code in ill-defined ways which make comparisons very difficult. Even when the terms are precise the counting methods are not always made explicit in documentation; for example when counting faults is a fault only counted on its first appearance in code or is there some account taken of the number of times that the section of code containing the fault is used within a single run of the software?

Evidence for the effectiveness of externally valid measures must be collected with proper experimentation to evaluate the role of these measures in predicting attributes. The design of trials and the appropriate use of statistics have a large impact on whether a particular measurement can be proved to be valid.

3.8.2 Scaling problems

The abuse of the information collected can arise from incorrect use of scaling to extend or project the metrics collected under one set of circumstances to another. As an example it would be absurd to say that as a program of 5 lines of code took 1 minute to write, so one with 5000 lines would take 1000 minutes or just over 16 hrs. Problems often arise from the failure to understand that when a project is scaled up combinatorial factors are involved.

Failure to carry out sufficient trials or the use of an inappropriate time scale can also lead to false conclusions. Common errors include making judgements based on too low a sample size or one that is not representative of the 'population' (usually software not people) in general. So proposed metrics for software should clearly state if they really only apply to particular application domain or special circumstances. Analysis of data to help validate a predictive measure should state how far into the lifecycle the data was collected. In particular, statistics designed to look at maintenance and reliability should be collected well into the post delivery period whilst those designed for cost must take into account all the training and preparation which may take place before the official start date of the project.

3.8.3 Statistics

Another source of error is the incorrect use of statistical tests to prove significance. The underlying assumptions and the limitations of each technique should be stated when they are used. Well-known errors enter the calculations if attention is not paid to independence of factors, levels of significance and assumptions of normality. Any conclusion drawn relating to correlation must also be carefully examined as there is always a possibility of a misleading indirect relationship where both factors under examination are related to a third not under investigation. Brooks reminds us that scientists from other disciplines have been perfecting these techniques of statistical inference for many years and computer scientists should aim for the same standards [Brooks 1980].

A more fundamental problem is the fact that many common statistical operations and tests are only meaningful for interval, ratio and absolute measures. It is not possible to cannot calculate the mean and standard deviation of 3 ‘excellents’, 2 ‘fairs’ and 5 ‘poors’ or any other nominal or ordinal measurements.

3.9 *Common software metrics for products*

By considering some of the existing metrics used in software engineering we hope to:

- understand some of the general principals used in their application,
- look at their strengths and limitations,
- see if they can be applied to formal specifications.

First we studied some of the most common metrics used and then we looked for evidence of their effectiveness as measurements by considering the validation studies carried out on these metrics. Full and detailed descriptions of all the metrics considered, both product and process, appear in Appendix A and the validation studies are analysed in Appendix B. What appears here and in section 3.10 is in the form of a summary of the most relevant work.

We can categorise software metrics by the development phase or phases within the software lifecycle when they are applicable. We shall concentrate on those most pertinent to the specification and design stage but we also touch on some which might apply right through to the coding phase.

3.9.1 Function Points

Function Points [Albrecht 1979] and Symons Mark 2 Function Points [Symons 1988] can be used at the specification stage as they are based on the structure of the model of the software rather than the coding. However the detailed information needed about the inputs and outputs and external files could not directly be applied to a Formal Method like Z which concentrates on capturing the user requirements in relation to the changes in the state. It might be possible to adapt the function points concepts so that they could be used on these methods. Different classes of specification methods which are more abstract than Z or VDM might also be better suited to being measured by this type of method, notably algebraic specification.

The main drawback for both Albrecht and Symonds methods is the subjective nature of the measurements. They rely on weightings given twice in the calculations, once to categorise the items like outputs and files into the 3 categories (simple, average and complex) and secondly to weight the 14 factors which comprise the Technical Complexity Factor on a 6 point scale (0 - irrelevant, to 5- essential).

Function Points were devised to study factors affecting productivity and as a predictor of size and therefore effort. In their paper Jeffrey et al. concluded that Function Points were not much better in this respect than simple count-type metrics. [Jeffrey et al. 1993]

3.9.2 Design metrics

In the initial stages of a system design we sometimes only have data and processes identified. As an initial graphical analysis and design method data flow diagrams (DFD) show the flow of information through a proposed software system. They are made up of 5 different elements each shown on the diagram by a different symbol. One symbolic representation is given in figure 3.3.

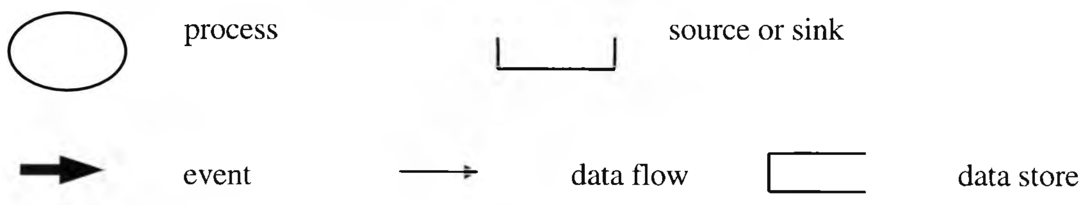


Figure 3.3 A representation of the main symbols of Data Flow Diagrams

When a DFD has been constructed certain metrics can be obtained from the diagram. De Marco, one of the original exponents of DFDs [De Marco 1979], used the diagrams to calculate Function Bang metrics which are a measure of the functionality of the specification.

At the same stage of system design entity relationship models (ER) are ways of capturing significant aspects of a system concentrating on data elements known as entities, their attributes or properties and the relationships between them.

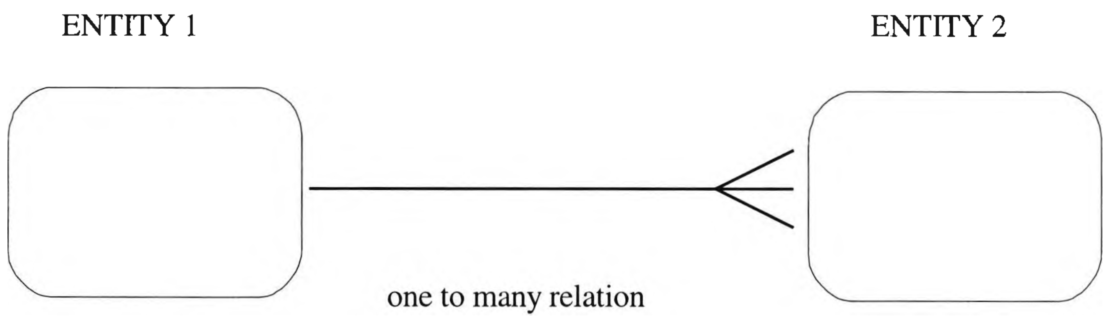


Figure 3.4 Part of an Entity Relation Diagram showing a one to many relation.

The ER diagrams may consist of items of the form shown in figure 3.4, and the relations shown can be of 3 types: one to one, one to many and many to one. De Marco used the ER diagram to calculate the Data Bang metric. These two Bang metrics were together later called specification weight metrics.

As both these metrics involve only calculations of functions and data structures they could be applicable to certain formal specifications but would need adapting from their basis in DFD and ER. One of their advantages is that they can be calculated automatically once the data and functions are made explicit and are therefore, unlike function points, not dependent on subjective judgements.

Z for instance makes its functions very explicit having a different symbol for each type of relation. All the data structures are also clear in the declarative part of the schema so they could be identified easily. The most obvious difficulties in adapting the DFD based methods to Z schemas is the repetition and inheritance of the information from schema to schema and the use of base types which may contain hidden details of the attributes of data. However it might be possible to link these inheritance mechanisms to the levels used in decomposing the top level DFDs into further detailed analysis.

3.9.3 Flow metrics

The next group of metrics we consider are based on the principle of information flow.

These measurements can be applied at the design stage only if there is detailed control flow information and a flowgraph representing an abstract form of the design. At a later stage they can be applied more retrospectively when you are modelling the data flow of the actual program code.

McCabe's metrics concentrate on the complexity of the flowgraph using notions derived from graph theory. This work was extended from single modules to look at the whole system with Henry and Kafura and later Shepperd considering concepts like Fan-in and Fan-out [Henry and Kafura 1981, Shepperd 1990]. These terms refer to the flow into and out of modules and help form a measurement of the total flow of data through a system. Terms like coupling and cohesion are used to describe the relationships between modules and these have a bearing on the complexity.

Bandwidth, as defined in [Lind and Vairavan 1989], is another metric dependent on the structure flowgraph and concentrates on the idea of nesting levels. Fenton extends these ideas by describing how various measurements can be obtained from the flowgraph of a program based on the mathematics of graph theory [Fenton and Pfleeger 1996]. If we try to apply flow metrics directly to a formal specification it must be one that is structured in such a way that there will be forms or constructions with parallels to the flowgraph. For some specification methods like Z, which has an 'inclusion' mechanism so that information is passed down through the structures, it may be possible to apply this type of measurement. The work done by Whitty [Whitty 1990, Bainbridge, Whitty and Wordsworth 1991] in applying flowgraph methodology to Z specifications is discussed in section 3.10. VDM on the other hand has a very flat structure and in its original form did not allow linkages between different operations on the state so it would be difficult to see how a flowgraph could be applied to the overall specification. The algebraic methods where properties are given by recursive definitions would also be difficult to adapt to this structural approach.

All these metrics are supposed to measure internal complexity attributes which are chosen because of their link to external attributes such as development effort and ease of modification.

3.9.4 Halstead's metrics

Halstead developed his Software Science metric to predict the size of code and the effort required generating a program. He used information about the operators and operands and their total number; from this basis he calculated an expected program length and hence effort. This work was later refined by Jensen but in recent years has fallen out of favour as a predictive method [Halstead 1975, Jensen and Vairavan 1985].

As it only depends on the operators and operands a formal specification containing that information in accessible form would be a suitable candidate for this type of measurement. The figures produced are usually used to predict program length and development effort so these results would have to be reinterpreted in the light of their impact on a particular Formal Method.

3.9.5 Length metrics

The most commonly used measurement of software length is lines of code, LOC. Even this simple definition is open to different interpretations as we consider

- blank lines,
- comment lines,
- headings.

Some metrics use non-commented lines of code, NCLOC, as a basis while others rename these effective lines of code, ELOC. It is sometimes useful for attributes such as maintainability to consider the commented lines of code, CLOC, separately whereas for a link with effort and productivity NCLOC might be better. Other length metrics try to take into account the difference between 'headings' lines which usually contain data declarations and header statements and lines of code which are executable. There are still problems with all these measurements resulting from the style of individual programmers. Whereas one will use a very open structure with a new line for each instruction another will condense several statements together on the same line. As with all metrics a measure of length will be valid within the constraints of its definition.

In formal specifications similar consideration must be made of the definitions of total number of lines of the specification. Many formal notations include headings, comments and natural language sections. We shall see in Chapters 7 and 8 that the same specification can be written with different styles and structures affecting the overall length.

3.10 Common software metrics for process

We need to consider these as part of our investigation into the effectiveness of Formal Methods. If claims are made that by using Formal Methods there is an impact on faults found and the maintenance post release we need to consider the metrics that are related to these processes. Some of the product metrics like Function Points can be used in a predictive way to estimate some of the process attributes like cost of development and effort involved.

3.10.1 Defect counting

As has been previously mentioned the terminology here causes difficulties when trying to assess whether a project completed using Formal Methods would differ significantly as far as defects are concerned. We shall use the standard definition of a fault as a human error which had led to a mistake in a software product, usually the code; reviews and testing are designed to identify these. A failure occurs when the system does not do what it should and is more usually detected when the software is operational. The claims that are usually made are:

- using Formal Methods will ensure fewer faults and failures[Hoare 1986],
- the use of formal specification will ensure faults will be detected at a much earlier stage and are therefore easier and cheaper to fix [Cohen et al. 1986],
- the rigour of Formal Methods will ensure that the code will be well designed and structured and therefore easier to maintain [Baber 1991].

To investigate these claims we are usually looking at the testing and review process and comparing defect densities. Defects can be a fault or a failure and the defect density is given by

$$\text{Defect density} = \frac{\text{number of known defects}}{\text{product size}}$$

This again gives us the problems of measuring size by LOC or some similar metric. A lot of defect density figures are concerned with the quality of the delivered software. What is also important as far as the influence of Formal Methods is concerned is the stage in development at which the defect is detected.

3.10.2 Maintenance metrics

In broad terms maintenance is the process of making changes to a product. These changes may be instigated to correct faults and failures or to improve the product with new features. To measure this process we are interested in the number of changes required and the time spent implementing them. Mean time to repair, MTTR, is defined as the average time it takes to make a change and restore the system to working order. Maintenance effort will be linked to such product metrics as size, structure and complexity and several models have been put forward to connect them [Belady and Lehman 1976, Neil and Bache 1993].

3.11 Application of software metrics to formal specifications

As we have stated in the introduction to the chapter we need to use measurement both for assessing the impact of Formal Methods on software development and for considering properties of the methods themselves. Having considered a large number of software metrics it would seem that we now need to split these two purposes of measurement.

The evaluation of the difference that Formal Methods can make to software needs to be achieved by using software metrics in their traditional way - applying measurement to the software produced from projects incorporating Formal Methods and to the processes that are involved in the development of this software. To this end we need to look at feedback from industrial projects and reports from software engineers working in commercial settings. These are covered in detail in Chapters 4 and 5.

To investigate the application of measurement on the formal specifications themselves we need to see how well we can apply the software metrics to this area.
In our summary shown in Table 3.2 the main metrics described in the Appendices A and B are listed together with the stage where they apply, the method of their collection and their purpose.

We can see from the third column that most require the specification or design to be in a certain form before the metrics can be calculated. Ideally any measurements taken on formal methods should be related to the characteristics of the specification itself and not after its transformation into graphical, tabular or modular form. Formal specifications are not normally written in an executable form (although some Formal Methods have been developed to include various prototyping or animating procedures) so measurements which are intended to capture behaviour will not be appropriate.

SOFTWARE METRIC	WHEN APPLICABLE	HOW IT IS DERIVED	ITS PURPOSE
De Marco Specification weight metric	specification and design stage	from DFD & ER	to measure complexity predict effort
McCabe Complexity	design stage	from the flowgraphs	>10 signals problems in a module written from this flowchart predicts effort
Coupling Fenton and Melton	detailed design stage	from coupling model graph	Complexity
Cohesion Bieman and Ott	detailed design stage	from the module structure	modifiability complexity
Henry and Kafura	detailed design stage	from module diagram and length	information flow
Shepperd IF0,IF3, IF4	detailed design stage	from module diagram	software quality predicts development time
Band width (Lind and Vairavan)	detailed design stage	from nodes on control graph	effort, modifiability
Halstead's Software Science Metrics	design stage	from operators and operands	to predict size of code and the effort needed to generate the program
Jensen	design stage	a refinement of Halstead	
Albrecht's Function Points	specification and design	data sources inputs and outputs	to predict size and productivity effort
Mark 2 Function Points (Symons)	specification and design	data sources inputs and outputs	Information processing size
COCOMO (Boehm)	different stages mostly coding	cost drivers drawn from environment factors size from LOC or source instructions	cost estimation (effort estimation and productivity)
LOC NCLOC ELOC	at coding stage	counting on program listing	effort, cost

Table 3.2 Common Software Metrics

There has been very little work on applying metrics to the specifications themselves or attempts to capture the characteristics of any individual notation. Ventouris and Pinleas did carry out a comparative study of algebraic specification methods using a set of criteria including comprehensibility, minimality, and ease of construction and executibility [Ventouris and Pintelas 1992]. Their responses were mostly qualitative in nature but some ordinal scales were used. Ardis et al. also tried to establish a framework for evaluating specification methods using 11 criteria they describe as fundamental and 5 they class as important [Ardis et al. 1996]. They evaluate each method on a four point scale giving it strength, adequate, weakness or not applicable as a consequence. We now look in a little more detail at two attempts to apply metrics directly to formal models.

3.12 Structural metrics on Z

This work by Whitty and others was an attempt to apply measurements to the structural nature of Z specifications [Whitty 1990, Bainbridge et al 1991]. It drew on ideas from Goal Question Metric (GQM) [Basili and Rombach 1988] and the application of metrication through program decomposition [Fenton and Whitty 1986]. In their measurement of Z schemas they used the following three restrictions in their undertaking:

- they concentrated on single schemas,
- they concentrated on the predicate parts of the schemas (not the signatures),
- they took a particular model of the predicate expressions.

They proposed the use of the 'short circuit evaluation model' which defines the flow of control through any predicate. This in turn makes use of the redundancy involved in evaluating logical expressions i.e. a $\wedge$ b and c$\vee$ d need not be fully evaluated when a is false or c is true (assuming a left to right evaluation). So the flow of control for expression a$\wedge$b can be represented by the graph shown in Figure 3.5.

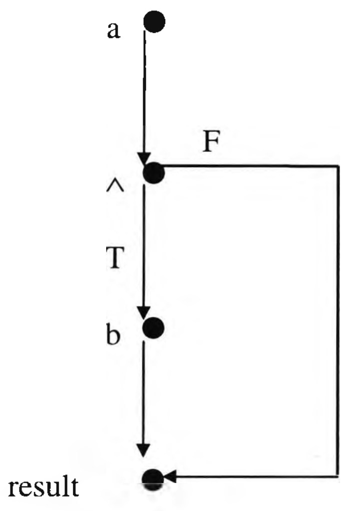


Figure 3.5 Graphical representation of a$\wedge$b

This can be extended to construct graphs involving quantification and implication giving quite complex expressions a graphical representation. In the later paper an attempt was made using a predicate file generator and a flowgraph analyser tool to collect metrics from the original Z schemas. The output was in the form of

- a boxplot showing node size,
- a scatterplot of depth of nesting against highest value prime.

Two specifications were tested and analysed, the first of 90 pages with 180 schemas and the second with 200 pages and 500 schemas. In summary they felt that the automatic collection of data derived from the structure of the schemas was essential if the metrics were to be used in the design phase of systems development. From the two types of plots they obtained as output they felt the most useful information might be the identification of the outliers. These might be derived from schemas identified as being of a more complex nature and therefore difficult to comprehend. Showing up as extreme values might indicate that these schemas should be partitioned into smaller ones in later reviews.

A tool was developed based on this work as part of the COSMOS project [Whitty and Lockhart 1990]. The authors were trying to model the cognitive notation associated with short circuit evaluation but it was not substantiated by empirical data and will not be adopted in this thesis.

3.13 Notation metrics

This study by Britton et al. reported at ESCOM 97 was an investigation into the notations used in software modelling [Britton et al. 1997]. Originally the work had a particular context of specification of multimedia for educational systems but in their conclusions the authors state that the paper presents a foundation for measurement for any specification notation.

They worked on a wide range of modelling techniques, 18 in total, and under the more formal notations included Z, VDM, CSP, Predicate Logic, Petrinets and some logic. The attributes under consideration were:

- richness - measured by the entries in a key or glossary,
- perceptual /symbolic- measuring the use of diagrams and labels,
- formality- this measured the relationship of a technique to branches of mathematics and its rigour,
- coverage -this covered the extent to which each method could model any of 8 facets of a system.

Tables were produced from a theoretical approach i.e. no empirical data was used and all calculations and nearly all measurements are taken from the definitions and glossaries of the notations. The results are difficult to interpret and use because some tables give definite figures for their entries (CSP 104, Petrinets 4 in glossary entries) whilst in others the entry does not allow much differentiation between methods (16 out of 18 do not handle time). The authors recognise the limitations of the results and acknowledge this as a first step to building and validating measures for notations. Nevertheless we shall find some of this work is closely related to our early work details of which are given in Chapter 6.

3.14 Conclusions

In our search for metrics to measure the effectiveness of Formal Methods we have considered the range of existing metrics as applied to software products and processes. We have considered two separate ways of using metrics:

1. to use them in a traditional manner to assess the software products and processes in various projects which have incorporated Formal Methods,
2. to adapt them where possible to apply them directly to the processes and products of formal specification.

We consider the first use of these metrics in Chapters 4 and 5 as we look at the surveys and case studies and attempt to measure the effectiveness of Formal Methods in practice.

Few of the metrics seem very suitable to transfer directly as measures of the attributes of formal specifications but several give indications of possible areas of interest. They have drawbacks for immediate application to Formal Methods for a variety of reasons;

- the attributes they are intended to measure do not correspond to a characteristic in Formal Methods,
- they have not proved to be effective as measures or predictors of the chosen attribute,

- the method of deriving the metric is neither scientific nor objective.

However we explore this second way of using metrics is in Chapters 6, 7 and 8 where we look at applying some product metrics to specifications concentrating on aspects of the notation and structure. We will also explore their connection with process metrics related to the reading and maintenance of the specifications.

CHAPTER FOUR

4. SURVEYS OF FORMAL METHODS

In this chapter we look at the evidence available on the use of Formal Methods presented by those using them in a variety of commercial and academic situations. We concentrate on the results presented by two large surveys of users of Formal Methods and the validity of the conclusions that were drawn from them. In particular we note that many of the results are based on subjective judgements rather than quantifiable properties.

4.1 Introduction

The two major surveys analysed here were undertaken as an attempt to get some feedback from industrial, research and educational settings concerning the use of Formal Methods. The approach by the two surveys is very different as is the presentation and analysis of their results. The first survey reported here concentrated on the Formal Methods aspect of twelve projects representing three main areas of application and they are divided into regulatory, commercial and exploratory categories. The second survey took a different approach and invited wide participation by use of bulletin boards and mailshots. Of the responses obtained the results are based on the analysis of 126 replies, about a quarter of those returned.

Both surveys were trying to look at the benefits of incorporating Formal Methods into a project and both tended to rely on subjective judgements from the teams involved.
They each put together a framework of questions referring both to the areas of the projects which might be affected by Formal Methods and also the implementation of the methods themselves. The first survey obtained responses by using questionnaires followed up by personal interviews with the members of the project teams whereas the second used postal questionnaires to those who expressed an interest from the original advertisements.

4.2 INTERNATIONAL SURVEY OF INDUSTRIAL APPLICATIONS OF FORMAL METHODS

by Dan Craigen, Susan Gerhart and Theodore Ralston.

This was a study sponsored by the U.S. National Institute for Standards and Technology, the U.S. Naval Research Laboratory and the Atomic Energy Control Board of Canada. Its remit was to look at the then current use of Formal Methods by industry, and to assess the efficacy of Formal Methods for meeting the needs of these three organisations. The main report was published in September 1993 [Craigen et al 1993] and some of the work was summarised in other articles [Gerhart et al 1993, Craigen et al 1995]. A lot of the material for the tables and summaries is drawn from these papers.

4.2.1 Overview of the projects

They covered 12 projects in a variety of industrial settings across North America and Europe. They grouped these by three categories: regulatory, commercial and exploratory.

Regulatory

In these cases the project was of a safety or security critical nature leading to some involvement from standards committees or regulators. Under this category came:

DNGS

Work done to give assurance to the Atomic Energy Board of Canada about the shut down systems for the Darlington Nuclear Generating Station.

SACEM

A project for the rapid transit authority in Paris (RATP) on a signalling system that would allow the time separating trains to be cut from 2.5 to 2 minutes whilst not compromising safety requirements.

MGS

This Multinet Gateway System is a device that ensures secure delivery of datagrams over the Internet. It was tested under the US Trusted Computer Evaluation Criteria process.

TCAS

Work on the logic and surveillance parts of the Traffic Alert and Collision Avoidance System designed to reduce the risk of mid-air and near mid-air collisions in aircraft. The work was required by the US Federal Aviation Authority

Commercial

Concern with profit and savings was the drive for these projects and they were:

CICS

Customer Information Control System, IBM's large transaction processing system (mentioned in more detail in Chapter 5) was re-engineered for a new release.

Cleanroom

Two application of the Cleanroom methodology were investigated. The ground support for a Satellite at NASA and the development of a COBOL Structuring Facility converting old COBOL to a semantically equivalent form.

Tektronix

The firm in Oregon developed 'non-executable prototypes' to develop reusable software architecture for a family of oscilloscopes.

Inmos

This manufacturer of microprocessors used Formal Methods in three interrelated projects: to specify the IEEE Floating Point Standard, a scheduler for the T-800 transputer and the Virtual Channel Processor for the T9000.

SSADM

Praxis developed a toolset to support the use of this Structures Systems and Design Methodology.

Exploratory

Here the organisations were investigating the potential of using Formal Methods.

HP

The Analytical Information Base of a patient monitoring scheme was a real-time database and its specification was undertaken in Hewlett Packard Specification Language (HP-SL) to see if there was any value in this type of technique.

LaCoS

Lloyd's Register and Matra were interviewed on their experiences under the ESPRIT projects on the evaluation of RAISE. Lloyds were looking at data acquisition and equipment management systems on ships and Matra builds systems for railways and other applications.

TBACS

A group from the National Institute for Standards and Technology specialising in Computer Security Technology were developing Token -based Access Control Systems which involves smartcards with cryptographic authentication.

4.2.2 Methodology

The study was carried out by using questionnaires and structured interviews. An initial questionnaire was used to prepare the way by getting participants to consider some of the issues involved in incorporating Formal Methods into a project. This was returned before the personal interviews and used as initial feedback. The structured basis for the interviews was an attempt to get a uniform approach to the 12 projects so that an analytic framework could be constructed for the later studies of the interviewer's reports. In total 23 interviews took place with 50 individuals, interviews lasting anything from half an hour to 1.5 days. About 1.5-person years in terms of effort was spent on the survey. The reports produced as a result of this process were sent for comments to the participants.

The varied nature of the projects made it impossible to get a totally uniform approach to the questions used in the interview stage but both the initial questionnaires and the interview questions covered six areas:

- organisational context,
- project content and history,
- application goals,
- Formal Methods factors (the why and what of selecting them),
- Formal Methods and tools usage,
- results.

Details of the size of each project and the Formal Methods used is shown in table 4.1.

PROJECT	**APPLICATION AREA**	**SIZE**	**FORMAL METHODS USED**
SACEM	Automatic train protection	9 KLOC	Hoare Logic (B method)
TCAS	Mid Air collisions	7 KLOC	Statechart-like language TCAS Methodology
DNGS	Logic for shut down of nuclear generator	1362LOC FTN 1185LOC Asm	A-7/SoftwareCostReduction (SCR) Tabular descriptions
MGS	Internet device	10pp maths 80pp Gypsy 6 KLOC OS	Gypsy Verification Environment Graphical Notation
CICS	Transaction processing	50 KLOC	Z
Cleanroom	restructuring Cobol	80 KLOC with 20 KLOC reused	grammars and transformers of Cobol Structuring Facility
Tektronix	Oscilloscope products	200 000 KLOC (30 pp Z)	Z
Inmos	Floating Point unit for transputers	Occam then VHDL (100 pp Z)	Occam,CSP (not mentioned in Gerhart 1993) and Z
SSADM	Toolset Support	37 KLOC (320 pp Z)	Z
HP	Real-time database	(55pp HP-SL)	Hewlett-Packard Specification Language
LaCoS	Experiment with RAISE		RAISE
TBACS	Security properties of Smartcard	2.5 KLOC related to 300 lines FDM	Formal Development Methodology

Table 4.1 Summary information about the 12 surveyed projects

	DNGS	MGS	SACEM	TCAS	SSADM	CICS	Cleanroom	Tektronix	INMOS	LaCoS	TBACS	HP
Satisfaction	+	+	+	+	n/a	n/a	n/a	+	+	+	n/a	0
Product Cost	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	+	n/a	0	0
Impact	+	+	+	n/a	n/a	+	n/a	+	n/a	+	+	n/a
Quality	0	+	+	n/a	+	+	+	+	+	n/a	+	+
Time-to -Market	n/a	n/a	n/a	n/a	n/a	n/a	n/a	+	n/a	n/a	n/a	0
Process Cost	-	n/a	0	n/a	n/a	0	0	0	+	0	0	0
Impact	0	0	+	+	+	+	0	0	+	0	0	+
Pedagogical	+	+	+	+	+	+	+	+	+	+	+	+
Tools	n/a	0	+	n/a	-	+	0	-	+	0	+	0
Design	+	+	+	+	+	+	+	+	+	+	+	+
Reuse	n/a	+	+	n/a	n/a	n/a	n/a	+	+	n/a	+	0
Maintenance	n/a	n/a	n/a	n/a	n/a	+	+	n/a	n/a	n/a	n/a	n/a
Requirements	+	+	+	+	n/a	+	n/a	+	0	+	+	+
Verification & Validation	+	+	+	n/a	+	+	+	n/a	+	n/a	+	+

Table 4.2 The responses to the 14 questions on the influence of Formal Methods

4.2.3 Analytic framework

By this means Craigen et al tried to highlight patterns across the 12 projects by characterising the effect of the use of Formal Methods in 14 areas. They called this the 'feature analysis' as they were concentrating on those features i) considered important for industry and ii) where it was suspected Formal Methods might bring benefits. The responses for the role of Formal Methods within these 14 areas of the project are graded into three types: + positive, 0 neutral or - negative. These subjective judgements were made by the researchers on the basis of the reports and using their knowledge of the organisations' usual approach. The gradings that are given are relative to this usual approach and the results are given in Table 4.2.

Taking the literature available about this survey as a whole it is hard to get a consistent and concrete view of the effectiveness of Formal Methods and our summary in Table 4.3 reflects this problem showing overall from a possible 168 entries the proportions of each response. The large numbers of n/a were entered if either there was no information available on this characteristic (not available) or if the characteristic was not relevant to a particular project (not applicable) and these also make generalised conclusions difficult.

	Regulatory	%	Commercial	%	Exploratory	%	Overall	%
+	29	52	37	53	19	45	85	51
0	5	9	7	10	12	29	24	14
-	1	2	2	3	0	0	3	2
n/a	21	37	24	34	11	26	56	33

Table 4.3 The response rate of the features for the three different categories.

However we can see from this analysis that overall the Regulatory and Commercial projects seem to have a similar profile but the Exploratory cases were slightly different. It is interesting to note that the final percentage of positive reactions to the effect of

Formal Methods is just 50.6% (all figures in the table have been corrected to 2 significant figures) and that the Exploratory projects, designed to achieve technology transfer or experimentation with Formal Methods, seem to have given rise to more neutral reactions.

In their reflections on the study, the impact of the researchers' statements is often reduced by the diverse nature of the projects. In some projects there was a concerted, management driven, company effort to incorporate Formal Methods and in others just one individual who wanted to experiment with a new approach. Dividing and subdividing the cases as the survey authors have, (grouping into the three categories and dividing into 14 features) has meant that often arguments seem to based in one instance on a single experience only to be contradicted by evidence elsewhere in another section. For example if the 'Time to market' feature is considered, any conclusions from the 12 projects will in fact be based on two responses one positive reaction and one neutral. The authors have stated the limitations of their work but nevertheless try to draw general conclusions relating to the three categories of projects.

4.2.4 Conclusions and recommendations

4.2.4.1 Regulatory conclusions

The drive to use Formal Methods here is one of demonstrating a high level of assurance. This means that the ability to specify requirements and build models using Formal Methods is almost secondary to using them to prove that the code conforms to requirements. The main findings of the survey in this area claim to be:

- regulators and developers discovered together what Formal Methods could do and how much evidence it provided for certification purposes,
- Formal Methods may be used in a different way according to context (as a design tool, as a documentation aid, as a testing framework, as a validating and proving mechanism etc.),
- no regulatory body in the survey would accept only Formal Methods based assurance, and none accepted just testing; most wanted a combination of techniques,

- tools support was essential for this group as they had most need of proof and refinement. In some cases tools may even be specified to ensure their quality as well as that of the development,
- certification is costly and here consumed a large proportion of the project budgets. Lengthy time was spent on Formal Methods because of the assurance required,
- it is not clear how Formal Methods should or will be used in this area. The two main applications could be for product certification and process evidence.

4.2.4.2 Commercial conclusions

In some senses the dividing line between the regulatory and the commercial projects was artificial as many of the companies going for certification under the former category were looking to develop the product later on a commercial basis. However in this group the main motivation in incorporating Formal Methods was to affect the success of their products in the market place, so the factors of most interest here were time to market, impact of the product, cost and cost benefit and quality. In the event, parts of several projects proved unsuitable for Formal Methods use whilst conversely advantages were gained in hindsight beyond the expected benefit of clarity.

The authors noted the bias in this category towards Z with 4 out of the 5 cases using this in part or entirely as the specification notation. This seems to have been not because of the inherent properties of Z itself, but because of other external factors such as availability of experts, availability of tools and strong advocacy from Z enthusiasts and governments.

The main findings for this category were:

- conceptual benefits from using Formal Methods can be achieved without an extensive toolset, whereas...
- development and assurance aspects of Formal Methods will need tool support to achieve full industrialisation,
- cost effectiveness and performance were impossible to assess as the cost and cost benefit analysis metrics used by industry cannot be applied to Formal Methods processes. The performance data collected using accepted metrics was also inadequate.

4.2.4.3 Exploratory conclusions

These three cases were the hardest to compare and [Gerhart et al. 1993] does not discuss them at all whilst [Craigen et al. 1995] absorbs them into the other categories. From the original report we can see that the aims, time allowed for incorporating the methods and the personnel involved was very varied. The LaCoS project involved a large commitment in time and people in a collaborative project under Esprit funding (cushioning the real costs) whereas the HP project was a group passing on techniques to an individual and TBACS was the reverse with a single person wanting more involvement in Formal Methods and passing the ideas to a group and both these projects had a brief time span. Perhaps not surprisingly there was a more positive attitude in the LaCoS project although as Table 4.2 shows several of the measurable features are left without comment. Both HP and TABACS chose not to pursue the specific Formal Methods used in their projects after this study.

The main conclusions for this category were:

- the 'culture' of a firm may have a significant effect on its willingness to adopt Formal Methods (or any new/ different methodology),
- tool support might have been of benefit for editing and checking.

There are fewer conclusions to this section as the exploratory nature of the projects meant that there were not many end products or processes to compare. There was also the previously mentioned divergency leading to few common experiences or results.

In the authors' summary the points can be condensed to:

- the role of Formal Methods is wider now than a narrow specification to code process,
- there are now large scale real world applications of Formal Methods,
- Formal Methods are often used in a re-engineering role for clarification and precision,
- they may be incorporated in future certification processes,
- tool support was neither necessary or sufficient,
- technology transfer is slow even in 'sympathetic firms',
- education in Formal Methods as part of software engineering is weak in the US,
- Formal Methods do not adequately cover all environments,
- metrics and models to measure benefits are not generally available.

In their recommendations the main points were:

- there is a need for better integration of Formal Methods with other techniques,
- tools need to be robust and integrated with other parts of the development,
- notations less mathematical in nature are required,
- Formal Methods need to evolve alongside OO, CASE, multimedia etc.,
- automatic deduction support is needed at least for regulatory cases,
- the gaps in the application areas that Formal Methods can cope with need to be filled,
- efforts must be made to widen the user base in this and other complex technologies.

4.2.5 Critique of the international survey

This was a very important piece of research in terms of the size of the projects involved and the detailed information collected. However there were limitations due to the basis on which it was conducted which are noted in [Gerhart 1993] as:

1. limits in time and effort to about 1.5 person years,
2. bias on behalf of the surveyors, sponsors and interviewees who have vested interests in applying Formal Methods,
3. the case selection which left out certain classes of methods,
4. cases of failures of Formal Methods were not included.

In their summaries and recommendations they make important points about the slow rate of adoption of Formal Methods by the software engineering community and the need to continue with quantitative studies that can measure their impact on projects. However if studies as detailed and far reaching as this one are to have impact some of the bias due to the above limitations must be addressed.

1. This is purely a funding issue and may be a reflection of the lack of interest in Formal Methods by commercial software producers.
2. This area is very difficult for all researchers who have to keep an open mind about the value of techniques which have become their specialised area. The solution may be to form mixed groups of Formal Methods supporters and detractors as an assessment panel or to conduct some sort of blind testing as was attempted in a small way in the Trusted Gateway project discussed in chapter 5. The dependence on

opinions rather than facts could be removed altogether with proper objective measurements. The strongest argument for the use of metrics to evaluate the benefits of Formal Methods is that it removes just this type of bias and the authors in the last sentence of the report do suggest that metrics should be an area for further research.

3. This is not such a serious limitation as it gives a clear boundary between what is under discussion and what is outside the remit of this study.

4. Possibly this limitation raises the most serious source of bias which makes Table 4.3 even more difficult to get into perspective. The team argued

... we did not seek examples of outright failure. Such cases - probably dozens of them - would probably be manifested in drop-outs and dispersion of personnel. And the project duration was too limited.[Gerhart et al 1993]

The failure to include these negative experiences is not entirely explained away by the limited duration of the project. At least 3 or 4 cases of this type should have been covered to give balance to the report.

Overall the lack of well designed, unbiased, scientific measurements make most of the conclusions to this survey very subjective. The authors do allude to some attempts at collection of data with reference to metrics but seem to suggest that even these were useless because of a lack of comparative data from similar projects without Formal Methods. They imply that because the development of a project using Formal Methods is such a different process that the 'typical' metrics would not handle these cases. They also do not distinguish between metrics which could have been derived from the specifications and those which could have been obtained from the resulting software.

Clearly we would disagree strongly with these conclusions on the use of metrics. Simple size and effort metrics would have been easy to collect on the specifications and faults and failures must have been noted in the development processes and testing reports.

The report's findings and recommendations, seen in the light of the subjective nature of the results, seem to leave the researcher open to the criticism that have made supporters' claims for Formal Methods based on the selected opinions of other supporters.

4.3 AUSTIN AND PARKER'S SURVEY

by Stephen Austin and Graeme I Parker

This was a literature survey and a survey of industry conducted in 1992 to look at the reasons for the low acceptance of Formal Methods in industry. A preliminary summary was produced [Austin and Parker 1992] and then four papers were written and combined together in Parts I - IV of the later report [Austin and Parker 1993]. These will be considered separately. In chronological order the literature survey reported in Part IV took place first and the industrial survey was an attempt to see if the claims and counterclaims were matched in practice.

4.3.1 Part 1 Overview: Survey of Formal Methods in Software Engineering

This was the overview of the industrial survey and there is much duplication between this and Part II. In general the information that is printed in full in Part II with tables of individual questions and detailed responses is condensed here and histograms are used to present the information as percentage summaries. All conclusions are reproduced exactly in both parts.

4.3.2 Part 11 Survey of Formal Methods in Software Engineering

This survey was conducted to try and find out the reasons for the low acceptance of Formal Methods in industry. After a mailshot of 3000, a questionnaire was sent to the 800 who requested it, 444 replies from industry were received and of these only 126 of these replies were analysed.

No reason for the use of just 126 replies is given nor is the method of selecting the 126 from the 444 stated. These are serious flaws in the basis for the survey and Table 4.4 taken from the paper shows apparent bias in the way the 126 were selected.

Questionnaires	Received (abs)	Received %	Analysed (abs)	Analysed %
Post (UK)	385	87	104	83
Post (Foreign)	48	11	3	2
Electronic (Foreign)	9	2	3	2
Discarded			15	12
Electronic (UK)	2	0	1	1
TOTAL	444	100	126	100
Research (UK)	60	14	18	14
Research (Foreign)	3	1	0	0

Table 4.4. Analysis of returned questionnaires

We see that for example although 11% of the 444 returned questionnaires returned by post were from a foreign source only 2% of those analysed were from this source. This could imply that the 126 were not truly representative of the original 444 responses.

The discarded replies consisted of those that were not complete or some from participants who thought that a structured method was a formal method.

The authors aimed to reach a wide spectrum of people and organisations and of the 109 different company businesses they put them in the following categories:

7 Safety Related businesses,

8 Scientific & Technical consultancies,

11 Education businesses,

20 Research,

35 categories with 6 or less participants of the same type (averaging 2 returns per category).

They asked 24 questions that were designed to be completed in under half an hour in total. Throughout the results tables Austin and Parker have distinguished between the wider group containing both those who have considered using Formal Methods as well as those who use them, and the narrower group containing just practitioners of Formal Methods. The main results are given in five sections:

1. The method used.

Under this heading came the question about which Formal Methods had been used and they found the main ones to be

Z	58%
VDM	62%
LOTOS CSP CCS	26% (concurrency methods)

There were a large number of other less popular methods mentioned.

2. The way it was used.

The extent to which Formal Methods were used in the lifecycle was

Specification	92%
Proof	48%
Refinement	26%

Formal Methods were not used by more than 6% of participants in later stages of the software development.

3. Combinations with other methods

The only other methods that seem to have been combined successfully with Formal Methods were structured analysis and design methods (SSADM and Yourdon).

4. Benefits, limitations and barriers

This section partly mirrors the one used in Part IV but there were large amounts of additional information from the industrial questionnaire respondents. For each of the three questions on benefits, limitations and barriers several answers could be accepted from the respondent and they were asked to rank their answers in order of importance. For example if the question was

What do you consider to be the barriers to the use of Formal Methods?

The answer might be:

1. notation

2. training

3. cost

This ranking system makes the interpretation of the tables in this section quite complicated.

Benefits

Over half of the respondents put the lack of ambiguity leading to a clarification of requirements as the highest priority on their answers. Few named cost saving as a high priority benefit.

Limitations

No limitation was given priority over any other although readability, costs, difficulty and mismatch with problem domains were all mentioned.

Barriers

Lack of tools was given by about a quarter of the replies and increased costs was mentioned here as well as in the limitation responses.

When asked for suggestions to overcome these problems training was given as the first priority by about 40% of the respondents.

5. Ways of assessing Formal Methods

Here 23% of returns mentioned a lack of suitable metrics as a problem in assessing the contribution of Formal Methods to the software lifecycle. A further 19% mentioned the difficulty of collecting metrics and using them in comparisons.

Under the section asking for suggestions on assessment methods 39% wanted detailed case studies of actual projects. These should be undertaken preferably in parallel to studies not using Formal Methods.

Summary

Austin and Parker conclude from the industrial survey:

- that lack of tool support is an important factor in adoption of Formal Methods,
- there is insufficient evidence of cost benefit in adopting Formal Methods,
- many barriers and limitations are only a symptom of a new methodology being adopted.

4.3.3 Part 111 Survey of Formal Methods in Higher Education

This was a survey of the teaching of Formal Methods in higher education and was carried out by a questionnaire delivered by mailshot to 94 institutes likely to be teaching

Formal Methods. Messages were also left on electronic bulletin boards and e-mail replies were encouraged. The total responses were 39 from the mailshot and 26 via electronic mail and out of this a total of 60 replies were used.

The statistics quoted are on the basis of 7 questions about the type of Formal Methods taught which courses it was taught on and to what level was the method taken. Another 8 background questions were asked for which no results are given but if these are the same ones as reproduced in Part II they are just information like contact details and addresses. In brief the results were

- Z was the most popular model based specification notation (78%) followed by VDM (54%),
- CSP was the most popular concurrency specification notation (17%),
- as expected, Formal Methods is taught on Computer Science, Mathematics and Software Engineering courses,
- whilst all respondents used Formal Methods for specification only 56% had gone as far as refinement and 54% had attempted proof.

Summary

The authors state that these conclusions are not very surprising.

4.3.4 Part IV Benefits , limitations and barriers to Formal Methods

This was stated to be a literature survey and concentrates in each section looking at the claims made throughout a range of about a dozen papers and books. They looked at the meaning of each statement in the three categories and discussed its validity. Their table of 'results' gives six possible responses to the statements

Yes	definitely true	**No**	definitely false
Yes$^-$	true with reservations	**No$^-$**	false with reservations
Yes but n/a	true but does not apply	**Undecided**	

Benefits

In this section they draw heavily on Hall's statements [Hall 1990] and comment on claims that formal specifications

1. are unambiguous,

2. are no harder to understand than a program,
3. are shorter than a program,
4. are shorter than an informal specification,
5. can to be analysed with tools,
6. remove errors at an early stage,
7. can describe functionality,
8. can prove properties,
9. can be used to develop implementation,
10. can be produced for any piece of software,
11. are easily produced for large packages,
12. have been used on large commercial projects,
13. decrease implementation costs.

In their opinion they found all these benefits to be true, definitely true for the majority of points and true with reservations only in the cases of points 3,4, 9,11 and 13.

Limitations

The limitations are drawn from a much wider range of literature by Hall, Wing, Cohen and Jackson and are stated as:

1. the drawbacks of the modelling process itself, abstraction necessarily leaving out some aspects,
2. some situations (time constraints performance) are difficult to define with formal specifications,
3. highly trained mathematicians are needed,
4. Formal Methods do not design the system,
5. proofs are essential to Formal Methods and add considerably to development time,
6. mistakes can be made in specifications,
7. Formal Methods cannot be applied to all types of software,
8. development costs increase with Formal Methods,
9. the specification is not readable by the clients,
10. Formal Methods do not scale up,
11. Formal Methods do not integrate with other software techniques,
12. different Formal Methods cannot be combined.

They found a lot of the limitations were due to the nature of modelling in general, (1 and 2), or limitations that don't just apply to Formal Methods (4,6, and 7). Other limitations that were raised were refuted with some reservations (3,5, 8-11). Only limitation 12 was left undecided by the authors.

Barriers

There were eight barriers to the use of Formal Methods drawn from the literature:

1. finding a problem with software when Formal Methods should have guaranteed a perfect product,
2. highly trained mathematicians required,
3. the software lifecycle will change and become more front loaded in time and effort,
4. Formal Methods have not been integrated into the software lifecycle,
5. Formal Methods have not been used on commercial projects,
6. Formal Methods are not mature enough,
7. tools are not available for Formal Methods,
8. no standards exist for Formal Methods.

The authors were undecided about the first barrier, agreed with 2 and 3 and refuted the rest.

Summary

Austin and Parker felt that all the claims for Formal Methods were justified and discounted most of the limitations as being of a general modelling nature only taking the need for mathematicians seriously. To the use of mathematics they added tool support as a possible barrier.

4.3.5 Critique of Austin and Parker's survey

The industrial survey had a flawed basis in its methodology. Designed to confirm or refute the conclusions drawn from the literature search, there is no rationale given for the way that responses to the questionnaire were chosen for analysis. The layout of the results is made more complicated by double entries for each table and yet nothing is made of the different responses for those actually practising the use of Formal Methods as opposed to the wider group. Open ended responses are difficult to quantify and the priority system of ranking responses from each individual, whilst giving an idea of the relative importance of the answers, makes interpretation of the results difficult.

The educational survey gave limited information about the teaching of Formal Methods in higher education but it would have been useful to have more background on the type of courses, the expertise available, whether there was industrial involvement etc. The eight questions not discussed may have contained this information. Some of the drawbacks here may be due to the limitations of the survey method which required academics to fill in and return questionnaires. It is perhaps better to get a good response rate for a small number of questions than to give people so much to fill in that they do not bother to return the questionnaire.

The literature survey seems to be a flawed piece of research with a lack of objectivity which leave the study open to charges of bias in favour of Formal Methods. Arguments are dismissed in a couple of lines and the only reservations they seem to have are with the use of mathematics and some lack of tools. It is a useful summary of the arguments taking place in the literature around the time (1992) but the tone of the 'discussion' seems to be dismissive of anyone who was not a supporter of Formal Methods. Their conclusions about the benefits have no doubts and seem in direct contrast to other work we have already discussed in Chapter 2, Naur for example. Some of the points are repeated in more than one section and the number of different responses could have been reduced to eliminate 'No' by rephrasing the statements with use of negatives. This would have made the summary tables much easier to read instead of the mixture of Yes's and No's to positive and negative statements giving four different possible combinations.

Together these papers and results represent a lot of time and effort. However the biased view of the authors, which clearly comes over in the literature survey, and the questions about the statistical basis of the results unfortunately devalue the authors' conclusions.

4.4 Conclusions

The glaring omission from both of these surveys is the lack of hard unbiased evidence on which the conclusions are based. Very little is reported about measurements taken or data collected and they are both full of opinions and conjectures. Measurements taken on the projects are not mentioned and the use of metrics are largely dismissed by Craigen et al. as impossible to use in the context of Formal Methods.
Austin and Parker had a broad sweep approach to the collection of data but did not use a scientific, quantifiable analysis of the data they amassed.

None of the studies had a base line with which to compare software quality. All judgements were then rather subjective and isolated.

In terms of evidence for the benefits or otherwise of Formal Methods both surveys fail to provide it in a form which can pass any rigorous examination.

CHAPTER FIVE

5. A CRITICAL REVIEW OF FORMAL METHODS (IN PRACTICE)

In this chapter we consider commercial or industrial projects where Formal Methods have been used in practise. We give detailed analysis of four in particular: the Cyclotron at Washington University, the CICS project at IBM, the CDIS project developed by Praxis for CAA and a smaller project at BASE developing a trusted gateway.We also include brief summaries for a selection of other published cases.

5.1 Introduction

So far in the preceding chapters we have looked at the twin themes of Formal Methods and metrics which run through this work. To focus on possible attempts at measuring the effect of Formal Methods we have analysed the results of two surveys which sought to catalogue their usage both in education and industry. Unfortunately we found that the results and conclusions from these surveys were not based on objective scientific measurements but more on subjective opinions collected in a somewhat biased manner. We now try to look more closely for evidence of the measurable effects of Formal Methods by studying cases of their usage in commerce and industry.

In their recent book entitled Applications of Formal Methods, Hinchey and Bowen have compiled 15 reports from a wide range of industrial applications to inspire others who wish to look at the possibility of incorporating Formal Methods in their own domain [Hinchey and Bowen 1995]. It was made clear in the preface to this book that the aim of the authors was to inspire others. However on closer inspection these reports contain very little hard evidence on which to base the inspiration. Only two give any indication that measurements were taken, results analysed or that there was even a working hypothesis underlying the introduction of Formal Methods into the projects.

With the two exceptions, any details or explanation given tend to be as examples of the formal notation or as descriptions of the problem domain. No data is included which could be used as a basis for measurements and quantitative analysis. If the system reaches the stage of implementation no comments are made about the difference the inclusion of Formal Methods made.

This is not only true of this book (which is in other ways interesting) but also of computer literature as a whole where it is very difficult to find case studies which have been carried out with the intention of gathering evidence about the effects of Formal Methods within a project. Literature searches using standard library resources, computer journal databases and the wealth of information available on the World Wide Web have yielded disappointingly few studies where any attempts have been made to evaluate the difference Formal Methods make.

Liu, Stavridou and Dutetre in their article giving examples of the use of Formal Methods in safety critical systems imply that judgements are made on some other basis than empirical evidence [Liu et al 1995]. They state:

Interestingly, and despite the lack of any documented factual evidence as to their efficacy, formal methods are clearly considered desirable in safety critical systems.

What we have collected here are four main case studies in which an attempt has been made to address some of the questions about the effect of Formal Methods in a realistic commercial setting. Each one has been included because it illustrates a different aspect of the practise of incorporating Formal Methods.

The first study, the cyclotron project, is of special interest as we have had access to the development of this project in a very privileged way and been in touch over a period of years with Jonathan Jacky, the main force behind it. This has given us insight to the development processes and the unique nature of the specification and implementation of this safety critical process. The evidence presented in this case is not given from many measurable quantities but from the insight and experiences of Jacky as he investigated new territory and techniques. He has also reflected on what benefits and drawbacks formal specification has brought to his project and the limitations of the Formal Methods used.

In contrast we look at one of the most commonly cited applications of Formal Methods in industry, the CICS project at IBM Hursley. There was documented evidence that measurements were taken in this case and an often quoted figure of 9% saving in costs together with a claim for a significant increase in post-delivery quality. We question the basis of the study and the interpretations of the results.

While considering the third project, the CDIS work undertaken by Praxis, we have the benefit of the detailed analysis undertaken by the team working on the SMARTIE project [Pfleeger et al 1995]. Their very thorough documentation on faults and testing added a second quantitative dimension to the comments from the original specification team.

The last study is small in size but is of interest because the scale of the problem allowed parallel development to take place, that is the same project was undertaken with and without Formal Methods. The team also attempted to take some relevant measurements during the development of the project.

In addition to these four studies a collection of projects are mentioned at the end of the chapter which represent most of the well known applications of Formal Methods.

5.2 CASE STUDY 1 The Radiation Therapy Machine

(The application of Z in the design of a control system in a safety critical area.)

5.2.1 The history of the project.

The University of Washington installed a machine called a cyclotron as part of its cancer treatment department in 1984 which was supplied with a computer operated control system. This facility was considered at the time a significant step forward in the treatment of cancer patients. Under the American health system patients are referred to the Radiation Oncology department at the University from their doctors. The cost of the treatment is normally paid out of the patients' health insurance directly to the department. Referrals come from all parts of the state, from other states and even from abroad in the cases of special types of cancer.

The smooth installation and running of this machine is vital not only because of the safety issues involved, but also for economical reasons. Failure of the machine to meet its specification in any way could result in the closure of the department. In particular

- incorrect delivery of prescribed treatment could lead to damaging patients' health, and the resulting law suits for incompetence would be financially crippling,
- a small interruption to treatments because of temporary breakdown would lead to great inconvenience to patients who have travelled long distances to get treatment,
- a major breakdown of the machine could lead to referrals being switched to other similar facilities with a subsequent loss of revenue.

In the extreme case of a major breakdown the loss of revenue and 'customer loyalty' would probably mean an irrecoverable situation and the closure of the department. We can see that the software driving this machine, as well as being safety critical due to the nature of the machine, had also to be reliable due to its vital role in the department's future.

The present system, installed in 1984, took 9 months to be fully functioning from delivery as there were many problems associated with its operation. After installing and working with this system for some time the department were not very happy with it and wanted to develop a new control system that would be easier and quicker to use, would be easy to maintain and could incorporate future hardware and software developments. They decided to develop a new control system while there was no immediate need so that there would be no time pressure on the project. However this time they could not afford a delay in getting the replacement machine fully operational as this would mean a gap in service when no cyclotron was working either under the old or the new system, so the new control system had to run correctly immediately from installation. No existing commercially available system met the requirements of the department.

5.2.2 The Cyclotron

The computer systems for the cyclotron controls among other items a 900 A electromagnet and a 30 ton rotating gantry and the system has to deal with large numbers of input and output signals. The main treatment room using the apparatus has a patient couch that can be moved into different positions and a moving floor that can be opened to let the gantry rotate underneath the patien (see figure 5.1).

The operation of these positions is part of the control function. The delivery of the treatment beams uses many control mechanisms including the angle of delivery, the appropriate dosage and the manipulation of the multileaf collimator (a device to ensure the beams are targeted onto to the right area).

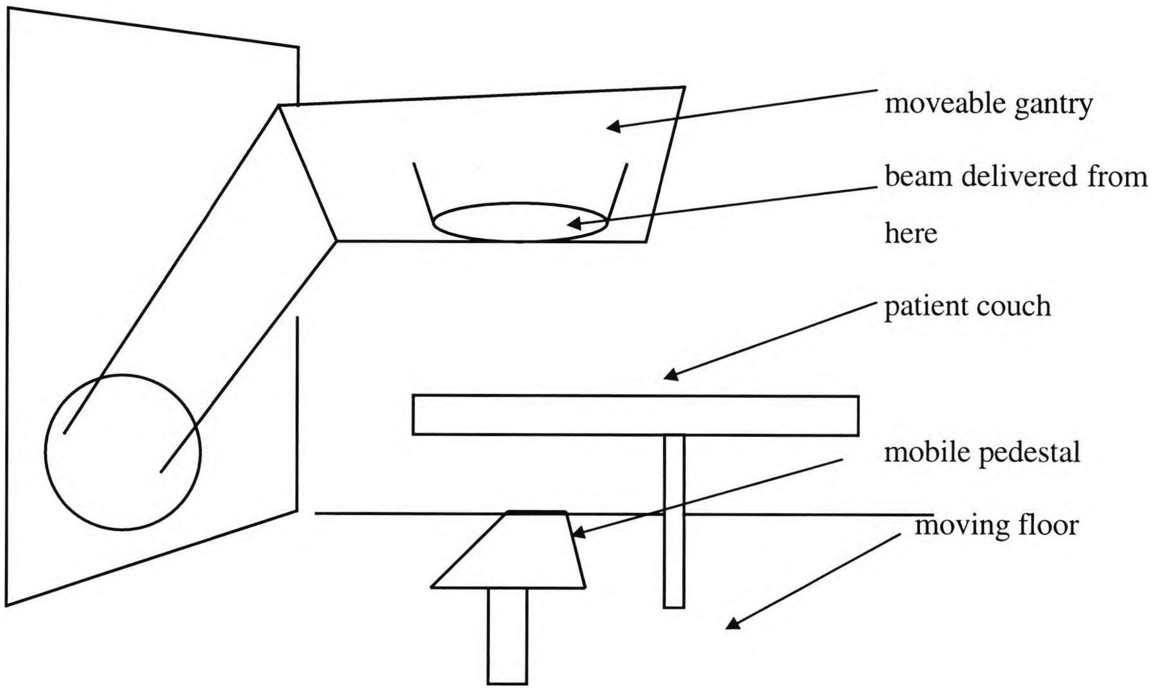


Figure 5.1 A diagrammatic sketch of the cyclotron and the patient area.

The beam can also be switched between the main room and a second treatment room which is mainly used for experimental work. When a patient has been set up in the correct position on the couch and the dosage and other parameters have been checked, the operation of the beam takes place remotely from another room. The set up of the beam is the responsibility of the cyclotron operator but the final administration of the dosage is in the hands of the therapy technologist.

The set of people involved in the use of this machine includes operators and therapists using the machine for treatments together with physicist and engineers concerned with its operation and maintenance. Some researchers are also involved with work carried out in the second treatment room.

5.2.3 The software project team

One of the unique features about the development of this project was the involvement throughout of Jonathan Jacky. He has been at the department for about 15 years after some time in animal laboratory research following a Doctorate in Physiology. Since the beginning of the project to rewrite the control system he has been working on the informal specification, the formal specification and also writing the resultant code for the implementation. Until the last year or so this has been in addition to his other work and he estimates that over the life of the project the time spent on it has added up to about 3 man years for him.

To date those who have been named as co-authors of the published literature on this project are

Informal specification	Formal specification	Implementation/Coding
J Jacky	J Jacky	J Jacky
R Reider	J Unger	J Unger
I Kalet	M Patrick	M Patrick
P Wootton		D Reid
J Unger		R Risler
S Brossard		

However due to the length of time of the project (nearly 10 years), some changes in personnel have meant that the formal specification and implementation have largely been carried out by Jacky working with one other person at a time.

The original computer control system consisted of over 60,000 lines of code in Fortran and several thousands of lines of assembler code. None of this old code was reused as the new design was intended to be very different from the old.

5.2.4 The informal specification

There had been a small team of people working closely together for several years under the old control system. They understood the requirements very well and were quite articulate in their demands for what the new system should deliver. The informal specification of the new system, written in natural language, took all their experiences and requirements into account. The informal specification was very thorough and consisted of about 250 pages of prose and diagrams capturing the desired behaviour and user interface. It was collated and edited by Jacky and had many iterations before a final version was produced [Jacky et al 1990,1992]. It was the intention that by taking great care to ensure the prose would capture the requirements specification, the rest of the development would be easier. These informal specification documents also formed the basis of the user manual and have not changed except in minor details since they were completed. None of the informal specification was written with a view to turning it into a formal specification as at the time nobody in the team had any knowledge of Formal Methods.

5.2.5 The formal specification

This was undertaken mainly by Jacky working with Unger and Patrick..

Jacky has estimated that there have been about 20 revisions of the Z specification and at least 3 major rewrites involving starting the whole formal specification from the beginning. The most recent version available has about 130 schemas so far and a total of 1137 lines of Z.

A major difficulty has been to get the level of abstraction right. If the level was too abstract then the jump to code seemed impossible. If the level was too concrete they ended up writing pseudo code. The strategy for the modularisation of the control system into logical subsystems was also a major problem. It was decided to split the formal specification into 2 levels:

- an abstract level expressing the overall safety requirements. This would provide a basis for properties that might need to be checked or proved,
- a concrete level serving as a detailed design and the basis for coding. It was intended that the formal specification should be written in such a detailed way that the translation to code should be obvious.

File organisation, file access and the appearance of the VDU displays were not specified formally as it was felt that the informal description captured their behaviour adequately.

In his paper [Jacky et al 1997] Jacky comments on the lack of guidance on large commercial specifications in available literature. He developed his own strategy for coping with a large Z specification, as follows:

- identify high level invariants,
- identify state variables,
- partition the state,
- define sets of Identifiers,
- define functions from these identifiers to values,
- identify the operations,
- identify when the operations are to be invoked,
- identify processes,
- decide what to leave out,
- use tabular form to organise the schemas and conditions.

This last point was crucial in the implementation stage.

Jacky also noted the dearth of examples on the linkage between the three stages of the development; informal specification, formal specifications and the code. In order to provide a connection between the informal and formal specifications the team, and in particular Jacky, undertook an extensive system of reviews and cross checking. Each paragraph of the Z was accompanied by the reference to pertinent sections, page numbers and often line numbers of the informal specification. In this way the team satisfied themselves that *all* the informal specification had been captured within the Z specification.

The formal specification concentrates on the therapy console which in turn consists of five subsystems. These subsystems are concerned with the following parts:

gantry/psa (all motion of the apparatus around the beam and the patient)

filters (flattening filters and wedges)

leaf collimator (the control of the shape of the final area to be treated)

dosimeter (dose monitoring channels)

therapy interlocks (safeguards on items like doors and operator keys)

5.2.6 The resulting Z

The Z document was produced after the extensive informal specification to act as a bridge between the natural language and the implementation code. It was intended to be the main source of guidance for the development of the software. In the event this proved to be the case largely because once again the main writer of code was Jacky together with other software engineers (Unger then Patrick then Reid)

The Z document became a very detailed design document showing every action of the console operator modelling right down to the level of individual key strokes. Consistently, preconditions were used as guards so that operations would not take place unless they were true in a logic sense.

The constant reviewing process of the Z against the informal specification meant that some errors were discovered by inspection. Numerous trivial errors had already been found by the type checker fUZZ [Spivey 1992].

It was during the pencil and paper review process that one of the most significant aspects of this case study came into use. As Jacky mentioned in his recommendations for tackling large Z specifications the team made great use of tables. These are similar to state transition tables and consist of columns under the headings:

- Z Operation name,
- state precondition,
- input precondition,
- progress post-condition.

In this way there was a systematic drawing together of all the relevant schemas to a particular operation. The completion of these very detailed tables with their lists of preconditions and post - conditions formed the basis for confidence about invariance, completeness, determination and liveness. It also brought out errors of conflict as there was, within the table, the opportunity to see schemas acting together.

Some main classes of errors are mentioned in [Jacky et al 1997]. Of particular interest are:

- omitted cases,
- inconsistent inclusion of schemas,
- displaced preconditions,

- overbooked state variable,
- implicit invariant.

5.2.7 Implementation

Implementation was spearheaded by Jacky. The final implementation had to be in C because of the interfaces to existing software and hardware. Pascal had originally been the target language but platform changes and technology updates half way through the project forced changes on the team. At the time of writing about 6000 lines of code have been written and trials and tests are underway before the target date of the first real trial in August 1997. Jacky suggests that the process of converting the Z to C is very obvious as

- there is a Z schema for i) each system event and ii) the transmission or receipt of every message. Each of these become modules in C,
- basic types and free types in Z become enumerations in C,
- Z state variables became program variables and data structures,
- Z operation schemas become functions and procedures,
- module inclusion in C arises naturally from Z schema inclusion.

5.2.8 Proof and refinement

No serious attempts have been made to refine the Z to code although Jacky did attempt early formal proofs of correctness on a portion of Z with less than 100 lines. This portion of Z led to a page of Pascal which was not in the final version after the target language had changed to C.

In his talk to the Z User Conference Hall [Hall 97] considered the analogy between the process of software development and a string of beads, see figure 5.2. With the requirements at one end and the code at the other he stressed that the "length of string" or gap between them could not be shortened but might be bridged in a variety of ways with many informal and formal techniques.

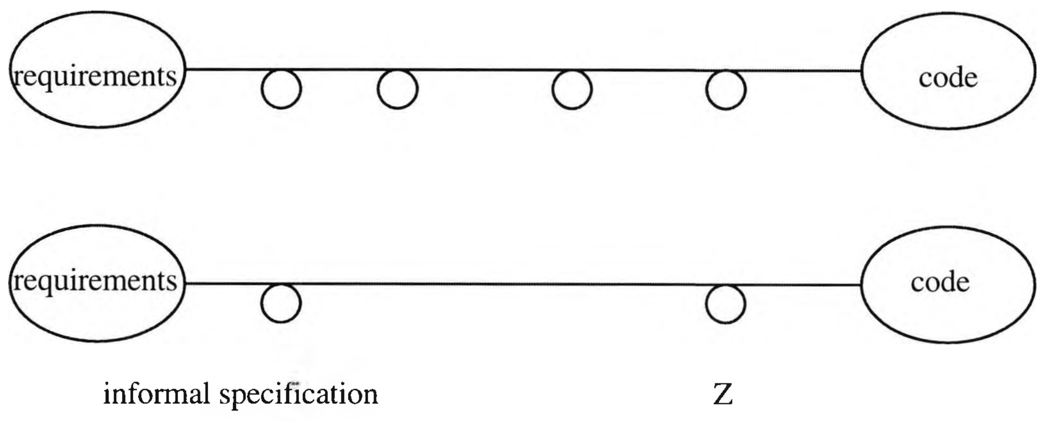


Figure 5.2 Hall's String of beads applied to Jacky's specification

What Jacky and his team did was to work very hard on the informal specification to make the match to requirements very strong and that part of the "string" short. They also put in a great deal of effort in making the step from Z to code a small one so that implementation into C could be almost automatic by inspection of the Z structures. This however left a large gap between the informal and formal specification and it was only after considerable effort, review and cross checking that the team satisfied itself that all the information had been correctly mapped between the two.

5.2.9 Summary

Jacky himself admitted in a review of the work at the Z User Conference in 1997 that the uniqueness of their situation led them to plough on through with little guidance and that in retrospect the constant reiteration and redesign was not perhaps the best approach. He stressed that over the 10 years, the processes of ensuring that the specification was captured informally, the formal specification matched the informal, and the final refinement to code was faithful to the specification had all been very hard. However he remains a supporter of Z and felt that it clarified the thinking about the requirements into a form which could be translated into code.

The process of specifying in Z was used very much as a design tool to discover the best model and there was no obvious bridge from the informal to the formal. The important use of the tabular forms, translating groups of schemas into a basis for coding, meant that the formal notation seemed to need an 'organisational tool' to make it a useful basis for code.

The fact that there was no direct mapping between the formal and informal led to a large amount of time spent on cross-referencing between these two. This might make modification of the system difficult.

The very unique set of circumstances in this project make it difficult to draw any firm conclusions about the way Z has been used or the measurable benefits of incorporating it in the development. The errors (if any!) in the software produced from the Z won't be fully apparent until the code testing is nearer completion.

As was mentioned in the introduction to this chapter the analysis of the impact of Formal Methods on this project has largely been a subjective one by Jacky. Even in retrospect some further work could be undertaken to evaluate their effect. Jacky's implicit hypotheses seemed to be

- using formal methods would ensure the safe operation of the machine,
- the resulting software would install and run smoothly

As there are as yet no details of any code metrics, the installation process or of the operational performance of the machine, so we rely on Jacky's judgements that the formal specification added to his assurance about the system.

What we do have are several versions of the Z specification. As a basis for future research it would be possible to analyse these for internal characteristics and to link them with the final code. As these versions are from 1990,1994 and 1997, it would also be interesting to note those parts of the Z specification which were discarded, changed or added as the specification evolved. Attributes of the specification which could be investigated and measured include:

- structure-did this alter between versions and was it more or less complex in the final version?
- readability- did familiarisation with the project lead to more or less effort in making the specification comprehensible to others? What technical level is assumed? Are natural language explanations a large part of the specification?

- style- was the Z written in good style so that other users (programmers and designers) could read and interpret the Z? Is the notation used with regard to reading by non-specialists?

The uniqueness of this project makes recommendations difficult about similar studies. The shortage of manpower in the specification team meant that few measurements were taken and version control was patchy. For a future undertaking along these lines more control, measurement and documentation of the specification together with metrics on the code and the post delivery performance could lead to a greater evaluation of the effects of Formal Methods on this type of project.

5.3 CASE STUDY 2 CICS : Customer Information Control System

The material for this section is drawn largely from the 3 reports by Phillips [Phillips 1989], Houstan and King [Houstan and King 1991] and Collins Nicholls and Sorensen [Collins et al 1991] Some of the analysis here has already been published [Finney and Fenton 1996].

5.3.1 What is CICS?

CICS stands for the Customer Information Control System and it is an on-line transaction processing system with many thousands of users worldwide. It was originally developed in 1968 by IBM with new releases approximately every two years since then. By 1991 it consisted of 800,000 lines of code, some in Assembler some in PLAS (a high level internal language). A major restructuring was planned in 1983 to clarify internal interfaces and as a basis for future developments. The aims of this revision were to increase the reliability, reduce modification costs and improve the documentation.

5.3.2 History of Oxford IBM collaboration

In 1981 Professor Tony Hoare of the Oxford University Programming Research Group had met Tony Kenny the IBM CICS manager at Hursley Park by chance. This resulted in a research contract between IBM and PRG in 1982 to study the application of formal techniques to large-scale software development. In the initial studies undertaken by PRG two notations were used for comparison, CDL (IBM's internal Common Design Language) and Z (the specification language developed at PRG). Primarily because of this research it was decided that Z would be used for most of the new code required in the restructuring of CICS.

The training of IBM staff in the use of Z techniques was contracted to PRG whose personnel were on hand throughout the project in a consultancy role.

The training that took place included

- 2 weeks for programmers on Software Engineering techniques,
- Z courses for writing intended for designers and developers,
- Z readers' courses for non-programmers.

In 1984 there were about 130 on writing courses, 33 on courses for reading specifications and 51 taking courses in refinement techniques.

From the point of view of PRG their involvement in the CICS project had as one of its aims to see if the concepts and techniques developed by PRG could be applied by programming teams in industry to control the development process and increase confidence in final correctness. Their aim was to answer the questions:

- could aspects of large and complex systems be captured in mathematics?
- would there be practical benefits?
- who in the development team should use the methods?
- what training would be needed?
- what tools support was necessary?

5.3.3 Claims and awards

This case study concentrates on reports and claims relating to the IBM's CICS/ESA version 3.1 which was released in 1990. The project was prominent in the international survey of industrial uses of formal methods [Gerhart et al 1993, Craigen et al 1995] as the largest commercial application.

The quantitative claims made about the effectiveness of using Z on CICS are impressive. The most notable are:

- 2.5 times fewer customer-reported errors,
- 9% saving in the total development costs of the release.

The cost saving is particularly important because, while reliability improvements are an expected benefit of using formal methods, there had been no such expectation of economic benefits. Indeed, [Bowen and Hinchley 1995] cited this cost saving in the CICS study as the main counterexample to the 'myth' that 'formal methods delay the development process'. Because of the high profile nature of the project, the claims about CICS have had far-reaching ramifications on the public perception of formal methods. For example, in 1992 IBM together with the Oxford University Programming Research Group won the highly prestigious Queen's Award for Technological Achievement. The citation reads:

> *The Queen's Award for Technological Achievement 1992*
> *Her Majesty the Queen has been graciously pleased to approve the Prime Minister's recommendation that The Queen's Award for Technological Achievement should be conferred this year upon Oxford University Computing Laboratory. Oxford University Computing Laboratory gains the Award jointly with IBM United Kingdom Laboratories Limited for the development of a programming method based on elementary set theory and logic known as the Z notation, and its application in the IBM Customer Information Control System (CICS) product.* ***The use of Z reduced development costs significantly and improved reliability and quality.*** *Precision is achieved by basing the notation on mathematics, abstraction through data refinement, re-use through modularity and accuracy through the techniques of proof and derivation. CICS is used worldwide by banks, insurance companies, finance houses and airlines etc. who rely on the integrity of the system for their day-to-day business.*

5.3.4 First project details

The CICS/ESA version 3.1 which came out in June 1990 included :

- 500 000 lines of unchanged code,
- 268 000 lines of new and modified code including,
- 37 000 lines 'produced from Z specifications and designs',
- 11 000 lines partially specified in Z,

(2000 pages of formal specifications in total).

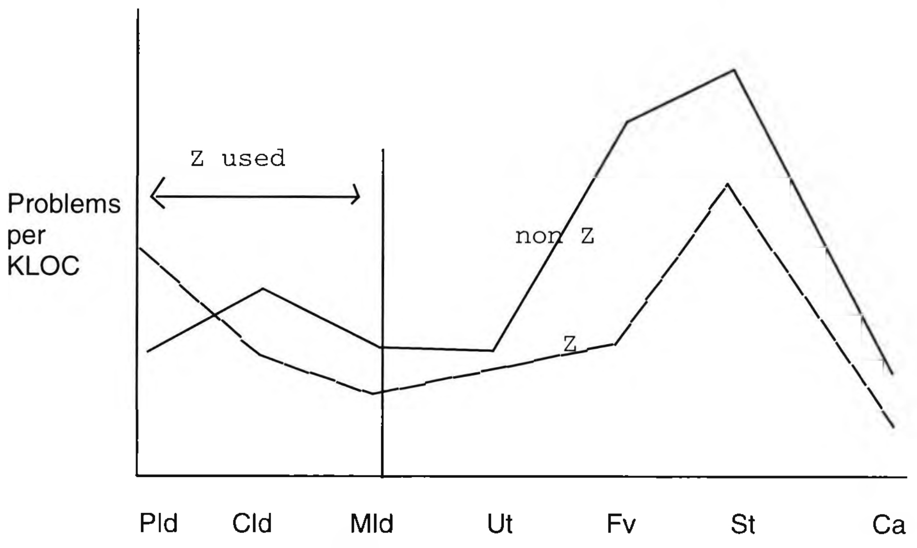
There was only limited use of refinement techniques and very little proof. Specifications and one or two levels of design were written in Z but a notation based on Dijkstra's guarded commands [Dijkstra 1975] was used to express designs and bridge the gap to procedural code. In most cases there was no formal relationship between the stages and noting the preconditions was the only attempt at rigour.

The claims for the benefits of the use of Z in this project seem to be based on the comparison between the development of the code from the specifications where Z has been used and that where no Z was involved (see figure 5.3). This still leaves a number of doubts about exactly what is being compared with what (see 5.3.4.1), but no matter what assumptions are being made it is clear that the 'Z data' is defined on a relatively tiny amount of code compared with the 'non Z data'. It is also clear from the reports that the Z modules were not chosen at random. From an experimental viewpoint these are major (related) problems that could have been easily averted.

5.3.4.1 Claim 1: Fewer problems overall during development

Figure 5.3 (which is copied from the only graph given in the literature) is a comparison of the reported 'problems per KLOC' (Thousands of Lines of Code) at each key phase of the project development. These phases are defined as part of the standard IBM development life cycle. [Phillips 1989] drew the graph up to the system test phase, but [Houston and King 1991] added the Customer availability data (8 months after release).

The problem rate appears to be larger in the early stages of development with the code derived from Z specifications but in the later stages of the life cycle of the software it has fewer errors. This is consistent with the findings when the effectiveness of formal methods were analysed on our third case study as part of the SMARTIE project [Pfleeger et al 1995]. It is proposed that the reason for this is that by using Z, specifiers are forced by the rigour of the notation to tackle all the complexities of the problem and therefore make a larger proportion of their mistakes at this stage. In contrast the non-Z users have developed code which has errors in it right up to the final stages of the process when fixing errors will be more expensive.



KEY	Pld	Product level design	Ut	Unit Test
	Cld	Component level Design	Fv	Function Verification
	Mld	Module level design	St	System test
	Ca	Customer Availability		

Figure 5.3 Reported comparisons between code resulting from Z and non-Z methods

The most serious concerns about the validity of this claim are:

- The omission of a scale on the vertical axis of the graph in Figure 5.3.
- From the text there is a lack of clarity, any of the following could be interpreted as the basis for comparison:

  37K lines with Z : (268- 37)K lines without Z,

  37K lines with Z: (268 - 37 – 11)K lines without full Z,

  37K lines with Z: (268 + 500) K lines without Z.

  [Houston and King 1991] assert that
  Since similar measurements were made on the non-Z code on this release, and on all the code in previous releases, meaningful comparisons are possible.

From this assertion it is our understanding that the most likely comparison being made is between 37 KLOC with Z and 768 KLOC without Z but the public papers do not make this clear. This basis for comparison is the most worrying scenario since the non-Z code is heavily biased with old code which has problem reports dating back many years whose impact on the study is not made clear. In particular the number of post release problems for such code could well be higher but these cannot have been included in the Z code from the timescale involved in the graph.

- The fact that we do not know what proportion of the Z specification relates to new modules (as opposed to just changed) is really crucial. Where an existing module is being re-specified we would expect to see a significant drop in problem reports compared to both the previous version and the baseline. This is true irrespective of the specification method used. After all, IBM has many years of experience with the existing code and has extensive knowledge of where the problems lie; this is a major rationale for making changes. If, as seems likely, the proportion of changed code in the Z specified portion is significantly different from the proportion of changed code in the non-Z portion, then this is further evidence that the problem density data between the Z and non-Z code cannot be meaningfully compared.
- As any involvement from Z stops after the 3rd stage (module level design), it is not clear what effect Dijkstra's language had and whether it was used on the non-Z code.
- It is not clear if the two types of code were subjected to the same regime of inspections and testing (it is highly unlikely this would be possible given the very different nature of the documents). Consequently it is not clear if the class of problems being reported (especially prior to Ut-Unit Test) are comparable. Nor do we know how problems were counted. If an error originated in a schema was it counted with every inclusion? It is also certain that not all errors are equally serious (there appears to have been no attempt to weight them by fixing time or severity); because of this the absence of a scale in figure 5.3 is an even more serious omission. It is not clear what KLOC means for non-code documents. Every phase prior to Ut in figure 5.3 involves a document which is not code. The most likely explanation is that the KLOC at, say, Pld (product level design), is measured as the KLOC in the code that is eventually implemented from the design. Also, it is not clear if comments are included in KLOC and if this is significantly different in the two types of code.

- The increased supervision of those writing in Z may have been a factor as they were employing a completely new technique.
- The expertise available from PRG on the Z parts may have been a critical factor.

5.3.4.2 Claim 2: Development savings

Of all the claims made about the benefits of using Z on CICS the most impressive and surprising is the well publicised 9% reduction in overall project costs. Specifically, [Houston and King 1991] assert:

> *IBM has calculated that there is a reduction in the total development cost of the release. Based on the reduction in programmer days spent fixing problems, they estimate a 9% reduction as compared to developing the 37,000 lines without Z specifications.*

The 'problems' on which the figure is based are restricted to those discovered during development; we believe that **they do not include post-release user reported problems**. [Houston and King 1991] do not state this, but a very similar quote using the 9% figure appears in [Phillips 1989] which was published before the new release (in fact Phillips' version of figure 5.3 only goes as far as system test).

The concerns we have already expressed about the basic comparison between Z and non-Z derived code make the 9% figure already seem less convincing. However, we are also concerned about how the figure was calculated. The second sentence in the [Houston and King 1991] quote is as much detail as is presented in all the papers about this, so we can only speculate on what it really means. There is a suggestion that the extra time spent on doing the Z specifications (not to mention training etc.) is not accounted for. This is a gross omission. We are also concerned about the word 'estimation'. It suggests that no actual data on time to fix the problems was recorded. Rather it appears that an IBM standard 'cost to fix a problem' was used. Let this cost be c. Now suppose the problem density of the portion of code derived from the Z specification was x KLOC and the problem density of the portion of code derived from the non-Z specification was y KLOC. Then it appears that they have computed

$$c*37*(y-x)$$

and that this comes to 9% of the actual total cost of the release.

It is still very unclear from the literature how much comparative measurement went on and what statistics were collected.

5.3.4.3 Claim 3: Customer benefits

[Houston and King 1991] assert:

> *... the figures on number of problems reported by customers are extremely encouraging: in the first 8 months after the release was made available, the code which was specified in Z seems to have approximately 2.5 times fewer problems than the code which was not specified in Z. These figures are even more encouraging when it is realised that the overall number of problems reported is much lower than on previous releases. There is also evidence to show that the severity of the problems for code specified in Z is much lower than for the other problems.*

There are actually three claims being made here that we address in turn:

1. *That the Z code has 2.5 times fewer problems than the non-Z code.* The 2.5 figure is presumably the differential shown in the last phase of the graph of figure 5.3. The authors do warn that

   > *the length of time and the size of the sample mean that figures available so far should be treated with some caution... that many customers do not change immediately to a new release when it is made available.*

This is a very obvious drawback to the 2.5 claim. We have already noted that the non-Z code contains a very high proportion of old and well used code, which would inevitably attract more problem reports. All of the Z code is either new (hence contains new functionality not yet much used on IBM's own admission) or changed (hence inevitably reducing the number of known problems). It is therefore our view that the 2.5 figure should be treated with more than just 'some caution'.

2. *The overall number of problems reported is much lower than on previous releases.* In fact it is not clear what is being compared. The problem reports for the new release only cover the first 8 months; the problem reports for previous releases cover their entire lifetime (in each previous release this is at least two years). It is therefore inevitable that the number of problems reported here is much lower. However, if the comparison is

genuine (that is if the comparison is with the number of problems reported in previous releases restricted to their first 8 months), then it is strange that IBM have not provided details of the data. There seem to be no subsequent reports to describe what happened after the first 8 months

3. *The severity of the problems for code specified in Z is much lower than for the other problems* Unfortunately, the 'evidence' is not presented at all. For example, we do not know if this is based on a subjective classification of the problems or on something more scientific like the relative time to fix the problems.

5.3.5 Summary

The CICS experience is widely regarded as the most significant application of formal methods to an industrially sized problem. From their original questions the two most relevant to the use of metrics with Formal Methods are

- could aspects of large and complex systems be captured in mathematics?
- would there be practical benefits?

There could be further investigation done with regard to the first question if measurements were available on the characteristics of the parts of the system chosen for specification in Z. Did the attributes of the specification itself reflect these characteristics and did that mean that these parts were particularly well chosen? For example it may have been the case that the portion chosen for specification in Z was suitable for modularisation and that this could have been demonstrated to be a strong aspect of the resulting specification with suitable metrics. From the second question the practical benefits could also have been supported by the use of metrics on the Z specification itself and more detailed measurements from the resultant code.

The claims made about the effectiveness of using Z on this project are highly impressive and often quoted. It is therefore essential that the claims are substantiated with rigorous quantitative evidence. We have found that the public domain articles do not provide such evidence. We believe the following are needed if any firm conclusions are to be drawn:

- an update with the results for the 1990 release under further customer experience,
- clarification of the basis under which the measurements were made,

- more details of the analysis of the comparative statistics mentioned in the papers.

Until these points are cleared up it remains a valuable case study but a flawed one.

5.4 CASE STUDY 3 CDIS : Central Control Function Display Information System

This project is of interest because it has been subject to much more rigorous measurement and analysis than the previous two. The CDIS system was developed and built using Formal Methods by Praxis under Anthony Hall and their use was monitored as the project progressed. Once the system was complete all records, including the data collected during the development and the post delivery fault reports, were made available to the team undertaking the SMARTIE (Standards and Methods Assessment using Rigorous Techniques in Industrial Environments) project. The motivation for this unusual co-operation came from Praxis. They have always been supporters and users of Formal Methods but they also wanted to know how the use of formal methods had affected the final code.

Much of the information here is derived from the findings of the SMARTIE project [Pfleeger et al 1995] and from the papers on the original project by Hall [Hall 1996] and Pfleeger and Hatton [Pfleeger and Hatton 1997].

5.4.1 Background

The UK is currently in the process of upgrading its air traffic management system. The Central Control Function (CCF) which provides automated support for controllers in London Area Terminal Control Centre (LATCC) is part of this system and its development is part of the upgrading process. This support is handled by several systems:

- an upgraded National Airspace System,
- new Radar System,
- closed Circuit Television,
- a new information system CCF Display Information System, CDIS.

Praxis was responsible for developing this last feature and delivered it to the Civil Aviation Authority (CAA) in 1992 and it subsequently went into operational use in Autumn 1993. In its final form it amounted to nearly 200,000 non comment, non blank lines of C code and the specification and design documents ran to 1200 and 2000 pages respectively. The total effort on the project was 15536 person days of which 270 were spent on requirements, 1274 on specification and 1556 on design.

5.4.2 What does CDIS do?

CDIS is the support system responsible for

- displaying information about departing and arriving flights,
- giving information about weather conditions,
- reporting equipment status at airports,
- displaying support information input by CDIS staff

Its output is displayed on a monitor in the form of 'pages' of information which can be scrolled through. Its data is received from three sources:

- the National Airspace System,
- the Airport Data Information System,
- closed circuit television through the CDIS Central Processing System which also acts in a failure management role.

Because of its safety critical nature information must be displayed quickly and accurately and the system must be operational 99.97% of the time.

5.4.3 Who uses it?

In LATCC there are about 30 controller workstations displaying the CDIS information along with the other support systems including radar and flight progress.

As well as these there are 20 simpler machines and 6 administrative workstations used by supervisors, engineers and data entry staff. One of the controller's workstations is shown in figure 5.4. The main CDIS display is on the screen marked 1. and 4. shows the device for scrolling through the pages of the display.

1. CDIS electronic display device
2. Closed Circuit television
3.Radar
4. Page selection device
5.Computer entry and readout device
6. Flight progress strip rack

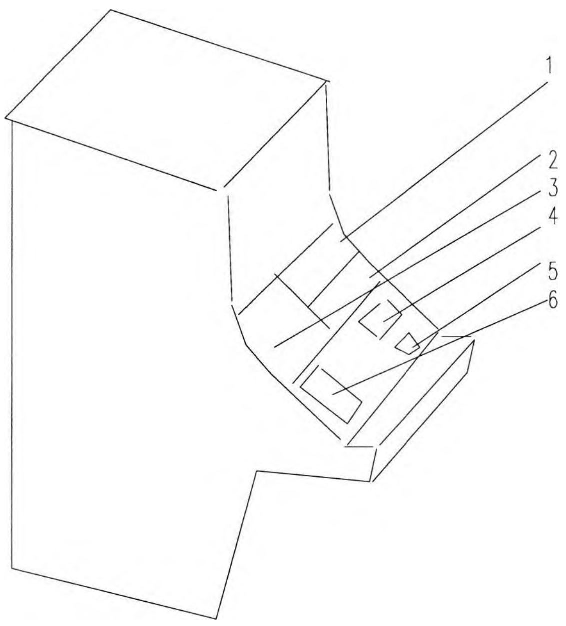


Figure 5.4 the CCF controller's workstation

There were two main areas of this project where formal methods were used - the system specification and the system design.

5.4.4 How was it specified?

The functional requirements of the system were developed using three techniques:

- entity relation diagrams to look at real world object (arrivals, displays) and the relationships between them,
- a real-time extension of Yourdon-Constantine structured analysis, using dataflow diagrams (DFD) to define the processing requirements,
- VDM to define the data being held by CDIS and the operations on this data,

5.4.4.1 First thoughts

Initially the idea had been to use DFD at the top level to define systems operations and bring in VDM to define the lower levels from these diagrams. This proved unsatisfactory as the lowest level got too near a design rather than a specification and the DFD did not clarify the meanings of the dataflows (Hall's string of beads problem again).

5.4.4.2 Second thoughts

It was decided to do a complete top level specification in VDM. So a VDM state was defined that represented the whole CDIS system and events in the structured model were transformed into VDM operations. Using VDM was intended to have the effect of clarifying understanding of the requirements and making them complete and unambiguous. However, although using it did force the team to a point where they had a good understanding of the system, there were several drawbacks to VDM:

- it did not capture global features like properties that had to hold for *all* operations,
- it only captured *functional* requirements not aspects of performance,
- it did not help in specifying user interface details,
- it did not deal with the concurrency issues.

5.4.4.3 Final solution

The development team finally used different formal methods to specify different aspects of the system. They produced a system specification in three parts - namely a core specification, user interface definitions and a concurrency specification.

Core specification

This was specified in VVSL which is a notation with a VDM type syntax [Middleburg 1989]. VDM was rejected because of its shortcomings in modularisation and Z rejected because of its clumsy error handling and the gap between its style and VDM. In fact VVSL had to be extended to handle combinations of operations.

User interface specification

This was specified with two types of information. The physical appearance was modelled using pictures and text whereas the syntax of the user dialogues was specified with state transition diagrams.

Concurrency specification

This was specified in two parts using CSP [Hoare 1985] for the processes and VVSL for the alphabet of the traces. Modified data flow diagrams were used to show the processes and the state variables they read and wrote to.

5.4.5 Lessons learnt

The specifications provided a good description of the functionality of the system and were a solid foundation on which to base the design. In writing them and reviewing the functionality with the CAA the requirements were clarified precisely. However there were problems:

- once the three specifications were finished there were difficulties in getting an overview of the system,
- problems arose in reading the core specification which were attributed to the lack of natural language comments,
- the nature of the threefold split caused boundary problems between them,
- separation of concerns was not as good as it might have been leading to changes in one area impacting on others,
- the level of detail was difficult to choose between this and the design phase,
- some of the assumptions and abstractions in the specification made it a crude approximation to the real operation of the system.

Once completed the specification was used for a variety of purposes:

- managing the change control,
- as a basis for design and implementation,
- to derive system tests (used in preference to the original functional requirements),
- as a basis for user documentation.

5.4.6 How was it designed?

We have seen that the system requirements specification had three different views of the same system. When it came to the design phase there was a split into four different parts of the software. Hall summarises the design components and their relationship by a diagram similar to figure 5.5.

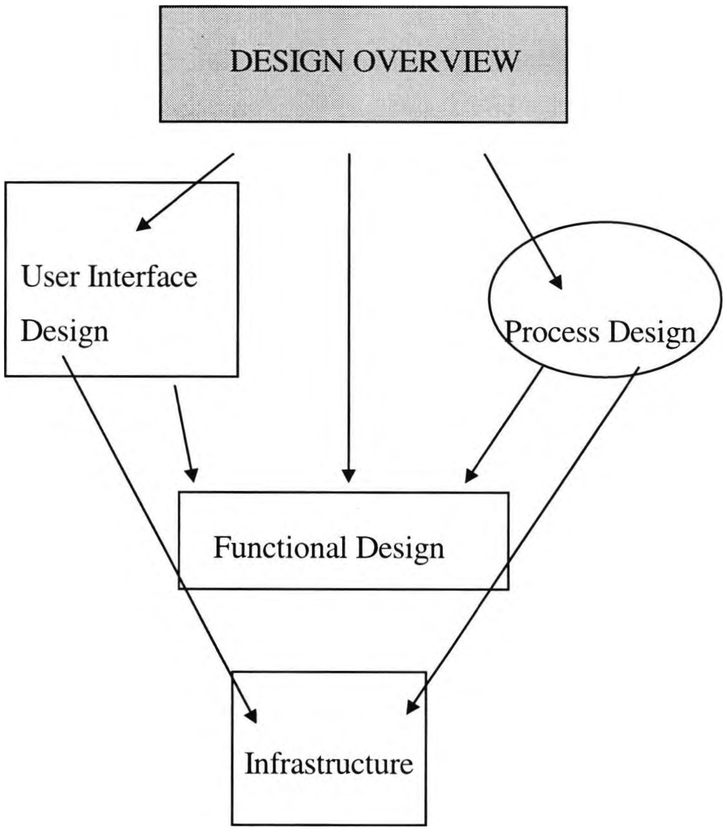


Figure 5.5 The design components of CDIS

The use of Formal Methods was proposed in three out of four of the design areas;

- functional design; the application modules were intended to be specified in VVSL,
- process design; from the concurrency specification the processes were modelled using finite state machines and VVSL,
- the infrastructure; the LAN software requirements using VVSL.

The user interface design was implemented using IBM Presentation Manager so the specification of this design followed the style of this particular package.

Functional design

The module structure of the applications modules was derived from the core specification. Each module was layered into an operational layer (used by processes or user interface) and a services layer (used by other application modules). Each module was divided into three parts for CDIS Central Processing System (CCPS) specific processing, Workstation processing and common code. Originally the plan for the more critical modules was to refine the VSSL used in the corresponding specification modules, writing down the refinement relations and proof obligations. For the less critical modules more code-oriented specifications were planned.

In the event all modules were specified using VVSL (according to Hall or VDM according to Pfleeger) and there was no attempt to formally connect the specification and design stages. This was partly because although there was refinement of the data, there was no obvious correspondence between the operations on the specification module and the application module. There was a team of 10 developers used on the application modules.

Process design

This was derived from the concurrency specification. DFD's were used to document the processes and their communication, Finite state machines diagrammed the process designs and VSSL predicates characterised the complex states and the actions in the finite state machines.

On reflection it was felt it might have been better to encode the FSM in VSSL

Infrastructure design

This mainly involved the LAN software in three respects:

- definition of the LAN protocol,
- the interface between LAN and the rest of CDIS,
- the design of the LAN software.

As we noted VSSL did not suit the requirements specification of concurrency and CSP was used instead but for the later parts of the requirements specification CCS [Milner 1989] was preferred. They translated straight from the CCS design into code but did attempt some proofs of correctness. In trying the proofs they detected a concurrency problem that would have otherwise not shown up even by testing.

5.4.7 Lessons learnt

The main benefits of the design were:

- the functional design produced few faults attributable to incorrect mapping between specification and design,

- the application design gave clients an exact description of the features they could expect from the module interfaces,
- implementation was straightforward from code to design,
- the CCS design of the infrastructure coped very well with the complexity.

On the other hand drawbacks were:

- again the problem of fragmented design, done in four parts meant that an overview of the design was hard to obtain and there was no unifying method that could draw the four design techniques together or validate the design as a whole,
- refinement broke down when the design structure was very different from the specification structure,
- proofs or formal procedures were not carried out to any great extent.

Overall Hall acknowledges that the use of Formal Methods was combined with good engineering practises to make the project a success. On reflection he felt the addition of good commenting and better structure would have helped when including Formal Methods as part of the specification and design.

5.4.8 Claims and measurement results

Hall claims

> *Using formal methods helped us to build the right system and helped us to build it right - at no extra cost. Our project shows that using formal methods on real large-scale projects is not only practicable but beneficial.*

In analysing the effects of Formal Methods on this project Pfleeger and Hatton summarise the work undertaken as part of the SMARTIE project. Praxis made their records available to the research team and in particular gave them access to all the fault and fix data kept during the development.

5.4.8.1 Pre - delivery analysis

Fault data analysis

To analyse the effect of Formal Methods on the code each final coded module was categorised as being influenced by four design types:

VDM, FSM VDM/CCS or informal.

The faults were also categorised by four severity types:

0	Problem non existent or previously reported
1	Operational System Critical
2	System Inadequate
3	System Unsatisfactory

The researchers used about 3000 fault reports having discarded about 900 reports labelled 0. The period of time covered by these reports was 1990 to June 1992 i.e. they stop at delivery. Other reports were used to compile the post-delivery results. Table 5.1 summarises the differences in modules developed using the different methods concentrating on the changes made to modules brought about by fault reports.

From this data it appears that the claims that a Formal Methods design would produce better quality code are not obviously supported and there was no one method that looked significantly better than the others. However the team went on to undertake further studies.

Timing data analysis

They considered the code changes as a result of fault reports and noted at what quarter of the year they occurred within the 8 quarters of the study. This would measure whether faults are found earlier in the lifecycle when Formal Methods are used (as is often claimed) and look at the pattern of fault reporting over time for the different design methods.

The results were inconclusive although each type of code seemed to reach a peak of changes late around the onset of system testing. The VDM/CCS code seemed to have its peak at an earlier point than the others but that may be due to other factors.

	FSM	VDM	VDM/CCS	Total formal	Informal
Total lines of delivered code	19 064	61 061	22 201	10 2326	78 278
Number of changes in delivered code caused by fault reports	260	1539	202	2001	1644
Code changes per KLOC	13.6	25.2	9.1	19.6	21.0
Number of modules of this design type	67	352	82	501	469
Total number of modules changed	52	284	57	393	335
% of delivered modules changed	78	81	70	78	71

Table 5.1 Relationship of changes in code to design type

Code audit

Using automated tools the code was analysed in two ways:

- looking for remaining faults in the modules (categorised into 6 types),
- calculating several structure and dependency measures (cyclomatic number, static path count, nesting levels etc.) for comparison with similar standard C software.

The results did not prove very useful for the first analysis but the team noticed some unusual features from the second. CDIS had an unusually low proportion of units with high complexity. The modules have simple design and loose coupling and because this was true for all code regardless of the design method it was concluded that the *simplicity seemed likely to be a direct legacy of a formal specification.*

However as with all previous analysis the researchers could not be sure that other factors might not have led to this simplicity of code.

Unit testing

The thorough testing on this project was due in part to a contractual obligation for Praxis to carry out 100% statement coverage. They used a variety of software, unit testing the VDM and FSM code using Softest, the VDM/CCS code end to end and the informal code with a test harness. The number of faults and the number of modules within that design type are given in the first 2 rows of table 5.2 resulting in a normalised figure for comparison.

	FSM	VDM	VDM/CCS	Total formal	Informal
number of faults	43	184	11	238	487
number of modules in type	77	352	83	512	692
faults/module	0.56	0.52	0.13	0.46	0.70

Table 5.2 Comparison of the normalised unit testing fault analysis.

Faults discovered at this stage clearly occur more often in informally designed modules.

Taking the distribution of pre-delivery faults over the three phases of testing as

- 340 during code review,
- 725 in unit testing,
- 2200 during system and acceptance testing,

an unusual pattern for the distribution of faults was noted. Normally more faults are found in the code review rather than the unit testing but again the reasons for a reversal of this pattern were not clear.

The delivered code was certainly less fault prone after systems testing with only 273 problems recorded post delivery (up to June 1994).

5.4.8.2 Post - delivery analysis

The results presented so far were collected from the pre-delivery data but now post-delivery changes were recorded and compared. These covered a period from the 1992 delivery and the end of the data used by the SMARTIE study in June 1994. There were 185 changes to delivered code broken down as shown in table 5.3.

Design Type	Number of Changes	Lines of code with this design	Changes normalised by KLOC
FSM	6	19 064	0.31
VDM/CCS	9	61 061	0.72
VDM	44	22 201	0.41
Total formal	59	103 326	0.58
Informal	126	78 278	1.61

Table 5.3 Comparisons of the changes to delivered code.

We can see from these comparisons that we are led to the conclusion that formally designed modules were much more reliable post delivery. The failures per KLOC for the formally designed code were a remarkable 0.58. In comparison with other reported project figures given by Hatton, which range from 1.4 for a Lloyd's language parser to 30 for an IBM standard project, this was an excellent result and partly due to very thorough testing pre-delivery [Hatton 1995].

5.4.9 Summary

A lot of the results from the analysis of this project seemed inconclusive. There were several other factors including:

- the sizes of the development teams,
- the overall 'culture' of development within Praxis,
- the nature of the problem domain,
- the concentration of the formal design in the critical modules which were then given to the best developers,
- the use of the informal design for the user interface where faults are noticed early.

These could all have affected the quality of the software produced and insufficient background data on these factors meant it was not possible to eliminate the bias they may have caused.

The main conclusions seem to be:

- there was no quantitative evidence to support the hypothesis that formal design techniques *alone* will produce higher quality code than informal ones,
- no one formal method used here is significantly different from another,
- formal specification led to simpler structure which was easier to test,
- high reliability can be obtained by using formal specification, static inspection and good testing but all three are needed.

5.5 CASE STUDY 4 BASE : Trusted Gateway

This was a project carried out at British Aerospace Systems and Equipment Ltd and reported by Larson, Fitzgerald, Brookes and Green [Fitzgerald et al 1994, 1995, Larson et al 1996].

Its purpose was to study the effect of introducing a modest amount of formal specification into an existing development process. Funding became available from the European Software Systems Initiative (ESSI) for a comparative study to look at development with and without formal specification. It was expected that information about costs and benefits could be gained from the results. For the comparison, parallel development took place so that two separate teams worked on this same problem. There was to be no formal proof so that the emphasis would be on system and software modelling.

5.5.1 The problem

The task was to develop a small but security- critical message handling device called a *trusted gateway* (see figure 5.6).

This device prevents accidental disclosure of classified or sensitive information. It is used when two information systems are connected to control the information flow between them and to filter the stream of messages. Each message is read as a stream of characters and the first test checks to see if it is a valid message and if not outputs it through the error port. A second check on those messages which are valid looks for the presence of a *category* string to match against those held in two lists within the trusted gateway. These category strings within a message will indicate the security classification of the message, high or low, and the message will then be output through the appropriate port.

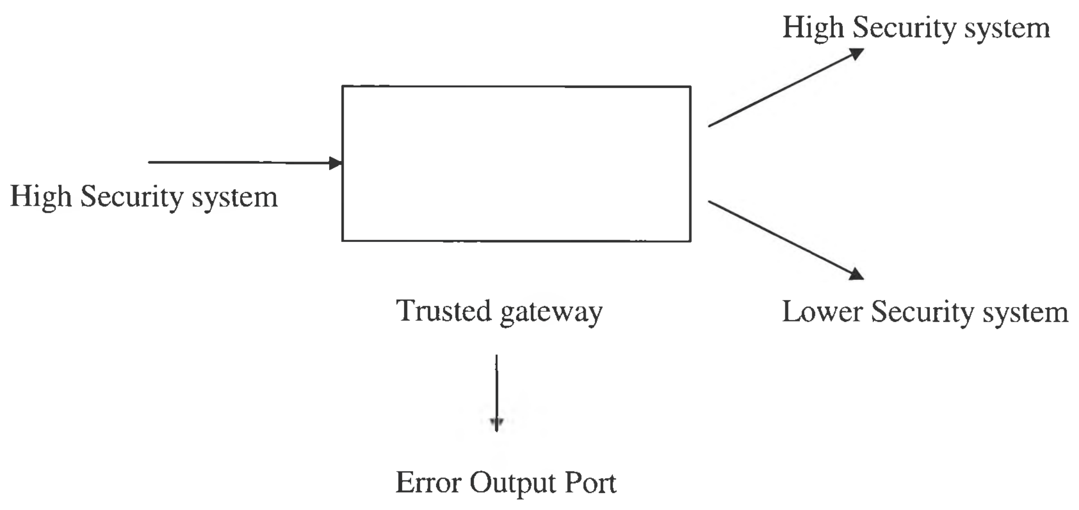


Figure 5.6 The function of a trusted gateway

The system called for a high level of assurance under the Information Technology Security Evaluation Criteria (ITSEC) which defines the methodology and development processes to be used to achieve each level [ITSEC 1991]. The levels go from E1 to E6 and this gateway needed to be developed to level E5.

5.5.2 The specification language

The factors that influenced the choice of specification language were the requirements that it should:

- be a well established notation,
- be accessible to computer literate engineers,
- have extensive support in the wider engineering community,
- be supported by tools.

For its ability to satisfy these four criteria VDM was chosen as:

- from its original form in 1983 it is now in the last stages of standardisation as VDM-SL under the guidance of the International Standards Organisation and the British Standards Institute. [Jones 1983,VDM 1991, Plat and Larson 1992, ISO 1993] so it has clearly a well established notation,

- its model oriented form and straightforward notation make it accessible (although Larson et al note that the engineers preferred not to use the mathematical form of the notation preferring a more verbose ASCII representation particularly when presenting the formal specification to colleagues),
- VDM has been quoted in surveys as one of the most widely used formal notations and as well as the large VDM bibliography users have an electronic mailing list,
- there were a number of tools available and the team used the IFAD (Instituttet For Anvendt Datateknik) VDM-SL Toolbox because it enabled validation of the specification by systematic testing.

5.5.3 The personnel

Two teams were involved in the development of the trusted gateway. One took the *conventional path* using standard BASE methodology of structured analysis (based on Yourdon) with Cadre's Teamwork and RTM (Requirements and Tracability Maintenance) CASE tool support. The other team took the *formal path* with essentially the same design process but adding Formal Methods wherever appropriate and at least in the security enforcing functions in the system. Both were working from the same initial customer specification given in the form of an initial system context diagram and first level data flow diagram.

The two teams were independent and kept apart physically so that they did not communicate about the project. Their background and skills levels were similar but their approach to working was not necessarily the same.

A very intensive one week course on Formal Methods and using the Toolbox was provided for the engineers, managers and quality assurance staff on the formal path and also some of the other BASE engineers so there would be the opportunity for the team to discuss their problems with a wider group.

Due to time constraints the other team were not given formal training on RTM and Teamwork. Informal introduction and extensive support on tool use was provided for both teams. Consultation and 'expert' help was available but in all but the simplest cases hints (not solutions) were given.

5.5.4 Development process

This followed a traditional 'V' lifecycle model where each main development step is accompanied by a test plan against which it will be tested. The development was divided into three phases: systems design, software design and implementation. At the end of each phase a review was carried out which included an inspection of designs and documentation by senior engineers and quality assurance personnel.

5.5.5 Systems analysis

This was carried out by a single engineer on each path who translated the customer requirements into collections of numbered clauses which were then translated by Teamwork into a model of the system. The engineer on the formal path in addition produced a formal specification in VDM-SL of;

- the data types recorded in the Teamwork data dictionary,
- the principal functions recorded in the process specification.

A central monitoring authority took on the customer role and recorded independently the questions and answers against the requirements.

At the end of this stage both teams produced a system specification and a system test plan. The document from each team was reviewed and then the monitoring team carried out a third comparative review (whose results were not given to the developers).

5.5.6 Evaluation

The log of the queries kept by the monitoring team showed that the engineer on the conventional path submitted about 40 questions about the requirements whilst the engineer on the formal path submitted about 60. These queries were categorised by several other engineers into four groups an the percentage of each type is given in table 5.4 (presumably a rounding error gives the total for the formal path >100%)

	Formal Path	Conventional Path
Function What the system does	42%	60%
Data	31%	10%
Exceptions What the system does under certain conditions	14%	8%
Design constraints	14%	22%

Table 5.4 The analysis of requirement queries from each path

This looks like a significant difference in the pattern of results with a much higher proportion of data queries in the path where the data was formally modelled, and a reduction in design constraint queries although this difference is not so marked. The effect of the Formal Method seems to have forced the engineer to clarify the requirements at an early stage while the engineer on the conventional path could leave a number of the assumptions implicit.

The number of queries about the function seem very different but it is interesting that if we convert back to the absolute values (given that the literature states the approximate total number of queries as 60 and 40 for each path) then the numbers are as shown in table5.5

	Formal Path	Conventional Path
Function What the system does	24.6	24
Data	18.6	4
Exceptions What the system does under certain conditions	8.4	3.2
Design constraints	8.4	8.8

Table 5.5 The analysis of requirement queries in absolute values from each path

We now see that both engineers asked about 24 questions about the function which seems logical as you would expect them both to want the same quantity of information at this stage. The figures for data now show how many more queries were asked by the formal path engineer as the information was needed for the extra data modelling. The exceptions queries now become noticeably different whereas the design constraints, like the function queries, have a global nature leading to the same number of queries independent of path.

The comparative review at the end of this phase showed that the formal path engineer raised some crucial queries about the trusted gateway which led to the identification of an exception not noted in the original requirements document. The exception was then incorporated into the formal design but was missed by the conventional path engineer and this omission had a large impact on the project at a later stage. This does seem to support the theory that the rigour of formal methods encourages practitioners to focus on defining exactly what the system does before getting into design details.

This may be particularly relevant in this problem domain as message processing systems are prone to errors caused by data definitions and exception behaviour both of which were the subject of greater scrutiny on the formal path.

5.5.7 Systems design

The main drawback with evaluating this phase of the project was that the engineers assigned to the formal and conventional path had different levels of experience. This fact must affect the validity of most of the comparisons that are made between the two paths.

Most of the queries raised on either path were about the models that were passed to them from the previous phase and not about any underlying assumptions. The engineers for this phase did not question the 'correctness' of what they had been given. This led to the error already noted in the conventional path specification persisting through this phase.

Despite the training in formal methods given to all the engineers at the outset of the project the engineer on the formal path used his normal design process. The literature suggests this could be for one of two reasons:

- he was not trained in techniques needed to refine the specification,
- he was an experienced engineer and did not feel that Formal Methods would result in any improvement to his normal practise.

For whatever reason he 'worked backwards' and wrote in pseudo code first and then translated type definitions and procedural specification of functionality into VDM-SL for testing.

Both engineers produced designs and test plans which were reviewed and compared as before. The formal design was noted to be at quite a high level of abstraction leaving more design decisions to the implementor. Testing the functionality at this stage on the conventional path might have found the error that persisted through this stage. The skills required to turn an abstract specification into a more concrete design are different from those required in the initial specification stage and need different training.

5.5.8 Implementation and code characteristics

It was at this stage as each detailed design in pseudocode was coded into C that the error in the conventional path specification came to light. A common user interface was supplied to each path so that blind testing could be done by the implementation engineers. The test suites produced by each path were used with their own code and then with the code from the other paths. It was at this stage the conventional code failed several of the tests from the formal test suite and the deficiency was spotted.

The authors point out that had the formal path test suite not been used i.e. only conventional development had taken place, then this error would have gone through to the delivered code. The patch that was then written to fix this problem affected the whole structure and efficiency of the code and again made further comparisons difficult. Some that are included are shown in table 5.6.

	LOC	Comments	Initialising Time (sec)	Processing rate (cps)	McCabe	%Budget allocation
formal	63	63	70	250	10	81
conventional	371	82	17	18	10 (74)	87

Table 5.6 Comparisons made on the code between the formal and conventional paths

We can see the code for the formal path looks precise, better structured, slower to initialise but much faster to process messages. The conventional code has a lower ratio of commenting and does worse on every measure except initialising time. However, as has been mentioned previously, the last minute patch must have affected a great number of these characteristics and it is interesting to note that the only time that 'before and after patch' numbers are quoted, with the McCabe complexity measures, then the conventional code before patching has a complexity of 10 (the same as the formal) whereas after it shoots up to 74 showing the impact of putting in a last minute code adjustment.

5.5.9 Summary

The small size of this trial in terms of the personnel involved and the size of the final code produced means that generalisations about the benefits of incorporating formal methods into larger projects cannot be made. However it was a useful study because of the parallel development enabling a direct comparison between formal and conventional development.

The results have to be interpreted carefully because other factors such as the difference in experience of the systems developers, or the effect of the software patch, can add bias.

It seems that the impact of using formal methods was:

- the rigorous semantics gave the required level of assurance to this security critical software,
- the specification process improved understanding of the system and identified errors,
- no cost or time overheads were incurred by using formal methods (if training is excluded).

The summaries contained in sections 5.6 - 5.9 are the results of searches for uses of Formal Methods in practise.

5.6 Further case studies of Formal Methods (no measurements given)

In each of these studies no useful metrics were given although it may be that they were collected in some cases.

5.6.1 Formal Requirements Analysis of an Avionics Control System

The re-specification of an existing system showing tools use and giving impressions of the benefits gained [Dutertre and Stavridou 1997]. This was new ground to a certain extent as Formal Methods were applied here to real-time systems and PVS was used. This is a tool supported analysis method concentrating on proof and verification, aspects often ignored in favour of the specification itself. However no measurements were taken of benefits or drawbacks and no details are given of the size of the project except it was declared to be large scale and that the work took 18 months to complete instead of the estimated 6 months. Work had already been undertaken in VDM and new problems were not anticipated. It was felt to be of benefit but the proof process was expensive and difficult to estimate.

5.6.2 The specification of part of a marine diesel engine monitoring system

This was a specification exercise using an early version of B as an example of this new Formal Method [Storey 1992]. The full version of the engine monitoring system was already being specified in RAISE. The paper concentrates on the features of B and the B tool.

5.6.3 Viper and INMOS

These case studies were included as examples in a paper which discusses the value of Formal Methods as a verification for hardware design and their possible transfer to industrial use. [Stavridou 1994]. The formal verification work on Viper (Verifiable Integrated Processor for Enhanced Reliability) was a design quality assurance exercise and proved very expensive. No measurements were taken and the exercise was not repeated with Viper 2. The INMOS work built on the earlier specification of the T800 floating point unit by Shepherd and May, which claimed a 12 month reduction in testing time [May and Shepherd 1987, Shepherd 1990] and is a report of May's findings for the T9000 version of the transputer as far as 1992. It gives no measurements but states the problem of compatibility with the earlier versions, unifying different notations and trying to find a 'window of opportunity' to apply Formal Methods.

5.6.4 Darlington Nuclear Power Generating Station

This work summarised by Parnas was essentially a formal specification arising out of a code inspection process [Parnas 1995]. A team of 60 took a year to conduct the inspection and tabular formal notation was used. The code had previously been thoroughly tested for many years and the main product of the inspection was confidence. The conclusion appears to be that Ontario Hydro, AECL and AECB, the three organisations involved, seemed to be sufficiently happy with appraisal that they would all continue to use Formal Methods in some way.

5.6.5 MSMIE at Sizewell B

The Multiprocessor Shared Memory Information Exchange (MSMIE) is a protocol for communications between processors in a real time system [Bruns and Anderson 1995].

There had already been code verification for Sizewell B using MALPAS [Ward 1993] which took about 200 person years. They now used CCS to construct two models for a subsystem. The first modelled hardware components as CCS agents and the second focused on the possible transitions between states. These were compared using behavioural equivalencies. The authors draw out comparisons of their requirements-driven specification to the process-driven specification carried out in MALPAS. All judgements appear to be subjective.

5.6.6 The Attitude Monitor

This project used a combination of Z and tabular notation [Coombes et al 1995]. It was set up to answer questions posed by British Aerospace about the necessary and sufficient set of requirements against which a design can be verified. Formal Methods were used on an element of the gust alleviation filter to try and answer some of these questions. Partial answers were obtained and the authors point to more work needed in this area.

5.6.7 Application of the B method to CICS

This follows the earlier work we reported as the second main case study, where Z was used. Part of a new component was specified and the B Tool adapted to the task. As yet no data is available to analyse the success of the project [Hoare 1995, Hoare et al 1996]. It was claimed to be successful in capturing new functional requirements but less successful at meeting the non-functional ones.

5.6.8 Formal Verification of AAMP5 Microprocessor

Specification was carried out in PVS using a theorem prover to check on correct implementation [Srivas and Miller 1995]. Costs were noted as significant, no figures were given and they were dismissed as normal for a new technology, acceptance on the part of the engineers seemed to be slow. The benefits seemed to be increased confidence.

5.6.9 CombiCom- the Rail Traffic Tracking System

CombiCom (Combined Transport Communications System) is a distributed System capable of tracking and tracing rail traffic across Europe. It was designed using VDM and VDM++ and implemented in Ada [Dürr et al 1995] One of the motivations of Cap Volmac in undertaking this approach was to unify and co-ordinate the approach of 7 different companies operating on several different traffic systems. A prototype was produced without any notion of distribution. Much resistance on the part of the partners to any reading formal specifications led to the use of graphical representation, informal descriptions and demonstration prototypes instead.

5.6.10 A330/A340 CIDS Cabin Communication System

This was a Z specification applied with the support of CASE tools to try and understand the complexity of the airborne computer system which controls such functions as the cabin PA system, the emergency signs and the cabin illumination [Hamer and Peleska 1995]. The benefits were felt to be a reduction in the complexity, the uncovering of logical errors, reduced effort in the design and implementation and the identification of test cases. No data is given about the claimed reduction in effort or costs.

5.6.11 AT&T Switching System

This was a small specification with only 25 pages of Z but is built on the work by Mataga Zave and Jackson on the multiparadigm approach [Zave and Mataga 1993, Mataga and Zave 1993, Zave and Jackson 1994, Mataga and Zave 1995].

5.6.12 The Design of a Human Computer Interface

This was work done by Jacob as part of the Military Message System [Jacob 1986]. State charts and Backus - Naur Form were used and the two representations compared.

5.6.13 The Electricity Meter

This was part of the EUROTRI project to develop a static electricity meter [Arnold et al 1996]. Formal methods were used to conceive design develop and test embedded software in a mass-produced device. The rationales for including Formal Methods were the short time to development and the low cost of production.

5.6.14 An application of TRIO

The TRIO formal specification notation is an extension of temporal logic. Here Mandrioli describes TRIO and gives the results of applying it to an industrial application. Important "lessons learnt" are given [Mandrioli 1996].

5.6.15 A Banking application

The banking system in question had been written in Cobol and this was an attempt to renovate the code using algebraic specification [van den Brand et al 1996].

5.6.16 Using AMN in a GKS case study

From the starting point of the draft Graphics Kernel System (GKS) given in a Z style specification, a re-engineering approach was used with Abstract Machine Notation (AMN) and the B-Toolkit [Ritchie et al 1994]. Refinement on the specification was attempted.

5.7 Further case studies of Formal Methods (some measurements given)

In these studies some data is given although not always relevant to the issue of metrics and Formal Methods.

5.7.1 The Groupe Bull's Flowbus

This was an application of both B and VDM to specify, design and implement an administrative sub-system of the second release of Bull's product. Metrics for quality and effort were collected and compared with historic data from 2 other similar projects.
[Dick and Woods 1997]. The comparisons are weakened by the use of estimates to cover the way the B Tool has generated code and the lack of detail on the nature of the faults. Some questions are left about the expertise used and the size of the project.

5.7.2 Railway Signalling

This work carried out by GEC Alathom was on the formal development of safety critical systems in railway signalling [Dehbonei and Mejia 1995]. The B Method was used
and this was the largest application they had used it for. The resulting program was about 35 KLOC in Ada, with the preliminary design taking 6 months with 3 engineers and the detailed design stage 7 months with 2 engineers. Cost and planning estimations were too small and the proofs were not completed because the program could not be delayed further. 13 500 proof obligations were generated by the B Tool but only 90% of those for the preliminary design were discharges and 50% of those associated with detailed design. The proof mechanism was difficult to manage and a better user interface will be designed. No further measurements are given and the benefits are thought to be modular maintainable code although no supporting evidence is given.

5.7.3 Crane protection system

This was a pilot study into techniques for developing safety critical software carried out by Rolls Royce for the Ministry of Defence [Hamilton 1995]. It involved the control and protection system for a crane handling hazardous material. Z was used supported by CADiZ, the tool developed by York University [York Software Engineering 1993].

The successes were felt to be the demonstration that it could be done, the highlighting of some deficiencies in the defence standards and the provision of a prototype. The drawbacks were the usual problems of front loading in effort. The results given include low error rates and no faults in the software in testing. Translation into Ada was done manually.

5.7.4 French Population Census

This work shows B Method applied to the geographical data part of the survey [Bernard P and Lafitte G 1995]. This was a real life example but the reference gives a simplified version. The system was developed using Pascal as the target language and the specification resulted in 65 pages of B achieved in 2 weeks with 2 men. The refinement and implementation took 2 months and resulted in 14000 lines of Pascal. No time allowance was made for training or familiarising with the B tool. The authors comment on the need for a background in mathematics from all members of the project team. The system was implemented in 1990 and is still in use. Benefits were felt to be clarity, speed and reliability.

5.7.5 NASA Case Studies

In their book written as a guidebook on specification and verification, NASA gives details of 6 case studies with measurements relating to the goals for each and the cost in terms of time spent [Covington 1995]. The benefits stated in each case seem to be in terms of clarifying requirements of finding omissions and some numbers are given for the errors found.

5.7.6 FME Applications Database

At their web sites Formal Methods Europe give a list of Industrial users of Formal Methods but only manage to find 2 well documented cases [FME a] and also several case studies which are otherwise unpublished [FME b]. They classify the studies by the size of the specification and it seems that those in the category 10 000 to 50 000 lines of specification are all checkers, tools and code generators for different specification notations. In the category 5 000 -10 000 lines the following 8 projects are mentioned.

1. B27 Traffic Control System

   No conclusions about this specification in Z given except to comment on the underestimation of the effort involved. No work published as yet.

2. NewCoRe

   This was a project undertaken by AT&T between 1990 and 1992 as a feasibility study for applying SDL. Published work mainly by Holzman. [Holzman 1994] Some statistics are given about the length of the specification and the time taken and errors found.

3. ELSA (control system of a power plant)

   TRIO was used to specify the control system that balances the load on various generators [Basso et al 1995]. Validation by means of simulation and some comparisons were made with similar projects which had used more traditional methods. They claimed that costs were slightly reduced overall.

4. The ITSEC-E4 Secure Gateway

   The security policy for the gateway was modelled using B. The number of proof obligations was about 500 [Bieber 1996] It was commented that elaborating the security model was more time-consuming than writing the formal specification.

5. T2: The global Second Level Trigger

   A Specification in VDM++ and implemented in Modism II. [Balke 1995] They found no problems in using VDM++ and found it an asset in analysing and developing model solutions.

6. Specification of Tracking Manager Architecture

   An application of VDM to the monitoring and controlling of the movement of materials through a processing plant. Some reservations were expressed about the difficulty of specifying the real-time aspects of some of the operations. Details of the architecture and proof can be found in [Fitzgerald 1996, Fitzgerald and Jones 1998]

7. European Space Agency project

   This was the specification of an instrument control unit in RAISE and LOTOS which was then architectured in HOOD and implemented in Ada. The main aim of the project was to consider the provision of tools to improve ESSDE (European Space Software Development Environment). [ESSDE 1997]

8. Data Management System (DMS) Design Validation (DDV)

   Work done for the European Space Agency to use a provable formal specification for verification and to show that error free transmission could be achieved in a component of a space system, the Fault Detection Isolation and Recovery. SDL was used and the investment in Formal Methods was deemed to have been worthwhile when a design error was uncovered. [DMS]

5.8 Case studies giving comparisons of Formal Methods

These studies either compare different Formal Methods or methods of validation.

5.8.1 An overview of common methods

The paper by Fraser, Kumar and Vaishnavi gives an overview of the most common Formal Methods and distinguishes between direct and transitional ways of incorporating them into the software lifecycle [Fraser et al 1994]. In the transitional method some other devices such as tables or state charts are used as a bridge between the natural language and the formal specification.

5.8.2 The framework approach

In their paper Ardis et al try to establish a framework for comparing 7 different Formal Methods and look at them in relation to the specification of a simple problem of Automatic Protection Switching [Ardis et al 1996]. They give two tables of analysis, one which identifies the importance of the various criteria to the different stages of the software development and the other which gives each Formal Method an evaluation based on these criteria. All judgements are subjective on a 3 point scale (+ = strength, 0 = adequate and - = weakness). Some criteria were judged as not applicable to certain Formal Methods.

5.8.3 The steam boiler problem.

This arose from a competition at an international seminar where researchers were invited by Abrial, Börger and Langmaack to take part in a case study. The World Wide Web was then used to invite further solutions with a deliberate late modification required. From those solutions submitted 21 were collected in a book [Abrial et al 1996]. It was noted at the Z Users conference in 1997 that nearly all specifiers had adapted and simplified the problem so that it would fit into their specification technique. It was also noted that the late modification sent to test the flexibility of the specifications had failed as a strategy because most researchers had left the work so close to the deadline that they had not made much progress or even started on the specification by the time the modification was sent.

5.8.4 The library problem

This was an analysis by Wing [Wing 1988] of the solutions to the library problem which was first posed by Kemmerer in 1981 in the course of his teaching of Formal Methods at the University of California at Santa Barbara. In 1986 a call for solutions to this problem at the Fourth International Workshop on Software Specification and Design resulted in the 12 specifications which are compared here. Wing concentrates on general problems of the capture of requirements.

5.8.5 The Nuclear Waste Tracker and the Trusted Gateway System

In this work Fitzgerald compares the use of Formal Methods and in particular the approaches to validation [Fitzgerald 1996]. He contrasts the use of proof to check the consistency of the specification of the first study taken from the nuclear industry, with the validation by testing undertaken in the second smaller study.

5.9 Conclusions

In Chapter 1 we presented a practical hypothesis in relation to Formal Methods namely

widespread take up of Formal Methods will occur only after the results from large scale case studies are published .

We propose that a careful case study approach can avoid the prohibitive expense of conducting a formal experiment by including the following:

- clear aims,
- a hypothesis to test,
- minimal external factors,
- minimal confounding factors,
- the planning and collecting of relevant measurements,
- analysis and presentation of results.

The four main case studies detailed in this chapter have been an attempt to consider the effect of using formal methods in software development. In the light of these six points we can consider how far they were successful.

The Jacky study was not very precise about its aims except for the need to avoid failures in the installation and running of the cyclotron. Some of the external factors were well controlled by having such a small team involved in the specification and implementation but the uniqueness of development process, controlled for 10 years by one man, was also a confounding factor. The measurements possible in this case would have to be carried out after the event and would consist of metrics applied to the specifications and code. As the installation is not yet complete the post-delivery maintenance and fault data is yet to be collected.

All this adds up to a valiant attempt at the use of formal methods for a real commercial and safety critical situation which cannot be evaluated as yet for lack of quantifiable data.

For all its obvious qualitative benefits, the CICS study only managed to be moderately successful in one or two of the above case study criteria. This partially explains why the claimed quantitative benefits of using Z in this study have not had the expected impact on current industrial practice. In contrast to the previous study we are sure there is more data in terms of fault reports and maintenance records but as yet this has never come into the public domain. The famous 9% will continue to be quoted but, as we have seen, with little hard evidence behind it.

The best of these studies in terms of the six pointers is the CDIS project. It had the benefit of the linkage between Praxis whose interest in Formal Methods is well developed and the SMARTIE project to carry out detailed data collation and analysis. The aims were clear so that appropriate measures were taken and only the effect of confounding factors such as team membership were not fully evaluated. The final analysis seemed to be that the whole project was of high quality and the use of Formal Methods would have been a contributing factor to this but not the only factor. The figure quoted of 0.81 per KLOC for post delivery failures certainly bears comparison with other reported projects. There may have been a significant number of other factors leading to the improvement but the suggestion to come from this work is that it will be of benefit to include Formal Methods where appropriate as one of a range of software engineering techniques. This case study represents the best attempt so far in data collection and analysis for validating the claims of Formal Methods.

The BASE project had very clear aims and methods in trying to provide evidence for the effects of introducing Formal Methods to a project. They did have good methodology and collected meaningful statistics. Unfortunately the small size of the project, which was an advantage in that it allowed parallel development with and without Formal Methods, is such a significant factor that it is not at all certain that the measurements made would scale up to larger scale systems.

As we have seen from these and additional studies mentioned there has been a great deal of time spent trying to apply Formal Methods to a variety of problem domains. Unfortunately, by comparison, little time has been spent collecting quantitative data that might help in evaluating the results.

The metrics which have been used here have largely been concerned with the characteristics of code resulting from a development which has incorporated Formal Methods. We shall now consider ways in which metrics can be developed from the formal specification itself.

CHAPTER SIX

6. EXTRACTING MEASUREMENTS FROM FORMAL SPECIFICATIONS

This chapter looks at work done on the development of metrics for formal specifications. We propose a classification of some of the characteristics of three notations; Z, VDM and ADT. Measurements are shown for small specifications.

6.1 Introduction

We have seen in the previous two chapters some attempts at collecting evidence of the effects of Formal Methods. We have criticised much of this evidence for its qualitative nature which is so easily misinterpreted and for the subjective nature of the judgements which formed the basis for many of the conclusions. The advantage of measurement over opinion is in its quantitative nature and therefore its objectivity.

Some of the case studies mentioned previously attempted to collect metrics from software that had been developed as a result of incorporating Formal Methods into its evolution. Measurements were taken to see if it could be demonstrated that the methods made a difference to software quality. In other studies claims were made about the development process. The most common claims were that the extra time and effort spent using Formal Methods at the beginning of a project, would be offset by the reduction in the need for fixing faults and by the ease of maintenance.

We now consider attributes of the formal specifications themselves rather than qualities of code subsequently produced or the development process involved. Our motivation is to try and establish those characteristics of the formal specifications themselves which influence external attributes which may have a bearing on the software development process and to determine if we can find suitable metrics to capture them. These will then provide possible indicators of the quality of resulting software. We will concentrate on impartial objective measurement rather than notions and feelings.

Most software metrics are attempts at measuring external attributes of software achieved indirectly by considering some internal attribute. In Chapter 3 we looked at some of the existing software metrics to see whether they could be applied to Formal Methods or provide pointers to possible metrics for formal specifications. We also looked at attempts to apply metrics directly to formal specifications and in particular the work of Whitty who tried to use flowgraphs to obtain structural metrics from a Z specification.

There has been very little work done on establishing metrics for formal specifications. Our starting point here is to reflect on the desirable qualities of a specification as perceived by the user, that is the observable external characteristics. In their paper Ardis et al. give a selection of fundamental criteria they felt were important for any specification notation proposed for reactive systems [Ardis et al. 1996]. Those they noted as especially important in the requirements and design phases were stated as:

- applicability - can it describe real world situations and be applied in a compatible way with current technology?
- testability / simulation - whether the specification be used to test the implementation,
- checkability- is the specification readable by domain experts who are not Formal Methods experts?
- level of abstraction / expressibility,
- soundness- whether the semantics are precise so that inconsistencies and ambiguities will be uncovered,
- verifiability - checking formally the links between levels of refinement.

In general the key attributes important to the user relate to:

- understandability,
- consistency,
- accuracy,
- completeness.

In Chapters 7 and 8 we describe experimental work which concentrates on aspects of understandability and which also have a bearing on the Ardis criteria of applicability, checkability and expressability. These would seem to be crucial to the use of Formal

Methods. It is self evident that these methods will not be widely adopted unless users are confident in reading and applying them. There is a lot of anecdotal evidence that formal specifications are part of a 'write only' operation and that only a few specialists can actually read and interpret them.

An alternative approach which may overcome these problems is the increasing use of validation by animation and prototyping. An example of this is given in [Agerholm and Larsen 1997] who describe an animation of the formal specification of an astronaut's backpack. Here users can alter conditions and parameters and see the effects on screen with a small astronaut figure in the manner of a computer game. While this type of validation can be very valuable we believe that the client, or more probably their technical advisors, will want to inspect the actual specification.

In this chapter we consider some key internal attributes of specifications such as:

- conciseness,
- clarity,
- complexity,
- flexibility.

As a means of measuring these internal attributes we find static measurements which can be extracted directly from the specification. We believe these impact on the external attributes such as comprehensibility investigated later in Chapters 7 and 8.

In studies of software metrics the following statistics are often recorded:

- total lines including comments,
- code lines excluding comments,
- total characters,
- lines of comments.

It has been found that both the simple LOC (lines of code) and other more refined metrics such as DSI (delivered source instructions) are not only useful measurements of length, but can also be incorporated into other more complex metrics reflecting measures of complexity, functionality and effort (see Chapter 3). The length of the formal specifications will need to be considered in a similar way with due account taken of commenting and characters. This will have a bearing on its conciseness.

The clarity and complexity of a specification will be affected by the variety of the notation used. Britton et al. in their investigation into measuring notations (not all of them formal in the strict sense) look at their richness by considering the size of the glossaries needed for each one [Britton 1997].

The flexibility of a specification relates to the ease with which it can be modified. One of the drawbacks mentioned in connection with Formal Methods is the poor re-use of specifications and the idea that adding some new information to the requirements or updating the software to a new version will entail creating a completely new formal specification from scratch.

6.2 Previous work on counting methods

The remaining work in this chapter is based on an earlier study carried out by the author and a fuller account of the work can be found in [Finney 1993].

6.2.1 Background

With the four aspects of conciseness, clarity, complexity and flexibility in mind a study had been set up to try and extract suitable measurements from formal specifications to try and reflect these attributes.

6.2.2 Defining the attributes and measurements

As part of this previous study we looked at some counting methods to undertake a comparison of the notations used in VDM, Z and a method using algebraic specification which we call ADT [Finney 1993]. These formal notations were chosen as three of the most popular described in journals and surveys on the use of Formal Methods. The experiments and results are summarised, expanded and extended here.

After some preliminary work to consider the desirable attributes of formal specifications a subset of these was formed. A loose mapping was set up between those attributes that might be of interest and the concrete parts of the specifications which might have a bearing on these. Table 6.1 gives a summary of the mapping between attributes (broken down into sub-attributes) and the measurable characteristics of the specifications.

Overall Attribute	Sub-attributes	Measured by
CLARITY	Readable	number of brackets and non-standard symbols /line
	Ambiguous	no measure was found
	Language	comment lines/total lines
FLEXIBILITY	Modifiable(ent)	number of extra lines for entity
	Modifiable(op)	number of extra lines for operation
	Standard	the number of forms of notation currently in use
	Modular	whether part of the specification could be developed independently
	Size	a count of entities and operations
COMPLEXITY	Abstract	the number of stages to implementation (a subjective judgement not used in the final table)
	Vocabulary(not)	the number of special symbols
	Vocabulary(mat)	the number of mathematical symbols
	Read time	no. of operations x no. of entities x 5 minutes
	Write time	no. of operations x no. of entities x 30 minutes
	Hidden detail	the number of auxiliary functions used
CONCISENESS	Symbols	the reduction in characters over natural language
	Repetition	counting one for each block
	Base types	counting each base type
	Length	total lines of specification

Table 6.1The mapping between attributes and measurable characteristics of specifications [Finney 1993]

6.2.3 Explanations and notes on the attributes and measurements in Table 6.1

6.2.3.1 CLARITY

Readable

This measurement of brackets and symbols was split into two components. The raw scores were counted but the normalised ones were also included for ease of comparison between specifications. It was expected that whilst VDM and Z would have the greatest use of symbols, ADT would have the most use of brackets because of its nested algebraic expressions. Symbols were divided into two types: special and mathematical. The mathematical category covered notation used as a standard part of set theory, mathematics and logic (covered in any introductory course on these topics) whilst the special notation was used for symbols which would only be seen in relation to formal specification or more advanced mathematics.

Ambiguity

This was a difficult measurement to make as it involved subjective judgements of meaning. One approach would be to try and estimate the number of possible interpretations for each statement. Another aspect of ambiguity would be to decide where the source of difficulty was. If a statement involved membership of a set for instance the ambiguity might arise from the poor definition of the base set rather than the interpretation of set membership as described by the notation. It was decided to omit the measure of ambiguity from the study.

Language

In the examples used for this study it should be noted that the specifications were laid out in accordance with versions of Z, VDM and ADT based on [Wordsworth 1992, Jones 1990 and Martin 1986]. Slight variations were sometimes used without loss of meaning due to the limits of the word processors at the time. A new line was started and counted following the standard practices for layout described in these books.

6.2.3.2 FLEXIBILITY

Modifiable (entities)

This is an estimate of the changes needed to the specifications if a modification occurs in part of the data structure rather than in an operation. This would most commonly be an additional item to be considered as an input variable.

From experience the calculation of this attribute developed into the rules;

for VDM	:	total changes = 2 x number of **wr** operations + 1,
for Z	:	total changes = number of modifying schemas,

for ADT : total changes = an extra 2 lines for the more involved conditional statements brought about by more entities to consider.

These rules were tested by considering modifications to stack and queue examples by adding a size limit. This introduced a constant 'max' to be checked. In the later example concerning aircraft boarding the modification involved introducing a luggage component to the boarding process so that both passengers and luggage are checked in.

Modifiable (operations)

The changes to a specification required when adding an extra operation are quite similar for VDM and Z but very different in ADT. In the former two methods all that is required is an extra operation with pre and post conditions for VDM or an extra schema in Z. In the case of ADT, the axioms which form the basis of the specification are formed as combinations of types of functions called constructors and non-constructors. The axioms are given as statements involving each constructor with each non-constructor. As an example if there are 2 constructors and 7 non-constructors then there are 14 axioms. It follows that in this case adding a constructor would increase the axioms by 7 x 1 but adding a non-constructor would only increase the axioms by 2 x 1. So the nature of the operation to be added is significant.

Modular

The modularity and independence of the specifications were intended as measures of how much a specification could be split up as independent parts so that different sections could be developed by different specifiers. This would depend to some extent on the good style of the specification and the overall linking. Efficient management of the passing of common parameters (in the case of VDM) and the correct use of schema calculus (in the case of Z) would affect the modularity. The problem domain could also make choice of modular structure easier in some cases than others. From the size of specification considered here it seemed that ADT did not allow separation of part of the specification, VDM did allow it and with Z it might be possible.

6.2.3.3 COMPLEXITY

Abstract

It had been hoped to try and measure the level of abstraction at which a specification was written by considering the gap between the concepts in the formal specification and their final implementation in code. This proved a difficult measure to define as the reification involved from the specification to code would be attempted by some practitioners in one large step whilst others would want a more cautious iteration towards implementation. In the final table this measurement was therefore omitted.

Read/Write time

These times were based on estimates derived from observing students.

Vocabulary

We should note that at the time this study was carried out the syntax for VDM and Z used had deviations from what would be the present day standard.

Here the notation was divided into:

- specialised vocabulary peculiar to these formal notations,
- more common mathematical symbols.

We attempted to get a measure of how much expertise is necessary to read specifications so using these measurements in a similar way to Britton's glossary. It should then be possible to note the difference between a notation that has a small basic vocabulary to master and one which introduces many new symbols for every part of its construction. Table 6.2 shows our division between the types of notation based on the examples used.

Hidden detail

Considering the auxiliary functions and looking ahead to the more complex problems, a distinction was made between the built in functions such as **max** and **head** and those defined by the specifier because they were not available in the standard notation. The number of functions defined by the specifier was counted. It should be noted that in a rapidly evolving and maturing area such as this, what is user defined today is often built in to the notation tomorrow. Since the initial work was completed, both VDM and Z have had standards or draft standards printed which may help to define the commonly accepted forms of the notation.

Specialised formal notation	Standard mathematical notation
Δ modifiable	∈ is a member of
Ξ read only	⊂ is a subset of
# length of	> greater than
\ without	< less than
∃ there exists	≤ less than or equal to
:: consists of	≥ greater than or equal to
: of type	∨ logical or
∀ for all	∧ logical and
∍ such that	∉ is not a member of
⁀ concatenated with	∪ union with
→ maps to	
' old state of	
::= reduces to	
$\underline{\Delta}$ is defined as	
* sequence of	
⇔ if and only if	
† overrides the domain with	
⊲ overrides	

Table 6.2 The classification of notation types.

6.2.3.4 CONCISENESS

Symbols

Table 6.2, defining the different types of notation used, also shows the note of meaning of each symbol in natural language. This leads us to estimate the overall saving made by the use of the symbols in the specifications rather than words. It will be a large element of the conciseness of a specification that a single symbol can represent a longer natural language phrase, for instance writing '=' for 'is equal to'. As can be seen from Table 6.2, by counting a symbol as a token word there is an estimated average saving of one word every time a specialised formal notation symbol is used and a saving of two words every time a mathematical symbol is used. In our example 'is equal to' is 3 words and is replaced by the 'word' =, giving a saving of two words. It may be a better measure to relate this to the overall totals in the readable measurement rather than put it as a vocabulary measurement because ADT, which is clearly the most concise, seems to save little by this method of counting. This latter discrepancy arises because the algebraic method has a very small vocabulary of terms which are used repeatedly.

Length

Boxes round schemas and details about states were left out of this total.

6.2.4 First evaluations with nine specifications

These first attempts at measuring characteristics of formal specifications concentrated on the standard text book problems where there were only five operations to be carried out with usually a single entity. Nine small specifications were written and the measurements calculated by hand. The specifications were checked by hand and by a second specifier.

6.2.4.1 P1,P2,P3 ***The priority queue***

These specifications relate to a queue consisting of items where each has a priority attached so that the item with the highest priority is removed from the queue. The specification was approached in three different ways within VDM using the different underlying structures used by this notation; sequences, sets and maps.

6.2.4.2 S1,S2,S3 ***The stack.***

This example must appear in nearly every introductory book on Formal Methods. It is an ordinary stack with items being placed on and taken off the top of the stack. In the VDM and Z versions the underlying structure was a sequence in both cases and here the difference with ADT becomes apparent. Using the method of algebraic data types the behaviour of the abstract form under the operations is what is being described and the fact that the stack is specified as a sequence is not apparent. All details about the sequence structure are abstracted out. These three specifications are included in Appendix C for comparison.

6.2.4.3 A1,A2,A3 ***Aircraft Boarding.***

These specifications, again written in all three notations, are using a set based structure to describe the movement of people on and off an aircraft. It is a simple version of the problem and could equally well apply to students joining a course or any membership class of problems.

The example given in figure 6.1 shows one of the small VDM specifications of a priority queue and, within the box, some of the measurements given on the types of notation used. All nine specifications are given in [Finney 1993]. Appendix D shows the results of the counting from all nine specifications.

6.2.5 Further examples with a refined set of measurements

A refined set of measurements was applied to a larger problem. The *readable* sub-attribute of the specification was split into 4 measurements to refer to brackets, symbols and the normalised figures of brackets / line and symbols / line. The *hidden detail* count took note of the number of built in functions used as well as the number of specifier defined functions. The measurement of symbols was split into 2 counts. The first only took into account the initial instance whereas the second figure estimated the occurrences for the specification as a whole.

Three specifications SP1, SP2, SP3 were written in Z, VDM and ADT relating to the same statistical calculating device. These specifications included definitions of standard mathematical procedures such as mode and variance. There was as much commonality as possible among the three specifications so that comparisons could be meaningful. When the results were analysed the differences between the notations became more apparent than in the previous smaller specifications and more significant conclusions could be drawn about the effectiveness of the measurements. Appendix E gives the results of this second set of comparisons.

Model	symbol count
Qtp = Qitem*	*
q:Qtp	:
q = [p_1,p_2,p_3,...	[2]
q_1,q_2,q_3,..]	
inv(q) p(q(i)) ≤ p(q(i+1))	(10) ≤
CREATE()	(2)
ext wr q:Qtp	:
post q = [ ]	[2]
ENQ(it:Qitem)	:(2)
ext wr q:Qtp	:
post ∃ i ∈ inds q ∍	∃∍∈
del(q,i) = q' ∧ q(i) = it	'(4)∧
DEQ() It:Qitem	(2):
ext wr q:Qtp	:
pre q ≠ [ ]	[2]
post q = tl q' ∧ hd q'= it	”∧
TOP() it:Qitem	(2):
ext rd q:Qtp	:
pre q ≠ [ ]	[2]
post it = hd q	
ISMT() b:Boolean	(2):
ext rd q:Qtp	:
post b ↔ q = [ ]	↔[2]

This gives totals of 34 brackets
22 symbols split into 6 formal types : _ ∃ ⇔ ' *
and 4 mathematical types ≤ ∈ ∍ ∧
the total lines were 23
NB = and ≠ were ignored and all brackets were treated alike so (4) + [2] = 6 brackets

Figure 6.1 The sequence version of a VDM priority queue (P1) with its counting totals

From these larger specifications it seems that:

6.2.5.1 Length

The overall length of the specifications was a good indicator of the differences in the conciseness of the methods. It is highest in VDM mainly because of the repetition of the specifier-defined functions; Z is more concise but ADT is the shortest as expected

6.2.5.2 Brackets

Counting brackets led to the conclusion that Z is the least complex as is to be expected by the structure of the schemas which do not insist on inputs in brackets and use line separation to imply conjunction. The higher numbers on ADT are what would be expected of a specification method that has many nested statements. The surprising result is the large total of brackets in VDM which was mainly the result of the application of so many specifier-defined functions.

6.2.5.3 Symbolic count and symbols normalised by lines

A simple count of the symbolic notation proves to be a good indicator of the inheritance properties of Z which is again seen to be reduced in its overall count because of its schema inclusion. This means that repetition is avoided whereas in VDM the statements and functions must be repeated in each operation as these are regarded as independent. However this measure proved to be misleading in the case of ADT because of the number of repetitions of the standard axiom form (::=.)

6.2.5.4 Normalised comment counts

It is difficult to know whether the measurement of comments per line reflects the true paucity of comments for ADT or whether it is impossible to put an explanation to what is essentially just an equation. It is more natural to insert natural language comments in the other two notations after each schema or section. Using the ratio measure of comments to lines might show a general indication of readability but the very diverse nature of the methods, particularly ADT in comparison to the other two, makes this measurement difficult to use for comparison.

6.2.5.5 Modification counts

Modification counting with respect to the entities did show up the drawbacks of VDM. The reduction for Z over VDM in the extra lines generated by an additional entity is due to the schema calculus that allows information stated in the original schema in Z to be automatically carried through to later parts of the specification whilst it must be restated in the 'wr' and 'rd' parts of each of the VDM operations. In ADT the extra entity is incorporated into the inputs with very little extra work. We recognise here that work and effort may not always be equated. For example propagating a change in state definition through the operations may be tedious and repetitious but will also be light in effort.

Counting modifications from the operations shows the problem of trying to introduce an extra operation into an ADT. If it is a constructor then the effect in this example is to add another 7 axioms estimated at an average of 5 lines each. A non-constructor would only lead to about 6 lines extra.

Revalidation effort associated with modifications has not been included here.

6.2.5.6 Hidden details

In tabulating the results for the hidden detail in specified base types or user functions there was a presumption that ADT does not have any built-in functions. It brought out again the richness of the VDM notation over the version of Z used here and also highlighted the fact that ADT needs very little in the way of functions as it operates at a more abstract level and only gives reduction rules rather than methods on the whole state.

6.2.5.7 Language reduction

This was one of the most striking features to come out of the comparison. The language reduction tabulated was taken as a sample from one of the operations and then multiplied up by a suitable factor. However it was obvious that the ADT notation was going to show very little saving in words because of its appearance as a precise method. Conversely VDM and Z both made heavy use of symbols in this particular operation and therefore the saving over natural language specification was great. In this instance the counting results are a reflection of how verbose the specification could be in natural language which may not be quite the same as how concise it is. The implication for this measurement is that other factors are affecting the reduction from natural language apart from the notation.

The problem domain has an influence on the language used and the form of the notation dictated by the style of the method itself also has an impact on the way that the natural language is translated and reduced to symbols. Certain structures are an integral part of the notation and perhaps should be treated in a similar way to type declarations or procedure headings within programming languages. These latter aspects of code are discounted in some counting methods and it could be argued that a similar approach might be taken with some elements of the formal notations.

6.2.5.8 ***Repetition***

Finally the assumption was made in looking at the re-use of specifier defined functions that ADT requires them to be restated if used again. Obviously VDM is assumed to restate as the operations are regarded as independent. It was therefore a good indication of the redundancy of the methods or the advantage of 'inheritance' mechanisms.

6.3 Summary of results

From these first attempts at applying metrics directly to the formal specifications we have noted:

- Some attributes seem impossible to capture with a simple measurement.
- The issue of modularity, inclusion and re-use within the specification could be reflected by several of the metrics. The method which had the highest repetition count was least able to promote structures or inherit properties. Length as a comparative measure between methods gave an indication of the concise nature of ADT and also the schema inclusion of Z which both helped to reduce the number of lines needed in relation to VDM.
- The symbol metrics had to be normalised to reflect the very concise nature of ADT. Even then it was difficult to reflect the small vocabulary repeatedly used in this method rather than the larger diversity of symbols in the other two. The metric should take into account the number of different symbols.
- The metric used with respect to brackets emphasised the nesting used in ADT which can make it difficult to read and interpret.
- The richness of the notation can be measured by looking at how many additional functions have to be added by the specifier.

- No comparative information was gained from the read or write times as these were speculative estimates rather than experimental results. They do give a very rough idea of the time to be expected but a lot of other factors such as the expertise of the user would affect these figures.

6.4 Conclusions

In this chapter we have considered some of the desirable external attributes of formal specifications. We have formed a subset of these characteristics and linked them with internal properties of specifications. Using examples from three different Formal Methods (VDM, Z and ADT), we have tried to construct suitable metrics to capture the internal attributes of each. We have taken a comparative approach to see how well the proposed metrics could capture the diversity and similarities between the three Formal Methods.

The metrics used have concentrated on the structural aspects of the specifications which were thought to have a bearing on aspects of clarity, flexibility, complexity and conciseness. We found some sub-attributes, for example abstraction, impossible to capture using measurement. Some proposed metrics were easily distorted by the problem domain or the specifier's approach. Other metrics were poor indicators of the properties of the specification and this was shown when they were not sensitive enough to distinguish between the different methods.

The metrics which were judged to have some value in measuring attributes were further refined in the later experiments of the statistical device to better reflect the attributes they were intended to measure. Finally a small set of metrics proved to capture well the structural attributes of the specifications and provide indications of the ways in which these properties of the specifications might be investigated further.

CHAPTER SEVEN

7. EXPERIMENTAL WORK

This chapter contains the background and rationale to the experimental work. We consider both the parallels with similar work on the comprehension of programming languages and also other experimental work with specifications. We then describe the first two experiments that were designed to look at some of the attributes of a formal specification in Z with the emphasis on the factors affecting the comprehension of the specification. Results and analysis are given.

7.1 Introduction

Formal Methods have been proposed as part of the software development lifecycle for about 25 years but there is still much debate on the benefits their inclusion will produce and the claims that can be made, [Hall 1990], [Saiedian 1996], [Bowen and Hinchey 1995] and [Cohen 1989]. As we have stated before their widespread adoption in industry will only take place when there is convincing evidence of the advantages to be gained from using them. Whilst there is much anecdotal and qualitative support there is little in terms of facts, figures and statistics to make the case for Formal Methods and provide hard evidence. We have looked in Chapters 4 and 5 at some of the efforts made to collect this evidence.

Empirical work has been carried out to investigate areas of concern in programming and a variety of metrics have been developed in relation to software and its development as we have seen in Chapter 3. So far there has not been a similar move from the formal methods community to develop metrics which can be applied to formal specifications. In Chapter 6 we described our first attempts at establishing ways of measuring attributes of formal specifications that could lead to an understanding of the nature of the specification itself. In particular we concentrated on aspects of the comprehension we could measure by inspecting the notation.

There have been almost no formal experiments conducted to investigate the efficacy of Formal Methods so the experimental work relating to the comprehension of Z specifications we describe in this and the following chapter breaks new ground.

We begin by looking at some related research in three areas:

- general experimental work in software engineering on factors affecting quality,
- particular experiments carried out on the comprehensibility of programming languages,
- Formal experiments on logic and specifications.

7.2 Related experimental work

7.2.1 General experimental work in software engineering

Much of the work in this category was carried out to try and establish the validity of certain metrics or methods that might affect program quality. As such some of these have already been mentioned in Chapter 3 as part of the survey of software metrics. In the late 1970's and early 1980's there was an explosion in the number and variety of programming languages and this led to an interest in experimental work to try and investigate the properties of the ever increasing amount of software. Basili and his colleagues at the University of Maryland initiated work on the quantitative analysis of the software development process and its products. He has published papers on the methodology of empirical work, for example [Basili 1981, Basili 1985, Basili and Selby 1984, Basili and Weiss 1984, Basili et al. 1986], but also reports on his own experiments with results and statistical analysis. In [Basili and Hutchens 1983] he describes experiments to look at complexity metrics on student written compilers, and in [Basili et al. 1983] an attempt is made to compare metrics using about 110 KLOC in Fortran. His work on the TAME project [Basili and Rombach 1988] was an attempt to use the empirical lessons learnt to develop a software engineering process model. In more recent years Basili's prolific output has included experimental work in many diverse aspects of software including pattern recognition [Briand et al. 1992] and knowledge based analysis [Abd-El-Hafiz and Basili 1996].

For background on experimental methodology many papers still refer to Brooks' work which discusses the problems of the selection of subjects, materials and measures when carrying out studies on programmer behaviour [Brooks 1980]. It also contains a much quoted justification for using student subjects in experiments although this still has to be balanced by those who have reservations about this group of participants [Curtis 1986].

As examples of general experimental work done with student subjects we choose three representative papers. Zweben et al. describe three controlled experiments to look at the effect of layering in Ada on quality and development effort [Zweben et al. 1995], they found that with layering there was a reduction in effort while quality was maintained. Shepperd based his studies on student teams working in Cobol to look at the validation of some design metrics [Shepperd 1990]. A more recent paper discussed experiments to evaluate the effect of inheritance depth on the maintainability of C^{++} programs. [Daly et al. 1996]. They found that three levels of inheritance seemed easier to maintain than the equivalent software with either no inheritance or five levels.

7.2.2 Experimental work on factors affecting programming comprehension

We now consider some of the empirical work carried out with relevant attributes of programming languages in order to see if parallels can be drawn in order to develop a similar approach with formal specifications. One of the starting points for these experiments was the work done by Tenny [Tenny 1988], Harold [Harold 1986] Takang [Takang et al. 1996] and Jørgensen [Jørgensen 1980], who conducted studies with programmers to look at effects of style and structure within programs. There is much other relevant work in programming comprehension and among others Curtis, Green, Gilmore and Solloway have undertaken this [Curtis 1980, Green 1980, Gilmore 1990, Gilmore 1994 Solloway et al.1983].

Tenny analysed the results from 148 student programmers with experience of Fortran, Pascal and Cobol who had been given one of 6 variations of a program in PL/I. He tested the effects of adding or omitting comment lines in natural language. He also looked at the effect of breaking down the program using procedures. He measured their scores obtained by answering 12 questions. From his results he concluded that the commenting had a significant effect on the readability but that the structure of the programs had little effect on the readability. He also concluded that even if a program was badly structured good commenting might overcome that disadvantage. He felt that a larger program would be necessary to test the effect of structuring alone.

In his studies of Cobol programmers Harold tried experimental evaluation of program quality and included features of 'readability and understandability'. As factors that might affect these two aspects of a program he included comments, procedure names, sequential flow of logic, module size, indentation and logical simplicity. He found evidence to link the quality of programmes to the techniques of structuring; however the sample size of 20 was quite small and some of the results needed further evaluation.

In the study undertaken by Takang et al. the authors stress the need for proper statistical techniques to analyse the data from experiments on program comprehension data. In order to use these techniques in a valid way there also needs to be some understanding of the theories of experimentation. In their recent work they conducted an experiment using 89 students and used both an objective and subjective means of assessing comprehensibility. Three hypotheses were tested:

1) that programs with comments were more understandable than those without,

2) that programs with full identifier names were more understandable than those without,

3) the combined effect of comments and full identifier names was better than either one independently.

Only hypothesis 1 was supported by the objective scores and only hypothesis 2 was supported by the subjective scores.

Jørgensen followed up previous experiments by Weissman who concentrated on five significant style characteristics in programs [Weissman1974]. These were:

- choice of variable names,
- structure of the program,
- use of comments,
- choice of control structures,
- paragraphing of the listing.

He based his experiments on Algol programs and tried to correlate the readability of the programs, as judged by 10 experts, with some computer evaluated characteristics of the code. His results were not very conclusive but the criteria he undertook act as pointers to possible characteristics in formal specifications.

7.2.3 Experimental work on Logic

As Formal Methods are based on logical expressions we were also interested in experimental work done in this area. Studies by Almstrum [Almstrum 1994] and Vitner and Loomes [Vitner et al., Loomes and Vinter, Vitner 1996a, Vitner 1996b] have looked at the comprehension of logic and the limitations of computer science students and lecturers in this area.

Almstrum's work compares the difficulty students have with those parts of computing which are related to logic against the other areas that they study. She first used 40 'judges', including some well known names such as Dijkstra and Gries, to classify computing topics by their relation to logic (strongly related or not). The source of these topics was from the Advanced Placement Examinations in Computer Science (APCS) in America which is taken by several thousand students each year and is based on the first 2 courses in the post- secondary computer science curriculum. She chooses 2 levels either "liberal partitioning" where she assigns a strongly related label to a topic if 50% or more judges rated it a main concept or vital subconcept, or "conservative partitioning" where she uses a figure of 75%. Under liberal partitioning she found there was a significant difference between the difficulty distributions of the strongly related items to those only weakly related to logic. These results have a direct bearing on the willingness of people to adopt Formal Methods: if people experience difficulty with logic they will have problems trying to master Formal Methods.

In a set of technical reports Vitner looks at the psychology of reasoning about logical statements and does pilot tests on 12 subjects using logic and Z. He then extends this work with two further experiments making comparisons using computer scientists, 60 and 40 subjects respectively. These were found in several different institutions and took part by answering a series of questions sent by post. The conclusions seem to be, from the results of this work to date, that there are errors in reasoning about logic arising from the use of Z which are similar to those found when using the same information expressed in natural language. Again this has an impact on the case for the adoption of Formal Methods as it seems to imply that people will continue to make errors in reasoning even when presented with a formal specification designed to be unambiguous and clear.

7.3 Background to the new experiments

7.3.1 Teaching experiences

When we analyse the difficulties people have with formal specifications, we can see parallels in the general resistance there is to mathematics. One of the main barriers in the comprehension of mathematics has to do with the problems of its own specialised 'language'. Experience in the teaching of mathematics at any level reveals the difficulty that a large proportion the population have with the use of abstracted symbolic notations [Hackney, 1991]. Some, who have mastered arithmetic and can cope with geometry, reach an insurmountable barrier with algebra. An Assessment of Performance Unit (APU) report shows decreases year by year in the ability of the average pupil of the United Kingdom to cope with algebra and number, whilst reporting small increases in ability in geometry and data handling [Burghes, 1992]. Reasons cited include: the sudden introduction of 'modern mathematics'; a serious cutback in work involving natural numbers; a massive reduction in the basic algebraic content of the GCSE syllabus and indiscriminate use of calculators both in the classroom and in all examinations [Roy, 1992].

Information which is easily understood in natural language becomes incomprehensible to many people when expressed symbolically. In both the surveys described in Chapter 4 and in the majority of articles on Formal Methods the issues of training and mathematical background are usually mentioned as being crucial to the acceptance of the method.

Those who have taught formal methods to first and second year undergraduates all experienced the same resistance to Z as to ordinary school algebra. Students express difficulties with the Z notation because by its nature it has large numbers of expressions of predicate logic. On top of the logic symbols Z also has extra specialised symbols for such items as partial injective functions, schema promotion and post operation states. These we have found leave some students confused and unable to read even simple specifications with any ease.

7.3.2 Experimental rationale

The aims of the experimental work described here and in Chapter 8 were:

- to mirror the work already done in programming languages,
- to look at the problems already highlighted in the teaching of formal methods,
- to collect empirical evidence to test the hypothesis on comprehension.

Following the lead of the work done on program comprehension the experiments were designed to test the readability of formal specifications with special attention to naming of variables, added commenting and structure.

7.3.3 Scale constraints and external factors

To carry out an experiment which involved reading a formal specification we considered the following questions:

- could an existing commercial example be used?
- what effect would the experimental model have?
- what time scale was involved?
- what resources were available?
- could it be replicated?

One or two companies were approached regarding the use of portions of formal specification used in real projects, but it soon became clear that there was a commercial sensitivity about making them available to others outside the companies. This meant that a publicly available specification would have to be used. However, because it was decided to use a comparative type of experiment several versions of the same specification would be required. The solution would be to write a purpose built specification or part of a specification in several versions.

The time scale was dictated by the availability of subjects. Modifications to curricula meant that, although subjects were available in large numbers this might not be the case in the future. This piece of experimental work was not funded in any way and relied largely on the good will and co-operation of colleagues and students.

7.3.4 Subject availability

To get the numbers required for a comparison, and to be able to control external factors such as timing, the best option was to use the students of the Computing and Mathematical Sciences Department at the University of Greenwich. Other alternatives, such as postal testing, questionnaires or the use of practising software engineers, all had experimental disadvantages particularly as timing data would be one of the components.

7.3.5 Formal notation choice

From the surveys and the journals the relative popularity of Z seemed to make it a logical choice as the results could then be given a wider application and audience than if some more obscure notation was used. The use of subjects from the University also meant that Z would be the most practical option as this gave the widest choice of participants.

7.4 EXPERIMENT 1 - An experiment to consider variable names, comments and structure

7.4.1 Design

The two primary factors considered important in affecting the comprehensibility of a formal specification in Z were:

- the use of ***meaningful identifying names*** within the schemas, and
- The effect of ***comments*** in natural language between schemas.

The null hypothesis being tested was;

> **the meaningful names and comments in a specification make no difference to its comprehensibility.**

Thus the experimental design was a 2x2 factorial, involving four versions of the **same specification**, each having a different combination of these factors, as indicated in Table 7.1.

		Meaningful Names	
		No	Yes
Comments	No	A	B
	Yes	C	D

Table 7.1 Specifications in terms of the two primary factors

It was conjectured that version D (with meaningful variable names and additional natural language comments) would be the most comprehensible and A (without either) the least. All these specifications contain a certain degree of structuring although they are only small fragments of a larger specification. To enable an assessment of the effect of this structuring, a monolithic form of D was produced, E, by combining the original schemas into one. There are two possible effects of this:

- the lack of structure will not affect the expected advantage of D, or,
- the lack of structure will undermine the advantage expected in specification D.

The five specifications were assigned random numbers so that the subjects of the experiment could not deduce anything from the rank. They were coded as shown in Table 7.2.

Specification	1(D)	2(C)	3(A)	4(E)	5(B)
Meaningful names	1	0	0	1	1
Comments	1	1	0	1	0
Structure	1	1	1	0	1

Table 7.2 The coding of specification labelling

The hypothesised ranking, in terms of comprehensibility, from easy to difficult, was

1 4 (5 2) 3. It was conjectured that the lack of structure in such small specifications would be small, and the bracketing of 5(B) and 2(C) indicates our prior lack of view of the relative importance of comments and naming in isolation. The possibility of an interaction between comments and names was also of interest.

7.4.2 Response variables

The response variables were in three forms:

Times

The times taken to read the introductory document to the test, and the times taken to answer each of three questions.

Scores

The scores obtained on these three questions, where 1 was awarded for a correct answer and 0 for an incorrect one. The nature of the questions made it easy to allocate the marks and an answer sheet of acceptable answers was prepared before marking. All marking was done by one person so that there was consistency.

Rankings

Subjective rating of the comprehensibility of the specifications, by each of the subjects. A symmetric five point scale was used with -2 corresponding to 'unclear', and 2 corresponding to 'clear'.

7.4.3 The conduct of the experiment

The subjects of the experiment were undergraduate and postgraduate students at the University of Greenwich. They all had some knowledge of Z and were willing to spend time taking part in an experiment. The uniformity of the background and experience of the students also helped to control some of the bias to the results.

The students came from four classes which were, for the purposes of the experiment, called A, B, C and D. The students in A, B, and C were part-time evening computer science undergraduates. Class A were nearing the end of a one semester unit in formal specification using Z and was at level 2, that is, the equivalent of the second year of a full time degree. Nine students participated in the experiment. Five of these were direct entrants having successfully completed an HND with appropriate grades. The sixteen students of Class B were also level 2 but had completed (but not necessarily passed) the formal methods unit the previous semester and were now on a 'Theory of Computation' unit. Class C was in a software engineering unit at level 3, that is equivalent to the final year of a full-time degree; eleven of these students participated in the experiment. Their experience of Z had been part of a formal methods course in the previous year covering much of the same ground as the new formal specification unit taught to the current level 2 students. Twenty six students of Class D took part in the experiment. These were a mix of day-release part-time and full-time MSc Software Engineering students who had had one three-hour lecture on Z with tutorial exercises three weeks prior to the experiment. This was not a conversion MSc and all had a background in mathematics and computing.

Instruction Sheet

Do not turn over the question paper until you are asked to.

Thank you for agreeing to participate in this experiment. It is being conducted as part of research into styles and metrics in formal specification which we hope to publish in the coming year. The results will be treated confidentially and will not be used as part of any assessment of you or any other person.

As well as this instruction sheet, you will be given an answer sheet to record your answers on and a question sheet which you must not look at until you are asked to do so. The experiment will take part in two phases.

Phase I
On the other side of the question sheet is a simple Z specification and three questions for you to answer about the specification. We would like you to answer those questions and time your response at each step.

When you are asked to start, turn over the question sheet and record the start time on the answer sheet. There are five different specifications of which you have been allocated one on a random basis. Working as quickly as you can record the number of your specification on the answer sheet and then read through the specification.

When you have finished reading through the specification and you are ready to answer the questions record the time again. Answer each question in turn noting the time as you complete each answer. Raise your hand when you have finished. Do not attempt to modify your answers after you have recorded your finish time.

Phase II
After you have looked at one specification you will be given all five and asked to compare them for comprehensibility.

Taking as much time as you need put the five specifications into order on the basis of how comprehensible you feel they are. Carefully sort the specifications until you feel you have ranked them with the least comprehensible first, and the one which is most understandable last. Write down the numbers of the specifications in order on the answer sheet.

Once again, thank you for your time and effort.

Figure 7.1 The instruction sheet given to students

The students were thanked by the experimenter and told the experiment was being conducted as part of research into styles and metrics in formal specification. The students were each given an instruction sheet and a sheet to record their responses. The instruction sheet is reproduced in figure 7.1. It was explained that the results would be treated confidentially and not used as part of any assessment. The instruction sheet was read through and any arising questions answered.

The experiment was then conducted in two phases. In the first phase the students were asked to answer the three questions about a randomly allocated specification and record the time as they responded to each question. These times were later translated into elapsed time. An example of a specification used in the experiment with comments but without meaningful variable names is given in figure 7.2 and the same specification but without the comments but with meaningful names is presented in figure 7.3.

Answers to the three questions were marked generously and sample answers to the example given in fig 7.2 are given below.

Questions

1. What conditions give rise to error messages?

Ans $x? \in \mathrm{dom}\ Net$ or $Avids = \varnothing$ (just identifying the correct section of Z)

Or user already in group or no identity numbers available
(explaining the problems giving rise to error messages)

2. The size of which set would give you the number of current users on the network?

Ans Net' (after user added)

Or Net (before user added)

3. Which set or sets give you information about the total number of users the network will support?

Ans size of Net together with size of Avids

It was later realised that there may have been some minor mistakes in the Z but there was a loose enough marking scheme to allow for several interpretations. The choice of questions was based on testing the students ability not only to pick out the right part of a specification when looking for a particular aspect (in this case the error conditions) but also to understand the significance of the Z in relation to the data modelled.

In the second phase, when this task was completed, each student was given copies of all five specifications and asked to rank them in order of comprehensibility and record the ranking.

The new user's name is taken in and an identity number is assigned from the pool of unused numbers. The unused number set is amended and the new pair of user and their number is added to the existing users.

$$
\begin{array}{|l}
\hline
\textit{Add} \\
\hline
\Delta System \\
x? : Ngrp \\
n? : \mathbb{N} \\
m! : Res \\
\hline
n? \in Avids \\
x? \notin \operatorname{dom} Net \\
Avids' = Avids \setminus \{ n? \} \\
Net' = Net \cup \{ x? \mapsto n? \} \\
m! = OK \\
\hline
\end{array}
$$

Here error messages are generated by the failure of either of the two preconditions in the Add schema.

$$
\begin{array}{|l}
\hline
\textit{AddFail} \\
\hline
\Xi System \\
x? : Ngrp \\
n? : \mathbb{N} \\
e\_m! : Res \\
\hline
(x? \in \operatorname{dom} Net \wedge e\_m! = error\_type1) \vee \\
(Avids = \varnothing \wedge e\_m! = error\_type2) \\
\hline
\end{array}
$$

Finally the behaviours are combined

$$AddUser \mathrel{\widehat{=}} Add \vee AddFail$$

Questions

1. What conditions give rise to error messages?
2. The size of which set would give you the number of current users on the network?
3. Which set or sets give you information about the total number of users the network will support?

Figure 7.2 Specification with comments, without helpful variable names and with structure

$$\begin{array}{|l}
\textit{Add} \\
\hline
\Delta System \\
name? : Person \\
n? : \mathbb{N} \\
message! : Response \\
\hline
n? \in Unused\_Ids \\
name? \notin \operatorname{dom} Users \\
Unused\_Ids' = Unused\_Ids \setminus \{ n? \} \\
Users' = Users \cup \{ name? \mapsto n? \} \\
message! = OK \\
\hline
\end{array}$$

$$\begin{array}{|l}
\textit{AddFail} \\
\hline
\Xi System \\
name? : Person \\
n? : \mathbb{N} \\
e\_message! : Response \\
\hline
(name? \in \operatorname{dom} Users \wedge e\_message! = name\_in\_use) \vee \\
(Unused\_Ids = \varnothing \wedge e\_message! = no\_id\_available) \\
\hline
\end{array}$$

$$AddUser \mathrel{\widehat{=}} Add \vee AddFail$$

Questions

1. What conditions give rise to error messages?

2. The size of which set would give you the number of current users on the network?

3. Which set or sets give you information about the total number of users the network will support?

Figure 7.3 Specification without comments, with helpful variable names and with structure

7.4.4 Results

The first analysis of the results was a simple examination of the differences in scores and times between the classes. Class D, as might be expected having had the least training in Z, were on average slower in completing the first phase of answering three questions. They took an average of 577 seconds. These students were postgraduates and therefore with a higher educational level, but their shorter time studying Z seems to have led to a longer reading time. Class B, who had completed the course and taken an examination in Z a few months previously, performed the best, taking an average time of 363 seconds. One outlier distorts the figures and ignoring this reduces their mean-time to just 317 seconds. Class C, surprisingly, did better than A with an average time of 391 seconds as opposed to 541. However class A's times were distorted by an outlying result which if ignored reduced the class average time to 480 seconds. A graph showing the total time taken to complete the reading of the specification by the individual students, and grouped by class, is given in figure 7.4. Only 58 students are represented here as four failed to provide either a start time or a finish time on their answer sheet and therefore their results had to be ignored.

The scores obtained in answer to the three questions are shown in a bar chart in figure 7.5. This reveals a high proportion (19/62) of the students that could not answer any of the questions correctly, demonstrating a poor understanding of the specification. The bar chart also shows that a similar proportion (20/62) could correctly answer every question implying that with Z, as with a lot of mathematics, it is common that students divide into those who have finished a problem and those who cannot even start it.

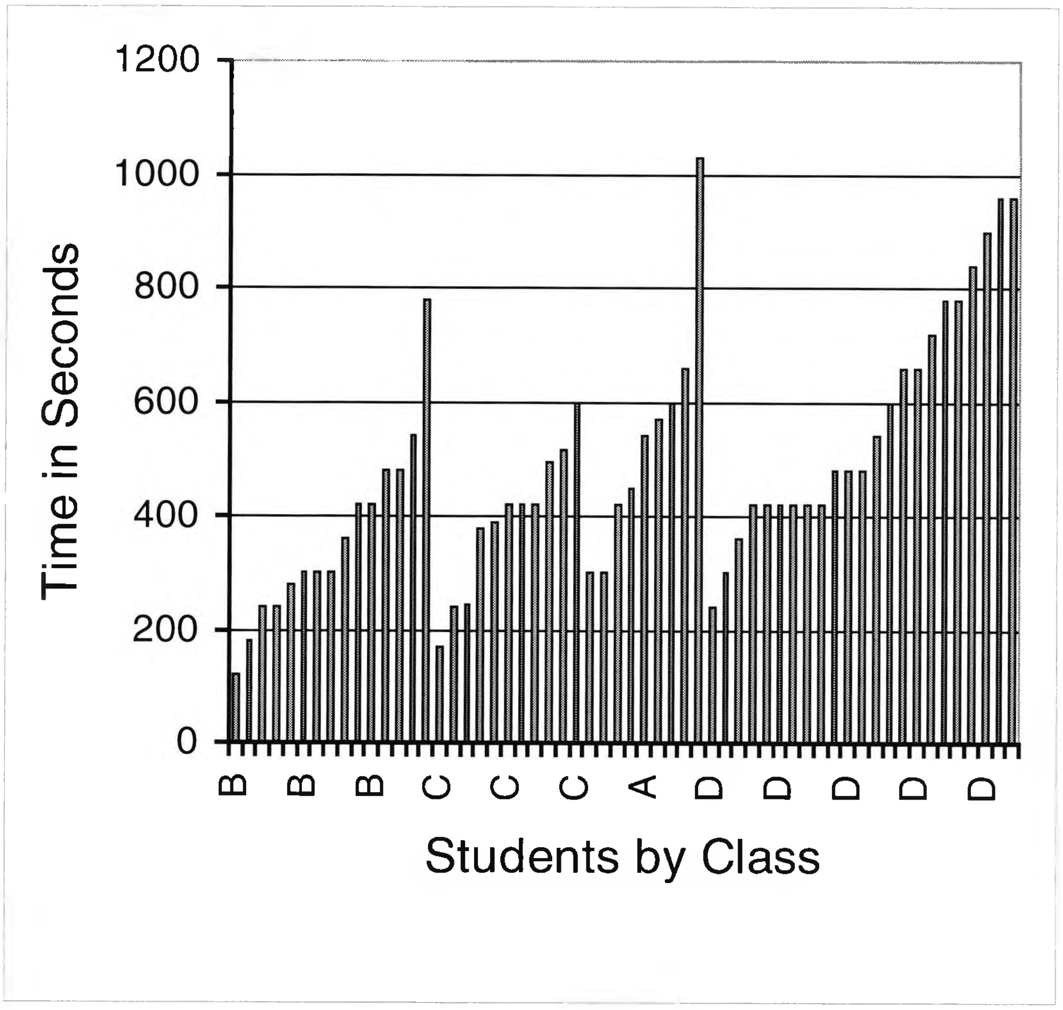


Figure 7.4 The total time taken by each of the students in the four classes

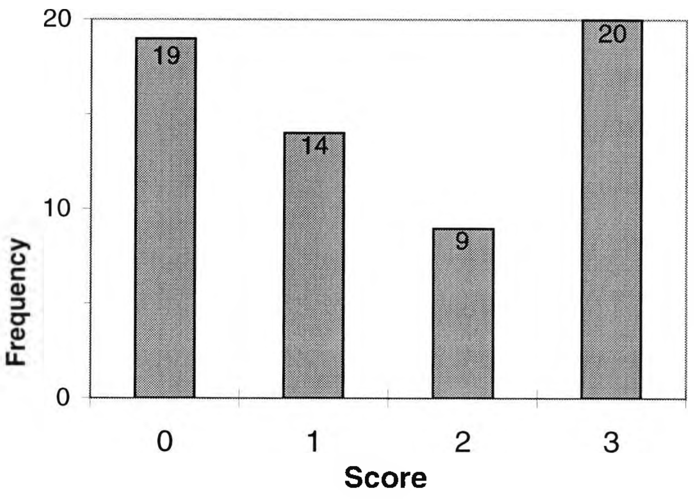


Figure 7.5 The total scores represented in a frequency chart

In figure 7.6 the scores for answering the questions are plotted against the time taken to answer them. The mean times have been marked showing an inverse relationship; this suggests that if students could read and understand the specification then they did not need a long time to do so, whereas those who took a long time to answer the questions got more of them wrong.

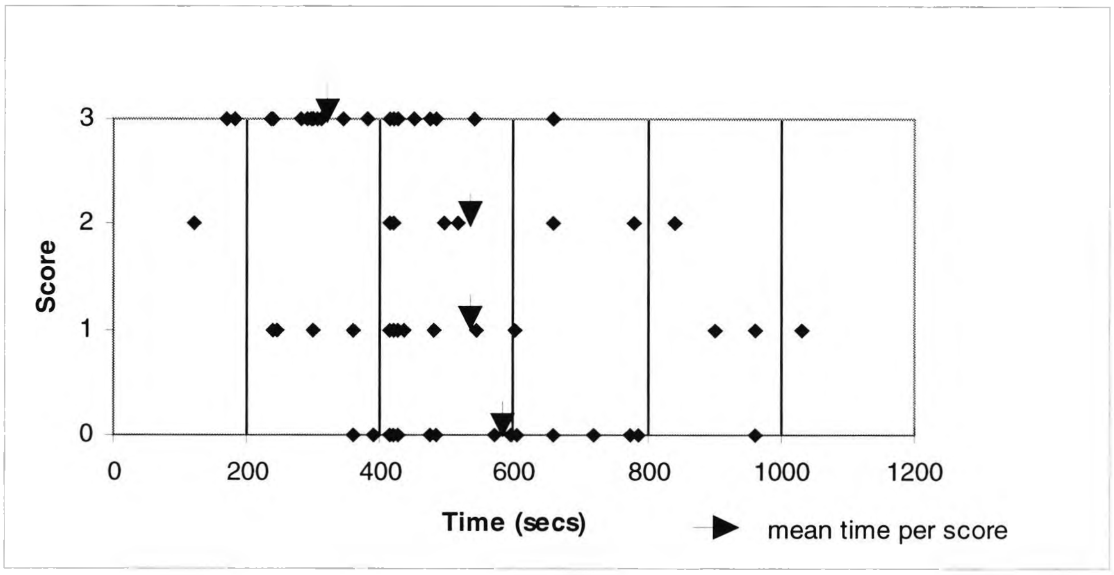


Figure 7.6 The distribution of the scores and times together

Regression analysis was performed on the scores against the three attributes: comments, names and structure. A simple linear regression was fitted by the ordinary least squares method.

The regression equation was:

$$score = 0.974 + 0.259c + 0.676n - 0.059s$$

where

c	is the variable showing the presence or absence of comments
n	is the variable showing the presence or absence of helpful names
s	is the variable showing the presence or absence of blocked schemas

As a result of the analysis only the use of meaningful or helpful identifier names in the schemas seemed to have a significant effect on the score with a p value of 0.05.

Finally the cumulative rankings of the five specifications in terms of their comprehensibility are shown in figure 7.7. This clearly shows the specification with comments and meaningful variable names is rated as most comprehensible and that with neither is the least. These match closely the predicted order 1, 4, 5, 2, 3 only differing in the order of specifications 4 and 5. This implies that in order of importance for the readability of a specification, helpful names are more important than comment levels and least important by comparison is the blocking of one or more schemas together. It is clear that different comments may produce different rankings. Similarly decomposition of schemas may be more important on large or complex specifications and further investigation does need to be conducted.

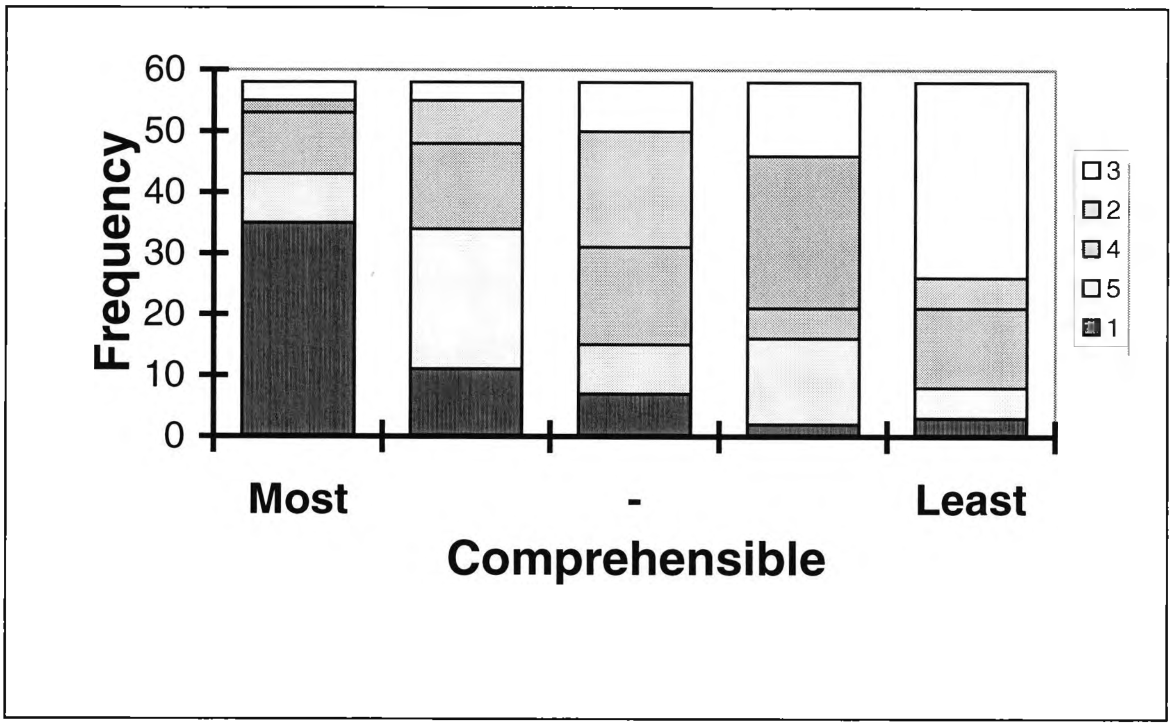


Figure 7.7 The distribution of the rankings over the five specifications

The data was analysed using Minitab [Minitab] and any incomplete data was not included. As only four students did not have either scores or total times this had little overall effect. The different classes taking part were used as a blocking factor in the model.

7.4.5 Conclusions to experiment 1

Large claims for statistical validity and inferences of a far reaching nature about the readability of Z could not be justified by such a small scale experiment. There were sufficient subjects to make some limited observations and to give pointers for further work.

However, it can be concluded that a positive difference can be made to a specification in Z by the use of helpful variable naming and the attachment of appropriate comments. Although there are compelling reasons to believe that large specifications are more readable when partitioned by judicious use of schemas, the scale of the specification used in this experiment was not large enough to show this effect.

From this experiment it seems valid to say that several lessons can be learnt for the teaching of formal methods. One should not underestimate the difficulty of reading a formal specification written in a mathematical notation. Specification must be recognised as a unique form of communication between human beings. Every opportunity should be taken to make the reading easier, particularly by suitable naming of variables and data. Time to assimilate the techniques involved is important and reading does not always imply semantic comprehension or the higher skill level - the ability to articulate. As it was clear that 19 of the subjects did not understand the specification sufficiently to answer any of the questions correctly despite their background, then we should not automatically expect clients and software engineers to master Z and other formal methods without training. Empirical studies such as those of Naur provide strong evidence that a significant proportion of engineers may never master formal techniques irrespective of the training given [Naur 1993].

It is not only for the benefit of the client that the specifications should be read easily but also the author. It is well known in programming, that a time lapse can create difficulties for the writers themselves with their own code. Indeed, the poor style of many programmers' use of C, for example, has given it the undeserved label of a 'write-only' language.

Training practitioners from the start to have appropriate names for identifiers and to include suitable explanatory text will ensure better comprehension from clients, fellow software engineers and implementors.

7.5 EXPERIMENT 2 - Extending experiment 1 with double the subjects and further statistical analysis

When the opportunity arose of a further set of subjects the first experiment was repeated so that a larger data set could be used and more sophisticated statistical analysis applied. The experimental design was the same as before.

7.5.1 Conduct of the Experiment

There were now a total of 147 undergraduate and postgraduate students involved. They came from 6 different classes studying at levels from Higher National Certificate to Masters, and are, for the purposes of the analysis, re-labelled C1 to C6. It was expected that there might be significant differences in the performances of these classes. However in this analysis 'class' is largely treated as an experimental 'blocking' factor to improve the sensitivity of the main factors of the experiment.

7.5.2 Data 'Cleaning'

All missing values have been treated as such and no missing value imputation has been used. Also responses where the time spent on either reading the introductory details or answering any of the questions were zero have been treated as erroneous, probably due to rounding, and the associated times have also been treated as missing values. The data obtained from the experiment, and a summary can be found in Appendix F.

7.5.3 Analysis of timing data

The times taken (in seconds) to answer each question and the total time taken for all three questions are displayed in Figures 7.8 - 7.11.

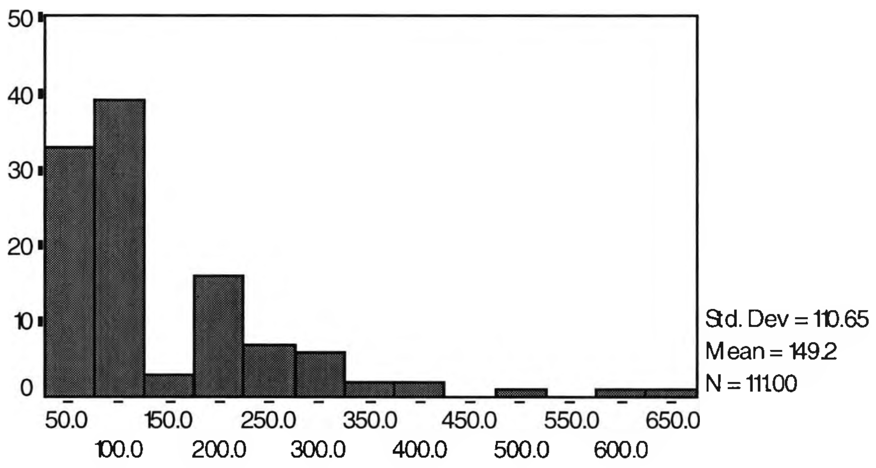


Figure 7.8 Distribution of times taken in seconds to answer Question 1

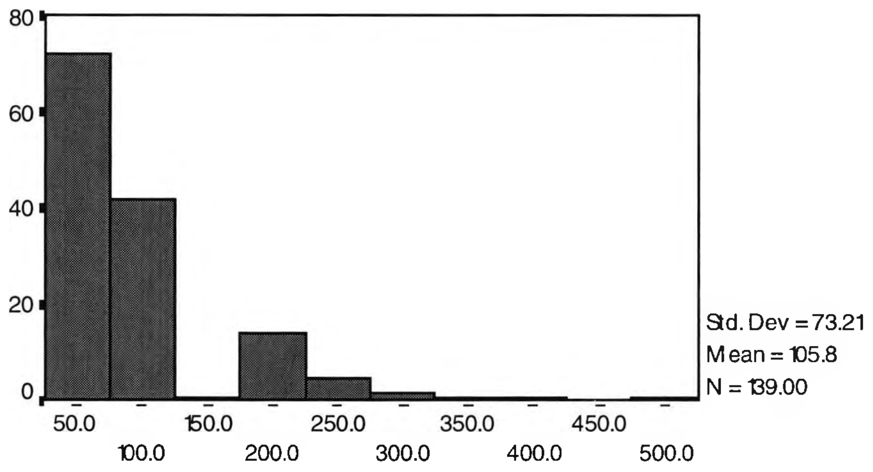


Figure 7.9. Distribution of times taken in seconds to answer Question 2

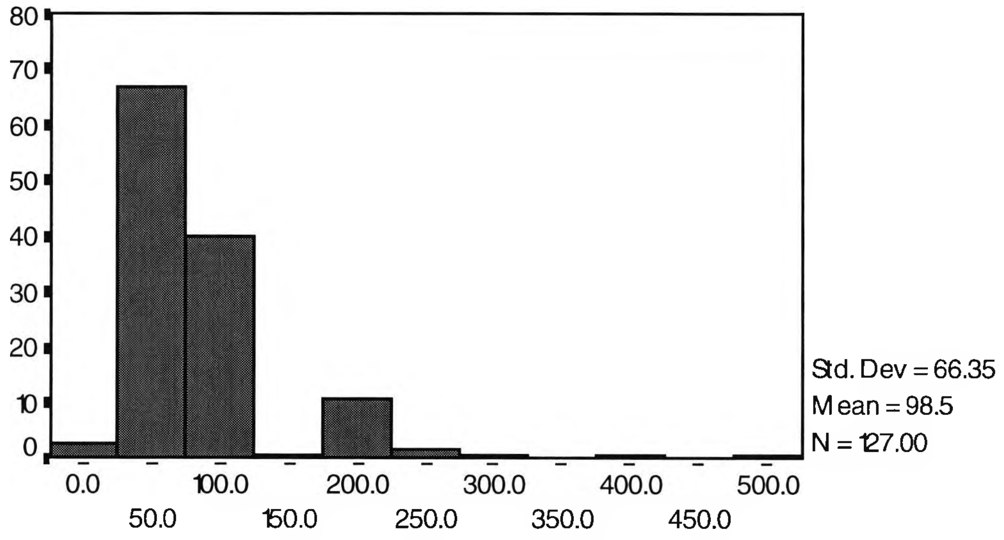


Figure 7.10. Distribution of times taken in seconds to answer Question 3

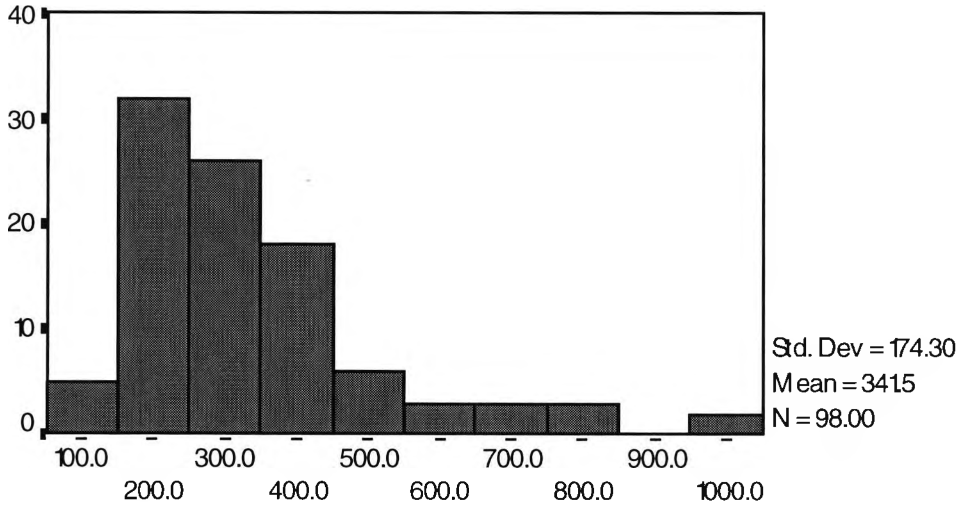


Figure 7.11 Distribution of time taken in seconds to answer all questions

Note that the multi-modality in Figures 7.8-7.10 is caused by the tendency of subjects to round times to the nearest minute. This has been smoothed out in Figure 7.11 for TOTQT. It would be possible, in principle, to take into account the 'rounding' tendency, in a model-based analysis of this timing data. However, such an analysis would be unnecessarily complicated in view of the relatively limited information contained in the timing data. The raw (cleaned) timings have therefore been used in the analyses reported in this case.

The correlation matrix between the timing variables is shown in table 7.3.

	TQ1	TQ2	TQ3	TREAD
TQ1	**1.0000**	**.1360**	**.2700**	**.0384**
	(111)	**(108)**	**(101)**	**(90)**
	P= .	**P= .161**	**P= .006**	**P= .719**
TQ2	**.1360**	**1.0000**	**.2371**	**.0483**
	(108)	**(139)**	**(122)**	**(93)**
	P= .161	**P= .**	**P= .009**	**P= .646**
TQ3	**.2700**	**.2371**	**1.0000**	**.1093**
	(101)	**(122)**	**(127)**	**(87)**
	P= .006	**P= .009**	**P= .**	**P= .313**
TREAD	**.0384**	**.0483**	**.1093**	**1.0000**
	(90)	**(93)**	**(87)**	**(95)**
	P= .719	**P= .646**	**P= .313**	**P= .**

(Coefficient / (Cases) / 2-tailed Significance)

Table 7.3 The matrix of correlation of the times for each question

The times to complete Q1 and Q2 are significantly correlated with the time for Q3, but not with each other. None of the times to complete the questions are significantly correlated with the time to read the introduction.

A Factor Analysis of the timing data was carried out using the correlation matrix, based on an initial Principal Components Analysis. Only one principal component had an eigenvalue greater than one, and hence only one factor was extracted. Hence, we conclude that there was no internal structure to the timings of the components of the test. We note that the proportion of variance accounted for, (the commonalties), are largest for Q1 and Q3 and that the time to read the introductory material is not significantly related to the common time factor. Details are given in Appendix G.

7.5.4 Analysis of scores data

Clearly, the variable most likely to give a good measure of comprehension is the total score. Figure 7.12 shows the distribution of total score, which is bimodal.

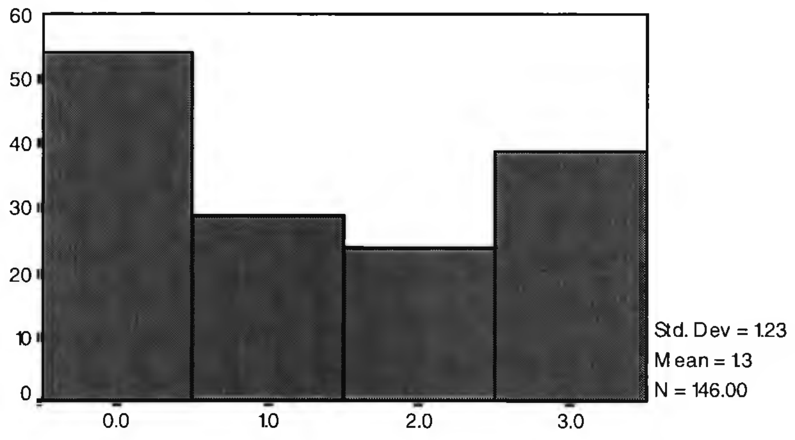


Figure 7.12. Frequency diagram of total score

The modes, 0(the 'failures') and 3(the 'successes'), are analysed in detail later.

With total score as response variable an initial Analysis of Variance (ANOVA) was performed with two factors:

- NLVL, level of student class,
- SPEC, the specification

while the total time taken was used as a covariate. The ANOVA table is given in Appendix H.

It may be seen that the form of specification, SPEC, is significant with a p-value of 0.026, but that the effect of class level, NLVL, and the interaction, NLVL*SPEC, have not been found to be significant. Accordingly, NLVL has been omitted from the next ANOVA. The mean scores for the different specifications are given in Table 7.4

SPEC				
1	**2**	**3**	**4**	**5**
1.71	**.84**	**1.06**	**1.74**	**1.32**
(21)	**(19)**	**(18)**	**(19)**	**(19)**

Table 7.4 Mean total scores,(number of cases), by specification

We see that the ranking of the specifications, by decreasing mean TOTS, is 4(E) 1(D) 5(B) 3(A) 2(C), which may be compared with the conjectured 1(D) 4(E) (5(B) 2(C)) 3(A). It is seen that 1(D) and 4(E), distinguished by the presence or absence of structure, are not significantly different in terms of mean response, with means 1.71 and 1.74 respectively. Also, the mean scores for 3(A) and 2(C), (1.06 and 0.84), without meaningful names are substantially less than that for 5(B), with meaningful names, (1.32).

The experiment had been designed with a factorial 2x2 + (1) 'treatment-structure', and the ANOVA for this design is given in Appendix I. The only significant factor is MNFUL, i.e. the indicator of whether meaningful names are given to variables. Commenting, CMT, and Structure, STRCT, with p-values of 0.169 and 0.667, are not significant, and there is not a significant interaction between naming and commenting. We note also that the covariate, TOTQT, the total time to answer the questions has been found to be very significant with an estimated regression coefficient for TOTQT of -0.00214 (se = 0.001), establishing an inverse relationship between total score and total time to answer the questions. Unsurprisingly, the better students take less time.

7.5.5 Success and failure analysis

Individual Question Analysis

Multiple Logistic Regression Analysis was done on the responses to each of the three questions. In all three analyses the use of meaningful variable names was the only significant factor, with a significant negative coefficient for time taken for both questions 2 and 3. The results are therefore confirmatory of the results obtained above.

Factors affecting Success, (score=3/3), and Failure, (score=0/3)

Multiple Logistic Regression Analysis was carried out separately on the outcomes: 'Success', (score=3/3), and 'Failure', (score=0/3). The results of the analysis of 'Success' are shown in Appendix J.

The fitted model is, $prob(Success) = \dfrac{1}{1+\exp(-\eta)}$, where $\eta = \sum_{i=0}^{9} B_i V_i$.

B_i is the estimated coefficient of V_i, (B_0 = constant, and $V_0 = 1$).

This model has an 81% correct classification rate on the data.

We note that the meaningfulness of variable names, MNFUL, is ***NOT*** a significant beneficial factor, but commenting, CMT, ***IS***, with a p-value of .0241. More able subjects are able to make use of the information in comments, and hence obtain high scores, but for these subjects the use of meaningful names is of minor relative importance, on such small specifications. We also note that the total time to answer all questions, TOTQT, is a significant inverse indicator of good performance, and that there is a more noticeable variation between the groups when considering this response variable.

In contrast, the logistic analysis of the probability of obtaining a 0/3 score unsurprisingly shows that none of the experimental factors are significant, but that there is a significant positive dependency of the probability of a 0/3 score on the total time to answer the questions, (p=0.034).

We conclude that none of the factors can really be used to help the weakest group who will take a long time over the test and still score zero. For the better group who answer all questions correctly they seem to do this quickly and get help only from the added comments.

7.5.6 Analysis of perceived-comprehensibility rankings

In the second part of the experiment, students ranked all five specifications, from least comprehensible (-2) to most comprehensible (2). Analysis is primarily by graphical means. The data may be found in Appendix K.

Examination of the data/graph for all the subjects, Figure 7.13, confirms that Spec.1, (with comments and meaningful names), is found most comprehensible by most candidates, and is found to be least comprehensible by the least candidates, as conjectured. The converse result is found for Spec.3, (no comments and names without meaning), also as conjectured. The ratings of Spec.'s 2 and 5 are arbitrarily dispersed, with neither getting high or low proportions of the extreme ratings. Spec.4 seems to be perceived similarly to 2 and 5 on the whole.

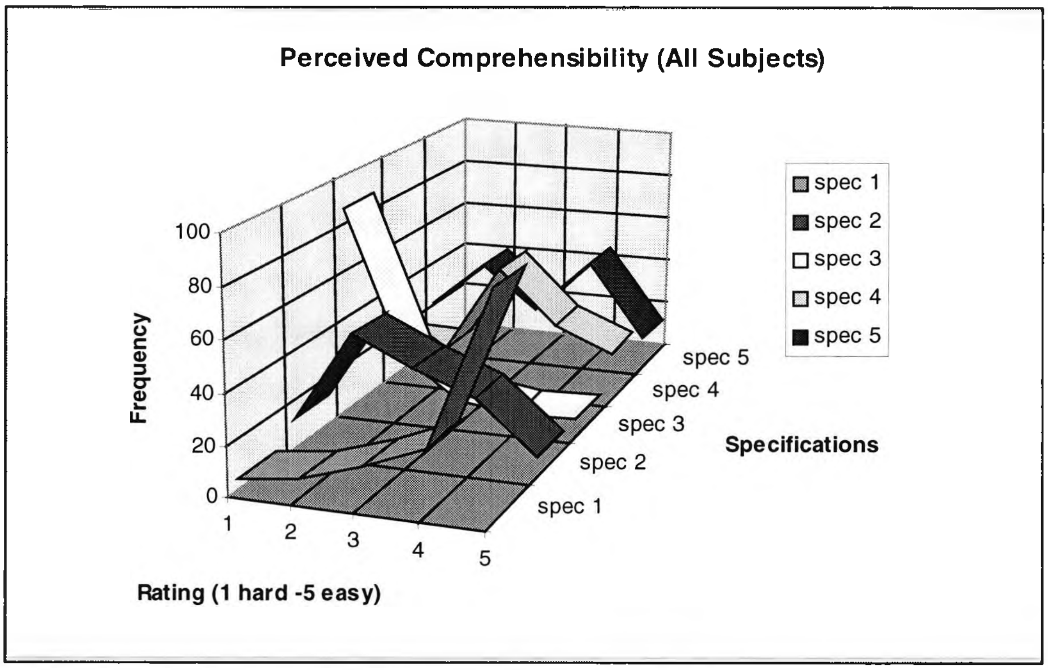


Figure 7.13 Frequency plot of perceived comprehensibility, by specification

Hence it would seem that the monolithic nature of the code in Spec.4 detracts from the perceived clarity of Spec.1, in spite of the fact that structure has not been shown to be a significant factor in the analysis of scores. This is an interesting result with a discrepancy between the actual performances shown by the scores and the perceived comprehensibility as shown by the ratings. It suggests that perceptions of comprehensibility are quite different from reality.

Kendal's W measure of concordance in rankings, [Kendal 1948], of the specifications has the value of 0.147, with a p-value of 0.193 for the null hypothesis that W=0.0. The clear trends for specifications 1 and 3 have been obscured by the lack of concurrence concerning the other specifications, when all subjects are considered together.

We can consider if the perception of the comprehensibility of the alternative specifications depends upon the ability of the subjects, as measured by their total score on the questions, TOTS. For those with TOTS=3, (full marks), the conclusions do not change, and Kendal's W measure of concordance is higher at 0.6047 with a p-value of 0.11, slightly closer to significance as we would expect from the graphical display, but still reduced by the effects of Specs 2,5 and 4. For those with less than full score, the lack of clarity of Spec.3 is unanimous. Also, whilst relatively few of these weaker subjects rate Spec.1 at the low end of the scale, rather surprisingly, few of them see Spec.1 as being superior to Spec.'s 2, 5 and 4. Perhaps the lack of perception of the advantages of comments and meaningful names (and structure), as present in Spec.1 indicates that the weaker subjects do not have the intellectual skills to make use of these meaningful cues. Reasons for this will undoubtedly vary and will include lack of basic intellectual ability and insufficient training.

7.5.7 **Conclusions to experiment 2**

This experiment has extended the results of the previous study to evaluate the impact of the style factors, of naming, commenting and structure on the comprehensibility of the five versions of a small specification.

It was found that those obtaining higher scores take less time than those who obtain low scores. Total time taken is therefore as a correlational surrogate of ability, and has been used as an adjusting covariate in the analysis of how the style factors affect the comprehension scores. Meaningful naming is the only style variable found to be significant in ANOVA, but commenting is found to be the only style factor predictive of a 3/3 score.

For the small specification of this study the contrast between monolithic and structured schemas does not significantly influence the scores obtained. However, the monolithic schema does seem to reduce the *perceived* comprehensibility.

From these results it would seem that:

- a significant number of people with aspirations of becoming software engineers will never be competent to read (let alone write) even the most basic formal specifications. This is irrespective of training and also unaffected by factors relating to the comprehensibility of the specification.
- variable names and commenting can make a small improvement for those people that are competent,

- perceptions of comprehensibility are not matched by performance.

7.6 Summary and overall conclusions

The results of these first two experiments are not good news for the Formal Methods proponents. Their suggestion that with relatively little training software engineers can use formal specifications does not seem to be supported by these findings.

The combined measurements of timings, performance scores, and attitude measurement as used in these two experiments might be taken as a paradigm on which the development of a comprehensibility metric may be based. Clearly, in order to improve the comprehensibility of Z specifications, attention must be paid to the way variables are named and comments used.

Acceptance of these principles is evident in the more recently published textbooks on Z. In most of these efforts are made to use variable names which give clues to the reader and much more natural language explanation is included to supplement the schemas and text. (Compare [Diller 1990] and [Lightfoot 1991] with [Rann et al. 1994]).

In order to develop the work of these experiments, a further study was needed of a specification large enough to investigate the effects of structure as a factor. This is described in Chapter 8.

CHAPTER EIGHT

8. FURTHER EXPERIMENTAL WORK

In this chapter we describe a third experiment which was designed to look at the effect of structure within a Z specification. We analyse the reactions to the same specification presented in three ways: one monolithic form, six large schemas or eighteen smaller schemas.

8.1 Introduction

In the experiments described in Chapter 7 we have investigated factors affecting the comprehensibility of a Z specification. From our results we found that the structure of the schemas did not significantly affect the comprehensibility as measured by the scores of the students. However we also noted that the *perceived* comprehensibility was affected and this has serious implications for the spread of Formal Methods. If Z specifications appear hard to understand because of the structural layout of the schemas, software engineers might be unwilling to put the effort into investigating them further.

Our specification fragments used for Chapter 7 were small, only 15 lines in total in 2 schemas (discounting comments and titles), so that there was a limit to how much structural variation could be introduced. Consequently we decided to conduct a further experiment with a larger specification to investigate the effect of altering the structure.

Here we describe this experiment and the analysis of the results. The subjects were tested on their understanding of the specification using a set of questions. The questions were graded to reflect the fact that they were written to test different aspects of comprehension. The subjects' scores on these questions form the basis of the analysis and are used as indicators of the comprehensibility of the specification. Additional information about the academic background of those taking part was then used to investigate the connection between their general ability and their competence in handling the Z specification. Conclusions are given at the end of the chapter.

8.2 EXPERIMENT 3 -The effects of structure on the comprehensibility of Z specifications

8.2.1 The experimental design

To be able to control the contributing factors completely we again decided that the specifications would not be adapted from existing work but written especially for the experiment. The null hypothesis being tested was:

the particular structure of a specification makes no difference to its comprehensibility.

To this end a Z specification was written of a small internal business telephone directory designed to deal with employees telephone numbers and office location. The same specification was re-written in three ways:

- Specification A was monolithic in form and consisted of 121 lines of which lines 12 - 121 were a single Z schema. (See Appendix L for specification).
- Specification B grouped the same information into 6 main schemas consisting of 3 schemas dealing with operations and 3 matching schemas dealing with error handling. Each of the schemas was about 20 lines long and there were 159 lines in all as some repetition was incurred by this process (see Appendix M for specification).
- Specification C, the longest at 165 lines, took 18 smaller schemas to convey the same information. (See Appendix N for specification).

All the material contained in each of the 3 specifications was identical. Great care was taken to ensure that commenting and notation matched in all 3 specifications so that each reader was given the same information, the only difference between them being their structural forms.

Twenty questions were designed to test the reading of the specification in a variety of ways (see Appendix O for a sample). All lines of the specification were numbered so that reference could be made to particular parts of the specification in the questions and answers. Broadly the questions fell into 4 categories, testing the subject's competence in:

- finding a relevant part of the specification -

  e.g. Which line in the specification tells you......
- understanding the notation -

  e.g. What is indicated by the difference in number? and number! in line ......
- relating the specification to the model-

  e.g. Describe the purpose of line.......
- modifying the specification by writing an extra feature -

  e.g. add the lines needed to include information about the department where an employee works.

The subjects of the experiment were 65 students who were just finishing a one semester course in Z. Their tuition time was approximately 40 hours. To ensure an even distribution of abilities among the 3 different specifications the students were ranked using the average mark for their previous year of study. The students were then assigned to a specification so that the average mark in each group and the spread of abilities were the same for all specification types.

8.2.2 The experiment

At the time of the experiment each student was given a named pack containing the correct specification, a question sheet and an answer template. They were then given an hour to attempt the 20 questions.

All marking was done by one person to ensure uniformity, scores were recorded for each individual question and a total out of 60 was awarded.

There was some small imbalance in the group sizes due to students not attending on the day. In the event the numbers in each group were

Specification A	23
Specification B	23
Specification C	19

8.2.3 The results

The results by score for all students sorted in ascending order are shown in Table 8.1 and the comparisons of the scores for each specification are illustrated in figure 8.1. From a possible total of 60, the scores varied from the highest at 47 to the lowest at 7. An initial inspection shows specification A clearly with lower overall scores than the other two.

SPEC A		7	8	10	11	13	14	17	17	18	18	19	19	19	21	23	23	24	28	28	29	30	35	41
SPEC B		9	14	15	15	19	21	22	23	24	25	26	27	27	29	29	31	32	34	35	35	35	39	47
SPEC C		10	16	19	19	19	19	23	25	25	26	28	28	29	32	38	40	40	41	45				

Table 8.1 Results by score for each student

The average scores were	Specification A	20.52
	Specification B	26.65
	Specification C	27.47

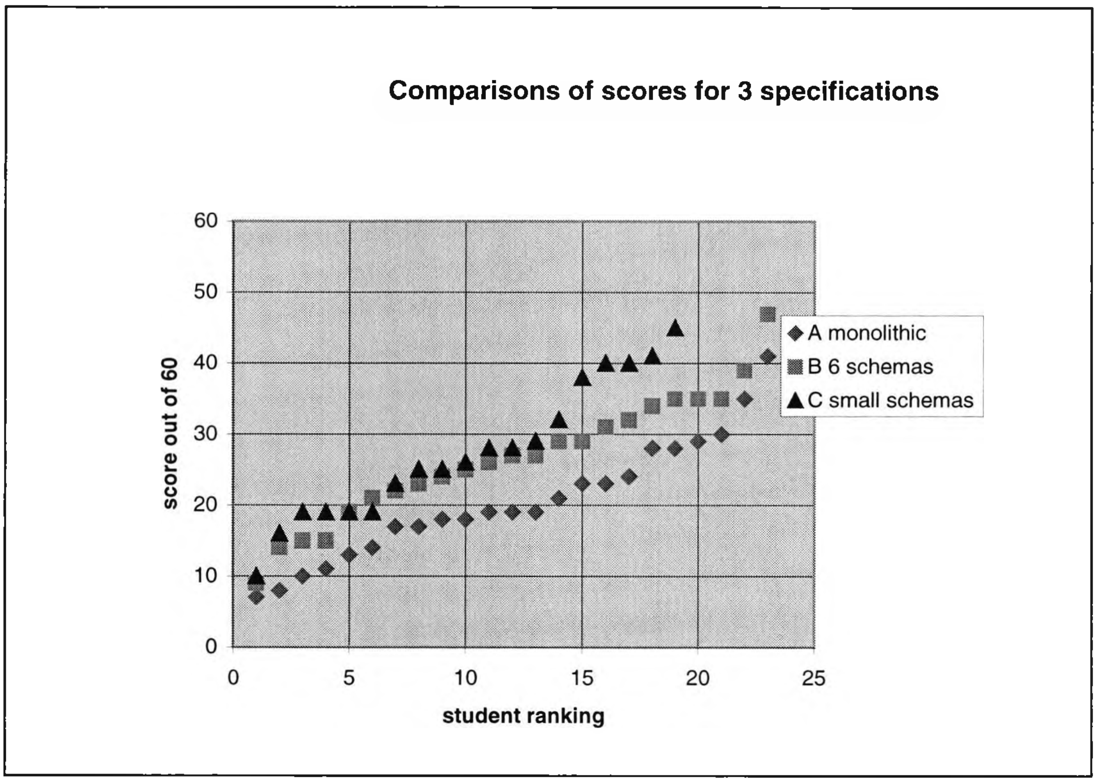


Figure 8.1 Comparison of student scores for each Specification

Figure 8.1 shows the tabular information graphically and Specification A can be seen clearly below the others. The horizontal axis represents the student's place in a ranked list and the vertical axis represents their score. We can see that there is the clearest difference in the lower score range inferring perhaps that the better students could overcome some of the disadvantages of a monolithic structure.

8.2.4 Statistical analysis

The first aim was to see if there were significant differences among the three specifications, bearing in mind the large variation in scores within each treatment. A one way independent measures analysis of variance (ANOVA) was applied to the data using the Minitab statistical package [Minitab] and Table 8.2 shows the output.

8.2.4.1 One-Way Analysis of Variance

Source	DF	SS	MS	F	P
Data	2	627.7	313.9	3.86	0.026
Error	62	5042.1	81.3		
Total	64	5669.8			

Table 8.2 The Output from the ANOVA using Minitab

We can see with a value of $p = 0.026$ that were significant differences amongst the 3 specifications.

With these differences established it is legitimate to look at some pair-wise comparisons amongst the different schema arrangements. These are shown in Table 8.3.

Specification	No. of Values	Mean	Standard Deviation	Standard Error of mean
A (Monolithic)	23	20.52	8.48	1.8
B (6 schemas)	23	26.57	8.95	1.9
C (small schemas)	19	20.9	9.71	2.2

Table 8.3 Comparisons amongst different specifications

Using a two sample t-test we obtained the results for the three pair-wise comparisons at the 5% level:

A and B; $p = 0.023$ with a 95% confidence interval for the difference of the means A-B [-11.2, -0.9]

A and C; $p = 0.020$ with a 95% confidence interval for the difference of the means A-C [-12.7, -1.2]

B and C; $p = 0.76$ with a 95% confidence interval for the difference of the means B-C [-6.8, 5.0]

By applying these pair-wise tests we confirmed that there were significant differences between A and B and also between A and C. There is no significant difference between B and C.

With the operation of 3 comparisons, the danger of using a t-test is the introduction of an unacceptable level of Type I error, i.e. we may falsely reject the null hypothesis when it is in fact true. The more the number of comparisons the greater the error and we can show it is approximately 14% in this case using

$$\text{error} = 1 - (0.95)^3$$

To check these results we used a more stringent test which makes use of

Tukeys Honestly Significant Difference (HSD) $= q\sqrt{\dfrac{\text{Mean square error}}{n}}$ where q is the value obtained from the Studentised Range Statistics tables. In this case the value of q (3,62) was 3.4. As there are unequal numbers in each group we use

$$n = \frac{3}{\frac{1}{23} + \frac{1}{23} + \frac{1}{19}} = 21.5$$

This led to a value of HSD = 6.63. This is simply used as a guide to the difference in means between the groups compared. We can see from Table 8.4 that the numerical differences were:

differences between means	A	B
B	6.13	
C	6.95	0.82

Table 8.4 Differences of mean scores of specifications (signs ignored)

By comparison with the Tukeys statistic we can see the differences between A and C were clearly significant and A and B were very close to significant while B and C again showed no difference. In his book [Hinton 1995] stresses the need to use judgement on those differences which almost reach the level of significance. In conjunction with the results of the t-tests we can take the difference between A and B to be important.

It is perhaps surprising to see such a low figure for the comparison between B and C. The Scheffé test allows a post hoc comparison of particular aspects of the experiment while omitting others. In this case we can form a weighted comparison between B and C but ignore A using weights of 0, -1 and 1 as coefficients in the calculation of the F statistic of comparison.

We find $F = 2.425$ which must be compared with the table value of $F(1,62) = 4.00$ and so is a confirmation of all the other tests that there was no significant difference between specifications B and C.
By applying pair-wise tests we can confirm that these differences were significant between A and B and also A and C, whilst there was no significant difference between B and C. All tests were done at the 5% level.

8.2.4.2 Initial conclusions
From this initial analysis we can reject the null hypothesis. The main conclusion is that a large monolithic schema is less comprehensible than a specification broken up into modules. Clearly one large block of Z makes comprehensibility difficult but there seems to be no significant improvement in comprehensibility to be gained by breaking down the schemas further once they are reduced to about 20 lines long. We can therefore recommend that in writing a formal specification some care should be taken over the structural presentation.

8.2.4.3 Further analysis
To investigate the connection between the performance of the students on the Z specification and their known ability from their previous academic history, scatter diagrams were plotted (Figures 8.2-8.5). Each of the diagrams shows the student's score for the specification plotted on the horizontal axis with their previous average mark plotted on the vertical axis.

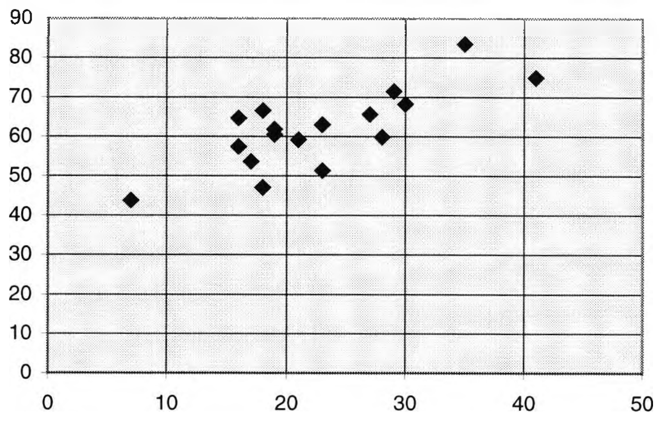


Figure 8.2 Scatter diagram for Specification A

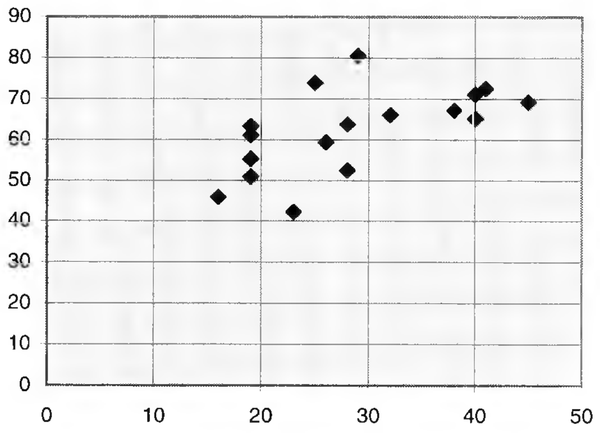


Figure 8.3 Scatter diagram for Specification B

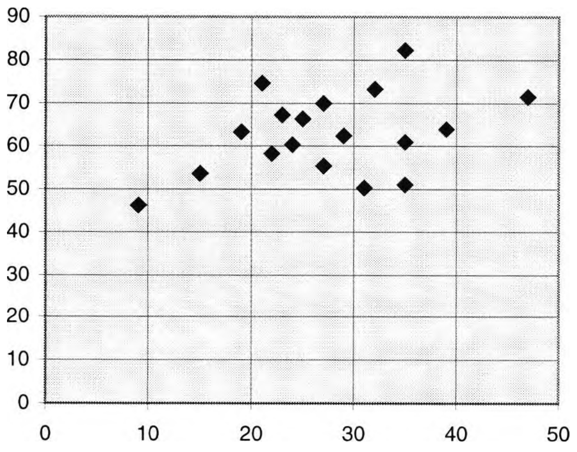


Figure 8.4 Scatter diagram for Specification C

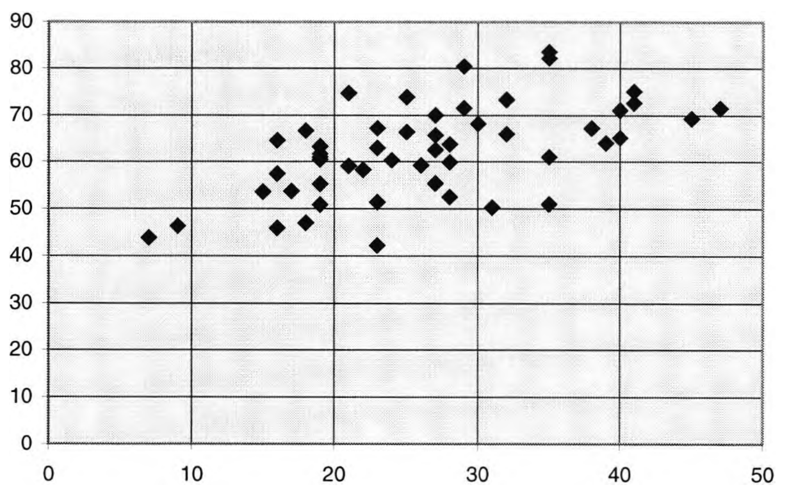


Figure 8.5 All Specifications scores against previous marks

Pearson's r correlation coefficient was calculated for each of the specifications and compared with the expected value. Not all students had a known previous history so only those with complete pairs of marks were used. The degree of freedom for each set of n students was n-2 and the test was one tailed as we would predict that those students with a good previous result would obtain high scores.

	pairs of marks	Calculated correlation coefficient	degrees of freedom	Expected value of r	significant
specification A	17	0.7819	15	0.4124	yes
specification B	16	0.3933	14	0.4000	no
specification C	17	0.5980	15	0.4124	yes
overall	50	0.5669	48	0.2400	yes

Table 8.5 The correlation between the three specifications

8.2.4.4 Results of the correlation tests

We can see from Table 8.5 that there is clear correlation between the scores of the test and the known ability of the groups given specifications A and C. The coefficient is not quite in the significance range for specification B. The very strong correlation in the first case would seem to imply that the monolithic specification has acted as a good differentiator of abilities; only the known able students could score well and the weak ones found the questions very difficult. This is less true of specification C where the correlation coefficient at 0.5980 is not as strong. Overall there is correlation as can be seen from the scatter diagram in Figure 8.5 showing all the students on the same diagram.

8.2.4.5 Analysis of question type

Analysis was undertaken of the scores for different types of question. The graph in Figure 8.6 shows the average score for each question plotted as a bar chart and the linear display over the top displaying the maximum marks available for that question. It can be seen that the questions at the end show a larger discrepancy between the average and the maximum. This is partly explained by the time factor. The average marks for questions 15-20 were reduced overall because several students failed to finish. It was perhaps a design fault that the most difficult questions were placed at the end but they required modification of the specification and so could best be attempted after the initial questions had been answered.

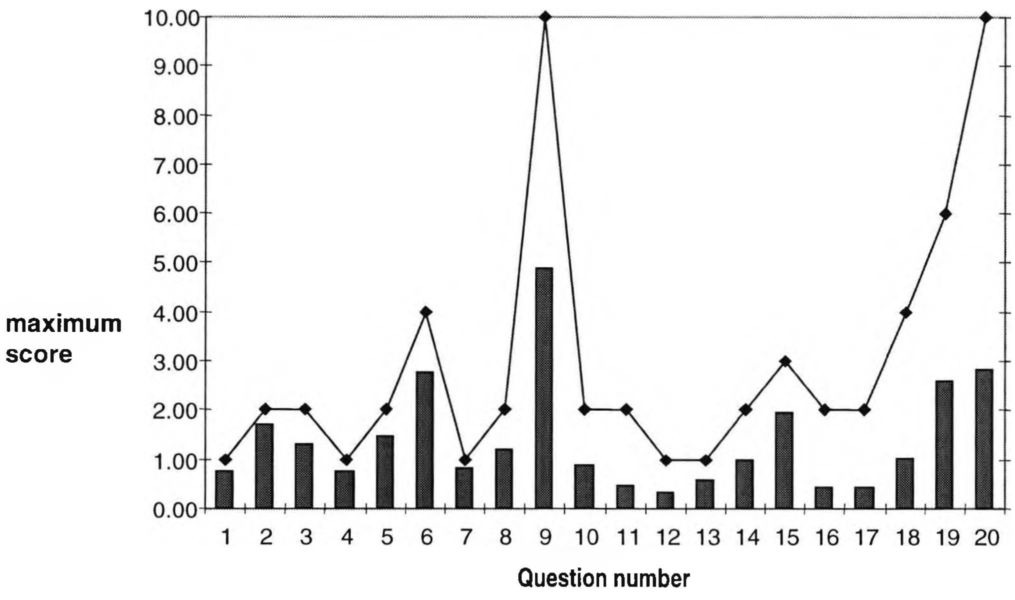


Figure 8.6 Average student scores for each question shown with the maximum

The questions were of 4 different types and were allocated a degree of difficulty from

1-4. The bar chart showing their classification of difficulty is given as Figure 8.7 and the full type classification is in Table 8.6

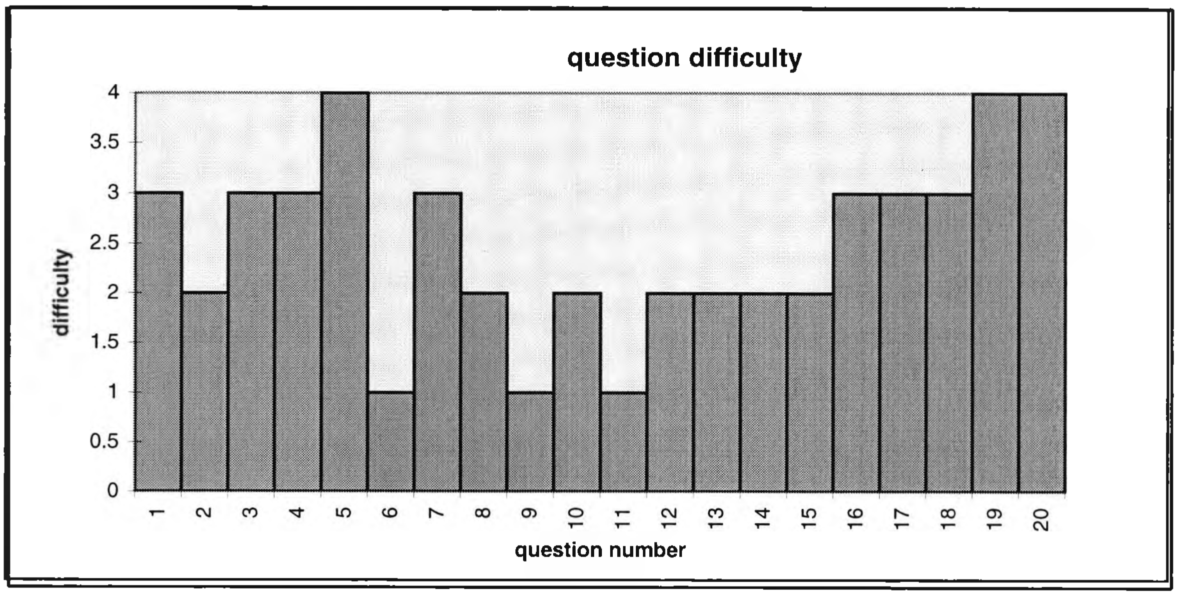


Figure 8.7 Showing the 4 point scale of difficulty assigned to each question

QU	difficulty	type
1	3	connecting model and spec
2	2	Z notation significance
3	3	connecting model and spec
4	3	connecting model and spec
5	4	writing Z
6	1	finding relevant sections
7	3	connecting model and spec
8	2	Z notation significance
9	1	finding relevant sections
10	2	Z notation significance
11	1	finding relevant sections
12	2	Z notation significance
13	2	Z notation significance
14	2	Z notation significance
15	2	Z notation significance
16	3	connecting model and spec
17	3	connecting model and spec
18	3	connecting model and spec
19	4	writing Z
20	4	writing Z

Table 8.6 Showing the classification of each question type.

The normalised scores for each question were calculated (that is the average scores of those attempting that question). This should take out the effect of those who ran out of time. Figure 8.8 shows a plot of the degree of difficulty with these normalised scores. The difficulty plot has been shifted up the vertical axis by one unit to enable a clearer view of the graph.

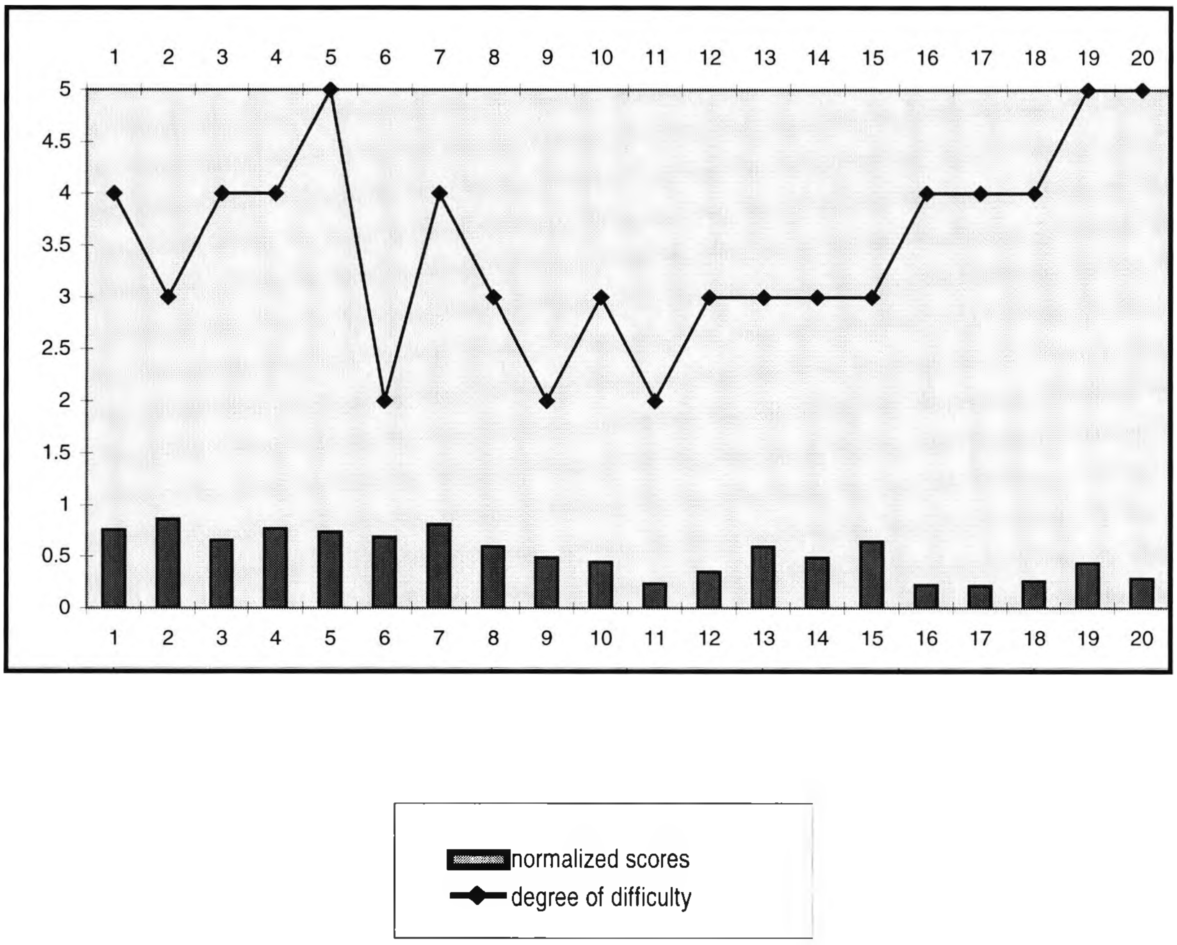


Figure 8.8 The normalised scores with the degree of difficulty

The expected interaction of these two plots would be low scores with high degrees of difficulty and vice versa. This does not seem to be the case and to investigate it further a scatter diagram was drawn to consider the correlation. From Figure 8.9 we can see that there is no obvious correlation.

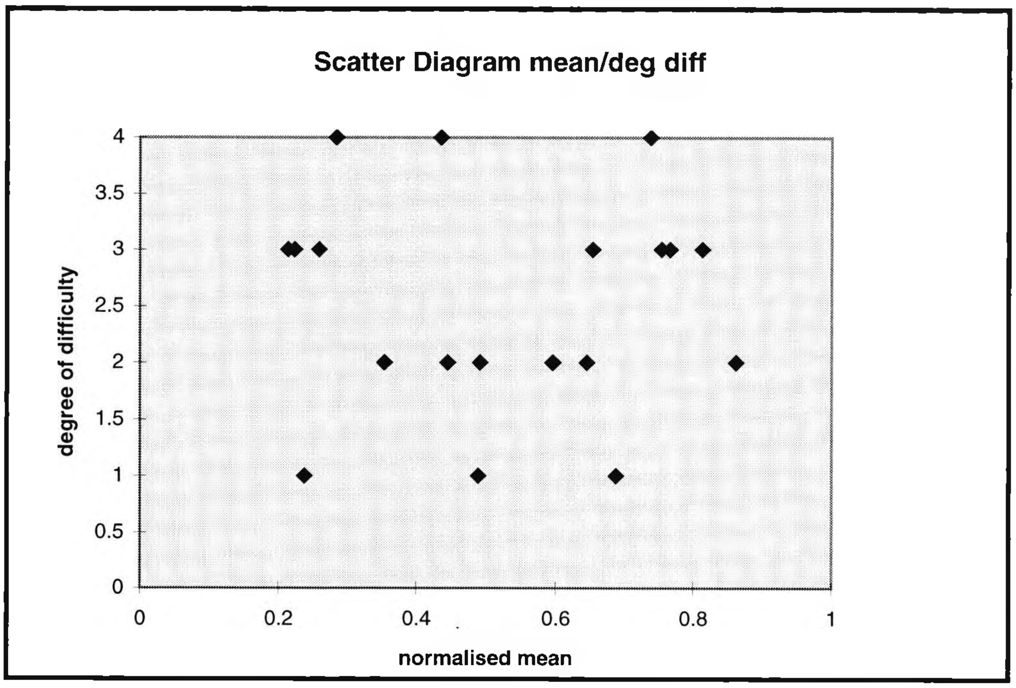


Figure 8.9 Plot showing poor correlation between scores and difficulty

8.3 Conclusions

We have shown through this last experiment that the structure of a formal specification in Z can affect its comprehensibility. We have measured the ease with which the subjects handle the specification with a combination of questions testing four key levels of comprehensibility:

- reading the specification,
- understanding specialised notation,
- relating the specification to the model,
- modifying the specification.

These have been shown to be improved when each part of the specification is kept to a length of about 20 lines. Larger unbroken structures are significantly worse in terms of comprehensibility. Breaking the specification down further does not make a worthwhile improvement.

We have seen confirmation that those subjects most able in terms of their academic background might well be those who can read the specification best.

These results have implications for the proponents of Formal Methods and those who would argue for their widescale adoption in industry. Some in the Formal Methods community have argued that only specialised groups of software engineers would need to be able to use Formal Methods to the extent that they could write specifications, but original writing is not one of the skills tested here. Specifications which can be written but not read by anyone other than the author are of limited use. If Formal Methods are to be used in industry on a wider scale than at present, the reading, understanding and modification skills tested here will be needed by a broad range of personnel including clients and programmers.

Clearly a good academic background is a help in reading the specifications but it should not be only the academic elite from the research departments or the specialised Formal Methods groups who can handle the specifications. Our results here give pointers to improving the quality of formal specifications with the wider audience in mind by adopting user friendly structural design.

From the results here and in the previous chapter, it should be possible to improve the quality of specifications in terms of their comprehension by:

- giving attention to the variable names, making them cues to further information and easily recognised shorthand for the objects which they refer to,
- improving commenting levels so that there is a high proportion of natural language providing explanation and information in addition to the more specialised notation of the Formal Method,
- keeping the structure to an optimum level so that the reader has a manageable amount of the specification to retain at a time. If it is too large there is an information overload; if it is too small there is too much fragmentation.

Timing data was unavailable in this experiment but could be incorporated in future work. It may be conjectured that the physical separation of a large number of small schemas would add considerably to the time taken to read a specification when compared with a more dense but compact structure.

Further analysis could also be carried out by isolating those components in the students' previous academic record which might seem to have a particular bearing on their abilities with Formal Methods. For instance at this level all students would have taken a first year mathematics module so that their performance in mathematics could be used for tests of correlation with their scores for reading Formal Methods.

Larger specifications could be used but there are many practical problems to overcome when conducting experiments on industrial sized specifications, not least the shortage of subjects with the time available.

This experiment represents an attempt to provide proper empirical evidence about specifications in Z. To make the case for or against Formal Methods we must stop relying on notions, feelings, biased opinions or unvalidated claims. Results such as these where the basis of the experiment is clear, the hypothesis stated and the statistical analysis given provide proper evidence with a firm scientific basis.

CHAPTER NINE

9. SUMMARY

In this chapter we summarise the work described in the thesis. We state again the hypotheses and place them in the context of existing research. We review the principal results and analyse the contribution they make to software engineering. Finally we suggest possible future directions and developments.

9.1 Introduction and overview

In this work our aim has been to investigate the application of software metrics to the area of formal specification. We focussed the work in our research hypotheses:

1. Formal Methods can be understood by any intelligent software developer with reasonable training,
2. the use of Formal Methods leads to improved software quality,
3. the structure of formal specifications impacts on their comprehensibility.
4. Formal Methods have not been used extensively in industry in realistic applications,
5. widespread take up of Formal Methods will occur only after the results from large scale case studies are published.

In Chapter 2 we first looked at the nature of Formal Methods and those most commonly in use. This suggested the scope for the type of specifications that we would later investigate. Metrics and measurement came under consideration in Chapter 3 as we sought to see what could be gained by looking at the work already done in software metrics. Concentrating on Formal Methods again in Chapters 4 and 5 we tried to establish current knowledge about their use and effectiveness. We investigated to see how their benefits had been assessed by surveys and case studies. Chapter 6 was based on work carried out in a previous study to investigate and compare attributes of three formal specification notations. This gave valuable insights into possible attributes, the way measurements might be made and possible metrics established. Drawing together the knowledge and experiences gained thus far experiments were designed to look at possible metrics for comprehensibility in formal specifications. The experiments, together with their results and statistical analysis were described in Chapters 7 and 8.

9.2 The context of the work

A recurrent theme of this work has been the claims and counter claims of Formal Methods. We believe that for these methods to be adopted or rejected by software engineers empirical work must be carried out to provide hard evidence. In software development as a whole much research effort has gone into trying to identify suitable metrics to capture software attributes such as complexity and information flow. Experiments have been conducted to validate suggested metrics and refinements have taken place in the light of the results. No similar work has been carried out in the area of formal specifications.

Proponents of Formal Methods claim that their use will improve the quality of the resultant software and yet with one or two honourable exceptions no attempt has been made to quantify this improvement. Worse still, some of the studies claiming to demonstrate this improvement are clearly flawed.

It was in this context that we set out to try and establish ways of assessing the contribution of Formal Methods to software development.

9.3 The contribution of this work to the field

We have shown that there is a plethora of opinions but very little scientific data to make the case for or against Formal Methods. We have tried to suggest ways to remedy this.

9.3.1 The surveys

To look at the current usage of Formal Methods we analysed two important surveys, one national and the other international. We considered the way the data was collected, the results given, their interpretation and the validity of the arguments put forward. Much of this data collected by survey and interview, we have shown to be subjective and the methods and interpretation we considered flawed.

9.3.2 The case studies

We considered case studies as another source of evidence and so looked for those projects in industry that had incorporated Formal Methods into their development. Few studies were available, few of these had made an attempt at quantifying the effect of Formal Methods. Those we have considered provided limited evidence and some of the claims seemed hardly justified by the data given. The results were mainly concerned with the final software product rather than intrinsic properties of the specifications themselves.

9.3.3 The experiments

We then focused on properties of the specifications themselves to see what attributes were important and whether we could construct specification metrics to measure them. We have made the first serious attempt to establish measurements on specifications themselves concerning the internal attributes which might have bearings on comprehensibility. Conducting three experiments, we looked at the effects of variable names, comments and structure on specifications written in Z. Our results gave pointers to the way these specifications should be written and warnings about the level of training assumed adequate for those who have to read them.

9.4 The limitations

With the time, money and opportunities available certain limitations were placed on the work:

- there was difficulty in obtaining unbiased first hand evidence of Formal Methods in practice. Papers and reports seemed to contain flaws, inaccuracies and inconsistencies that were difficult to check.
- no tool support was available to help with the production, checking and analysis of the Z specifications that formed the basis for the experiments.
- all experiments were conducted in an educational setting with computer science students as subjects. It has been suggested that this could be a limitation to the validity of the results as far as industry is concerned. However we believe that in all respects that are pertinent to the result there is no appreciable difference between our participants and practising software engineers.
- timing data (which might have given more information about the extra effort required to read badly structured specifications) was not available because of the constraints of operating within an academic timetable.

9.5 The achievements

We believe this thesis describes an original approach to the problems besetting the evaluation of the contribution of Formal Methods. We have questioned the basis of current opinion and argument. We have looked for the supporting evidence of the impact of Formal Methods in software development and challenged some of the results currently accepted. To investigate the nature of the specifications themselves we have tried to take the first steps in establishing empirically based data from well designed and conducted experiments. We believe this will enable future researchers to establish criteria for quality in specifications which may be linked to the quality of the software produced from them.

So far this work has resulted in several publications, reproduced in Appendix P.

9.6 Future research directions

There are several further areas to pursue following on from this work.

9.6.1 Additional metrics

We have concentrated on the factors affecting comprehensibility, reasoning that unless software engineers can read and understand formal specifications they will not be persuaded to adopt Formal Methods. Other possible areas for metrics could include;

- the effort required to refine the specification to code,
- the complexity of specifications perhaps related to layering approaches,
- the possibility of functional metrics along the lines of Function Points.

Including timing data as a factor in future experiments could also add to the assessment of complexity and comprehensibility.

9.6.2 Different specification notations

We have concentrated our experiments on writing specifications in Z. Other state-based notations could be used, such as VDM, and the results compared. Algebraic specification notations and those taking concurrency into account could also form a basis for further experiments.

9.6.3 Tool support

Due to a lack of resources no tool support was used in this work to write, check or analyse the formal specifications. There are now quite a few tools available that could be used to extend this work by analysing characteristics of formal specifications and refining them under prescribed conditions. This would extend the scope of the work by reducing the limitations imposed by writing, checking and testing everything by hand.

9.6.4 Industrial trials

Ideally the experiments that we have carried out on a student population should be replicated with practising software engineers to counteract charges against the validity of the results in commercial settings. The main drawback to this in practice is that it is hard to find a comparable number of software engineers with training in Formal Methods willing to participate. Again funding is an issue as trials in a commercial setting would have to be supported by grants or sponsored by interested industries.
Data could be obtained from specifications used in industry if more were made available in the public domain. Some automated analysis of the characteristics of large formal specifications could lead to further indications for metrics.

9.6.5 Links to software metrics

An exciting area of future research would be into the possible links between the metrics obtained from formal specifications and those available from the resulting software. We suspect that a specification which scores well in terms of the metrics we have applied will not necessarily lead to quality software. The two sets of metrics may not be related at all or may in fact be inversely related. To tailor the user requirements into a specification which scores well under specification metrics may impact badly on the attributes that form the basis of a measure of the quality of resulting software. It seems to have been borne out by anecdotal evidence that a specification which has been formally and very rigorously written becomes a very difficult document to use as a basis for a good implementation.

9.7 Conclusions

We feel that this thesis gives an account of an original piece of research that makes a valuable contribution to the debate on the role of Formal Methods in software development.

References

1. Adams E. Optimizing preventive service of software products. IBM Research Journal, 28(1) pp. 2-14. 1984
2. Abd-El-Hafiz SK and Basili VR. A Knowledge-Based Approach to the Analysis of Loops. Transactions on Software Engineering, 22 (5) pp. 339-360. 1996
3. Abran A and Robillard PN. Function Point Analysis: An Empirical Study Of Its Measurement Processes. IEEE Transactions on Software Engineering, 22 (12) pp. 895-910. 1996
4. Abrial J-R, Börger E and Langmaack H. Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control. Springer-Verlag. 1996
5. Abrial J-R. The Mathematical Construction of a Program. Science of Computer Programming, vol. 4 pp. 45-86. 1984
6. Abrial J-R. The B-book. Cambridge University Press. 1996
7. Agerholm S and Larsen PG. Modeling and Validating SAFER in VDM-SL. Proceedings of the Fourth NASA Langley Formal Methods Workshop, NASA Conference Publication 3356, September 1997
8. Albrecht AJ and Gaffney JE. Software function, source lines of code and development effort prediction: A software science validated. IEEE Transactions on Software Engineering, 9 (6) pp. 639-648. 1983
9. Albrecht AJ. Measuring application development productivity. Proceedings IBM Application Development Joint SHARE/GUIDE Symposium California, pp. 83-92. 1979
10. Almstrum VL. Limitations in the Understanding of Mathematical Logic by Novice Computer Science Students. Doctoral Dissertation, University of Texas at Austin. May 1994
11. Andrews DJ (ed). Information technology-Programming languages, their environments and systems software interfaces-Vienna Development Method-Specification Language Part 1: Base language ISO International Standard ISO/IEC 13817-1. December 1996
12. Ardis MA, Chaves JA, Jagadeesan LJ, Mataga P, Puchnol C, Staskauskas MG and von Olnhausen J. A Framework for Evaluating Specification Methods for Reactive Systems: Experience Report. IEEE Transactions on Software Engineering, 22(6) pp. 378-389. 1996
13. Arnold A, Bejay D and Radoux J-P. The Embedded Software of an Electricity Meter: An Experience in Using Formal Methods in an Industrial Project. Proceedings 5th International Conference Algebraic Methodology and Software Technology, Munich Germany eds Wirsing M and Nivat M, pp.19-32. 1996
14. Austin S and Parker GI. Formal Methods: A Survey. National Physical Laboratory report. March 1993
15. Austin S and Parker GI. Notes on +/- of Formal Methods. Unpublished paper, National Physical Laboratory DITC SEG E1 N6. June 1992
16. B-Toolkit from B-Core (UK) Ltd, Magdalen Centre The Oxford Science Park, Oxford, OX4 4GA, UK
17. Bainbridge J, Whitty R and Wordsworth J. Obtaining Structural Metrics of Z Specifications for Systems Development. Proceedings 5th Annual Z User Meeting 1990 Oxford, pp. 269-281. Springer-Verlag. 1991

18. Balke AC, Carter J and Haveman J. Experience using Formal Methods in High Energy Physics. International Journal of Modern Physics C, 6(4) pp. 469-473. 1995
19. Baber RL. Error-Free Software:Know-how and Know-how and Know-why of Program Correctness, Wiley. 1991
20. Barroca LM and McDermid JA. Formal methods: Use and Relevance for the development of Safety Critical Systems. The Computer Journal, 35 (6) pp. 579-599. 1992
21. Basili VR. Data Collection, Validation, and Analysis. In Tutorial on Models and Metrics for Software Management and Engineering. IEEE Computer Society Press. 1981
22. Basili VR. Quantitative Evaluation of Software Engineering Methodology. Proceedings of the First Pan Pacific Computer Conference, Melbourne, Australia, September 1985 .
23. Basili VR and Hutchens DH. An empirical study of a syntactic complexity family. IEEE Transactions on Software Engineering, 9(6) pp. 652-663. 1983
24. Basili VR and Rombach HD. The TAME project: Towards improvement-oriented software environments. IEEE Transactions on Software Engineering, 14(6) pp. 758-773. 1988
25. Basili VR and Selby RW Jr. Data Collection and Analysis in Software Research and Management. Proceedings of the American Statistical Association and Biomeasure Society. 1984
26. Basili VR, Selby RW and Phillips TY. Metric analysis and data across FORTRAN projects. IEEE Transactions on Software Engineering, 9(6) pp. 652-663. 1983
27. Basili VR, Selby RW and Hutchens DH.Experimentation in software engineering. IEEE Transactions on Software Engineering, 12(7) pp. 733-743. 1986
28. Basili VR and Weiss D. A methodology for collecting valid software engineering data. IEEE Transactions on Software Engineering, SE-10 (6), pp. 728-738. 1984
29. Basso M, Ciapessoni E, Crivelli E, Madrioli D, Morzenti A, Ratto E and SanPietro P. Experimenting a Logic Based Approach to the Specification and Design of the Control System of a Pondage PowerPlant. ICSE-17 Workshop on Industrial Applications of Formal Methods, Seattle, Washington. April 1995
30. Beiber P. Formal Techniques for an ITSEC-E4 Secure Gateway. Proceedings 12th Annual Computer Security Applications Conference. December 1996
31. Belady LA and Lehman MM. A model of large program development. IBM Systems Journal, 15(3) pp. 225-252. 1976
32. Berbard P and Lafitte G. The French Population Census for 1990. Proceedings 9th International Conference of Z User, Limerick, Ireland. Bowen JP and Hinchey MG eds pp. 269-281. Springer-Verlag. 1995
33. Bieman JM and Ott LM. Measuring functional cohesion. IEEE Transactions Software Engineering, 20(8) pp. 644-657. 1994
34. Bicarregui JC, Fitzgerald JS, Lindsay PA, Moore R and Ritchie B. Proof in VDM: A Practitioners Guide. Springer-Verlag, FACIT Series. 1994
35. Bicarregui JC (ed). Proof in VDM: case studies. Springer-Verlag, FACIT Series 1998
36. Bjørner D, Haxthausen AE and Havelund K. Formal, model-oriented software development methods: From VDM to ProCoS & from RAISE to LaCoS. Future Generation Computer Systems, 7 pp. 111-138. 1992

37. Bjørner D and Jones CB (eds). The Vienna Development Method: The Meta Language. Lecture Notes in Computer Science, vol 61. Springer Verlang, Heidleberg. 1978
38. Bjørner D and Jones CB. Formal Specification and Software Development. Series in Computer Science. Prentice-Hall International, Englewood Cliffs, NJ. 1982
39. Boehm BW. Software engineering Economics. Prentice-Hall New York. 1981
40. Booch G. Object Oriented Design with Applications. Redwood City,CA. Benjamin- Cummings 1991
41. Bowen JP and Hinchey MG. Ten Commandments of Formal Methods. IEEE Computer, 28 (4) pp. 56-63. 1995
42. Briand LC, Basili VR and Thomas WM. A Pattern Recognition Approach for Software Engineering Data Analysis. Transactions on Software Engineering,18 (11) pp. 931-942. 1992
43. Brien SM and Nicholls JE. Z base standard. Technical monograph PRG-107, Oxford. 1992
44. Brinksma E. What is the Formal in Formal Methods? In Formal Description Techniques IV, eds Parker KR and Rose GA. Elsevier Science. 1992
45. Britton C, Jones S, Myers M and Sharif M. An investigation into measurement of notations used in software modelling. Proceedings of European Software Cost Modelling Conference (ESCOM), Berlin. 1997
46. Brooks RE. Studying Programmer Behaviour Experimentally: The Problems of Proper Methodology. Communications of the ACM, 23 (4) pp. 207 – 213. April 1980
47. Bruns G and Anderson S. Gaining Assurance with Formal Methods. In Applications of Formal Methods. Hinchey MG and Bowen JP (eds) pp. 33-54. Springer- Verlag 1995
48. Burghes D. Recent Changes in School Curricula and their Effect on University Mathematics Courses: A View from Mathematics Education. The Bulletin of the IMA, 28(1) pp. 17-20. 1992
49. Chidamber SR and Kemerer CF. Toward a metric suite for object oriented design. IEEE Transactions on Software Engineering, 20(6) pp. 476-498. 1994
50. Cohen B. A rejustification of formal notations. Software Engineering Journal, 4(1) pp. 36-38. 1989
51. Cohen B, Harwood WT and Jackson MI. The Specification of Complex Systems. Addison Wesley. 1986
52. Collins BP, Nicholls JE and Sorensen.IH. Introducing Formal Methods: the CICS Experience with Z. Mathematical Structures for Software Engineering. Clarendon Press, Oxford, pp.153 –164. 1991
53. Conte SD, Dunsmore HD and Shen VY. Software Engineering Metrics and Models. Benjamin -Cummings, Menlo Park, California. 1986
54. Coombes A, Barrocca L, Fitzgerald JS, McDermid JA, Spencer L and Saeed A. Formal Specification of an Aerospace System: the Attitude Monitor. In Applications of Formal Methods. Hinchey MG and Bowen JP (eds), pp. 307-332. Springer-Verlag 1995
55. Computer. Special Issue devoted to Formal Methods, 29(4). 1996
56. Covington R. Formal methods specification and verification. Guidebook for software and computer systems Volume 1: Planning and Technology Insertion. NASA-GB-002-95 available from the web site http://www.ivv.nasa.gov.

57. Craigen D, Gerhart S and Ralston T. Formal methods reality check: industrial usage. IEEE Transactions on Software Engineering, 21(2) pp. 90-98. 1995
58. Craigen D, Gerhart S and Ralston T. An International Survey of Industrial Applications of Formal Methods. US National Institute of Standards and Technology, Technical Report NIST GCR 93/626, vol. 1. 1993
59. Curtis B. Measurement and experimentation in software engineering. Proceedings IEEE 68(9) pp. 1103-1119. 1980
60. Curtis B. By the way, did anyone study real programmers? Empirical Studies of Programmers, Soloway E and Iyengar S (eds). Ablex Publishing Corp NJ, pp. 256-262. 1986
61. Daly J, Brooks A, Miller J, Roper M and Wood M. Evaluating Inheritance Depth on the Maintainability of Object Oriented Software. Empirical Software Engineering 1 pp. 109-132. 1996
62. de Morgan RM, Hill ID and Wichman BA. Modified report on the algorithmic language ALGOL 60. Computer Journal 19 pp. 364-379. 1976
63. de Neumann B. The role of formal mathematics in the assurance of software. Proceeding of the Centre for Software Reliability, pp. 153-165. London 1989
64. Dehbonei B and Mejia. Formal Development of Safety Critical Software Systems in Railway Signalling. In Applications of Formal Methods. Hinchey MG and Bowen JP (eds) pp. 227-252. Springer-Verlag. 1995
65. DeMarco T. Structured Analysis and System Specification. Prentice-Hall Englewood Cliffs NJ. 1979
66. Dick J and Faivre A. Automating the generation and sequencing of test cases from model-basedspecifications. In J. C. P. Woodcock and P. G. Larsen (eds.), FME'93: Industrial-Strength Formal Methods. Springer-Verlag Lecture Notes in Computer Science, Volume 670, pp. 268-284. 1993
67. Dick J and Woods E. Lessons learned from rigorous system software development. Information and Software Technology, vol. 39 pp. 551- 560. 1997
68. Dijkstra EW. Guarded Commands, Nondeterminancy and Formal Derivation of Programs. Communications of the ACM, 18 (8) pp.453-457. 1975
69. Diller A. An Introduction to Formal Methods. John Wiley and Sons. 1990
70. DMS Design Validation Final Report. ESA 9558/91/NL/JG LOT-RP-1134-10-CRS issue 1. 1997
71. Dürr EH, Plat N and de Boer M. CombiCom: Tracking and Tracing Rail Traffic using VDM++. In Applications of Formal Methods. Hinchey MG and Bowen JP (eds) pp. 203-225. Springer-Verlag 1995
72. Dutertre B and Stavridou V. Formal Requirements Analysis of an Avionics Control System. IEEE Transactions on Software Engineering, 23(5) pp. 267-278. 1997
73. Ehrig,H, Mahr, B , Classen, I and Orjeas, F Introduction to Algebraic Specification parts 1and 2 : Formal Methods for Software Development Computer Journal 35(5) pp. 460-477. 1992
74. ESSDE European Space Software Development Environment: Advanced Methods and Tools. In ESSDE Final Report ESA 9598/91/NL/JG, LOT-RP-1134-10-CRS issue1. 1997
75. Fenton NE and Whitty RW. Axiomatic approach to software metrication through program decomposition. Computer Journal, 29(4) pp. 329-339. 1986
76. Fenton NE. Software Metrics: A Rigorous Approach Chapman Hall London 1991

77. Fenton NE. Software metrics: theory, tools and validation Software Engineering Journal, 5 (1) pp. 65-78. 1990
78. Fenton N. Software Measurement: A Necessary Scientific Basis. IEEE Transactions on Software Engineering, 20 (3)pp. 199-206. 1994
79. Fenton NE and Pfleeger. Software Metrics A Rigorous and Practical Approach (2nd Edition). International Thomspn Computer Press. 1996
80. Fenton NE, Pfleeger SL and Glass RL. Science and Substance: A challenge to Software Engineers. IEEE Software, 11(4) pp. 86-95. 1994
81. Fenton NE and Melton A. Deriving structurally based software measures. Journal Software Systems, vol.12 pp. 177-187. 1990
82. Finney K. An investigation into the use of formal notations in the specification of mathematical problems. MSc Project Report. South Bank University 1993
83. Finney K and Fenton N. Evaluating the Effectiveness of Z: The Claims Made About CICS and Where We Go From Here. Journal Systems Software, 29(11) 1996
84. Fitzgerald JS. Two Industrial Trials of Formal Specifications. Proceedings 5th International Conference on Algebraic Methodology and Software Technology. Munich Germany. Wirsing M and Nivat M (eds) pp. 1-8. 1996
85. Fitzgerald J and Larsen PG. Modelling Systems: Practical Tools and Techniques in Software Development. Cambridge University Press. 1998
86. Fitzgerald J, Larsen PG, Brookes T and Green M. Developing a Security-critical System using Formal and Conventional Methods. In Applications of Formal Methods. Hinchey MG and Bowen JP (eds) pp. 333-356. Springer Verlang. 1995
87. Fitzgerald JS, Brookes TM, Green MA and Larsen PG. Formal and Informal Specification of a Secure System Component: First results in a comparitive study. Formal Methods Europe '94, Industrial Benefits of Formal Methods, pp.35-44. Springer-Verlag. Berlin 1994
88. Fitzgerald JS and Jones CB. Proof in the analysis of a model of a tracking system. In Bicarregui JC (ed.) Proof in VDM: case studies. Springer-Verlag FACIT Series, pp.1-29. 1998
89. Formaliser User Guide Logica Cambridge Ltd April 1994
90. FME Applications database at
91. http://www.csr.ncl.ac.uk/projects/FME/InfRes/applications
92. Fraser MD, Kumar K and Vaishnavi VK. Strategies for Incorporating Formal Specifications in Software Development. Communications of the ACM, 37(10) pp. 74 –86. 1994
93. Fuchs NE. Specifications are (Preferably) Executible. Software Engineering Journal 7(5) pp.323-334. 1992
94. Galton A. Temporal Logics and their Applications. Academic Press London 1987.
95. Gaudel M-C. Testing can be formal too. Proceedings of TAPSOFT'95: Theory and Practice of Software Development. Springer-Verlag, pp. 82-96 1995,
96. Gerhart S, Craigen D and Ralston T. Observations on Industrial Practise Using Formal Methods. Proceedings 15th International Conference of Software Engineering. Baltimore MD. May 1993
97. Gilmore DJ. Methodological Issues in the Study of Programming. In Hoc JM, Green TRG, Samurcay R and Gimore DJ (eds). The Psychology of Programming. London Academic Press. 1990

98. Gilmore DJ. Does the notation matter? In Gilmore DJ, Winder R and Detienne F (eds). User-centred Requirements for Software Engineering Environments. Heidleberg Springer-Verlag. 1994
99. Goguen J and Tardo J. An introduction to OBJ: a language for writing and testing software specifications. Specification of Reliable Systems 1979
100. Green TRG. Programming as a Cognitive Activity. In Smith HT and Green TRG (eds) Human Interaction with Computers. London Academic Press, pp. 271-320. 1980
101. Guttag J. Abstract Data Types and the Development of Data Structures. Communications ACM, 20(6) pp. 396-404. 1977
102. Hackney J. The Internalisation of Symbolism. The Bulletin of the IMA 27(10) pp. 214-216. 1991
103. Hall JA. Seven Myths of Formal Methods. IEEE Software, 7 (5) pp. 11-19. 1990
104. Hall JA. Taking Z Seriously. Proceedings ZUM 97, 10th International Conference of Z-Users, eds Hinchey MG and Bowen JP, pp. 89-91. Springer Verlang. 1997
105. Hall JA. Using Formal Methods to develop an ATC Information System. IEEE Software 13(2) pp. 66-76. 1996
106. Halstead MH. Elements of Software Science. Elsevier N- Holland 1975
107. Hamer U and Peletska J. Z Applied to the A330/340 CIDS Cabin Communication System. In Applications of Formal Methods. Hinchey MG and Bowen JP (eds) pp. 253-284. Springer-Verlag. 1995
108. HamiltonV. The Use of Z within a Safety-critical Software System. In Applications of Formal Methods. Hinchey MG and Bowen JP (eds) pp. 357-374 Springer-Verlag. 1995
109. Harold FG. Experimental Evaluation of Program Quality Using External Metrics. Empirical Studies of Programmers. Soloway E and Iyengar S (eds). Ablex Publishing Corp NJ, pp. 153-167. 1986
110. Hatton L. Programming Languages and Safety-Related Systems. Proceedings Safety-Critical Systems Symposium. Springer-Verlag New York, pp. 48-64. 1995
111. Havelund K and Haxthausen AE. RAISE Language group, The Raise Specification Language: Prentice Hall 1992
112. Hayes I. Specification Case Studies. Prentice Hall 1987
113. Hayes I and Jones CB. Specification are not (necessarily) executable Software Engineering Journal 4(6) pp. 330-338. 1989
114. Henhapl W and Jones CB. A formal definition of Algol 60 as described in the 1875 Modified Report. In The Vienna develpment method Bjorner D and Jones CB (eds). Springer-Verlag. Lecture Notes in Computer Science 61 Berlin Heidleberg New York pp.305-336. 1978
115. Henry S and Kafura D. Software structure metrics based on information flow. IEEE Transaction on Software Engineering, 7(5) pp. 510-518. 1981
116. Hinchey MG and Bowen JP eds. Applications of Formal Methods Prentice Hall International 1995
117. Hinton PR. Statistics Explained. Routledge 1995
118. Hoare CAR. Maths adds safety to computer programs. New Scientist, pp. 53-56, September 1986
119. Hoare CAR. An overview of some formal methods for program design. Computer 20(9) pp. 85-91. 1987
120. Hoare CAR. Communicating Sequential Processes. Prentice Hall. 1985

121. Hoare J, Dick J, Neilson D and Sorensen I. Applying the B Technologies to CICS. Proceedings FME'94: Industrial Benefit and Advances in Formal Methods eds Gaudel MC and Woodcock J, pp. 74-84. Springer-Verlag. 1996
122. Hoare JP. Application of the B-Method to CICS. In Applications of Formal Methods. Hinchey MG and Bowen JP (eds) pp. 97-123. Springer-Verlag. 1995
123. Holloway CM and Butler RW. Impediments to Industrial Use of Formal Methods. Computer 29 (4) pp. 25-26. 1996
124. Holzman GJ. The Theory and Practise of Formal Methods: NewCoRe. Proceedings IFIP World Computer Congress. Hamburg, Germany. 1994
125. Hörcher H-M and Peleska J. Using formal specifications to support software testing, Software Quality Journal, pp. 309-327. 1995
126. Houstan I and King S. CICS Project Report: Experiences and results from the use of Z in IBM. Proceedings of the 4th International Symposium of VDM Europe Springer Verlang Vol 1: Conference Contribution pp. 588-596, 1991
127. Ince D. Software Quality Assurance-A student Introduction. McGraw Hill London 1995
128. Ince DC. An annotated bibliography of software metrics. ACM SIGPLAN Notices 25(8) pp. 15-23. 1990
129. INMOS Ltd. Occam Programming Manual. Prentice Hall International. 1984
130. ISO Information Technology Programming Languages -VDM-SL First Committee Draft Standard CD 13817-1. Doc.No.ISO/IEC JT-C1/SC22/WG19/N-20 Nov 1993 now superceded by [Andrews 1996]
131. ISO 9126. International Organisation for Standardisation Information technology - Software product evaluation - Quality characteristics and guide lines for their use. ISO/IEC IS 9126. 1991
132. ISTEC Information Technology Security Evaluation Criteria Office for Official Publications of the European Community, Luxembourg 1991
133. Jacky J, Risler R, Kalet I. and Wootton P. Clinical neutron therapy system, control system specification, Part I: System overview and hardware organisation. *Technical Report* 90-12-01, Radiation Oncology Department, University of Washington, Seattle ,WA December 1990
134. Jacky J, Risler R, Kalet I, Wootton P and Brossard S. Clinical neutron therapy system, control system specification, Part II: User operations. *Technical Report* 92-05-01, Radiation Oncology Department, University of Washington, Seattle,WA May 1992
135. Jacky J and Unger J. From Z to code: A graphical user interface for a radiation therapy machine. Proceedings ZUM 95 10th International Conference of Z-Users, eds Hinchey M.G. and Bowen J.P. pp. 315-333. Springer Verlang. 1995
136. Jacky J. Specifying a safety-critical control system in Z. IEEE Transactions on Software Engineering, 21(2) pp. 99-106. 1995
137. Jacky J, Patrick M and Unger J. Formal Specification of control software for a radiation therapy machine. Technical Report 95-12-01, Radiation Oncology Department, University of Washington, Seattle ,WA December 1995
138. Jacky J, Unger J, Patrick M, Reid D, and Risler R. Experience with Z Developing a Control Program for a Radiation Therapy Machine. Proceedings ZUM 97 10th International Conference of Z-Users, eds Hinchey MG, Bowen JP and Till D. pp. 317-328. Springer Verlang. 1997

139. Jacob RJK. Using Formal Specifications in the Design of a Human-Computer Interface in Software Specification Techniques. Gehani N and McGettrick AD (eds). pp. 209- 222. Addison Wesley 1986
140. Jeffrey DR, Low GC and Barnes MA. Comparison of Function Point Counting Techniques. IEEE Transactions on Software Engineering, 19(5) pp.529-532. 1993
141. Jensen H and Vairavan K. An experimental study of software metrics for real time software. IEEE Transactions on Software Engineering, 11(2) pp. 231-234. 1985
142. Jones CB. A short history of function and feature points. Int Function Point Users Group Conference Proceedings. 1988
143. Jones CB. Systematic Software Development Using VDM. Prentice-Hall International (2nd edition). 1990.
144. Jørgensen AH. A Methodology for Measuring the Readability and Modifiability of Computer Programs. BIT vol. 20 pp. 394-405. 1980
145. Kendal MG. Rank Correlation Methods. Charles Griffin London. 1948
146. Kitchenham BA, Pickard LM and Linkman, SJ. An Evaluation of some Design Metrics. Software Engineering Journal, 5(1), pp.50- 58. 1990
147. Krantz DH, Luce RD, Suppes P and Tversky A. Foundations of Measurement vol.1. Academic Press. 1971
148. Kroger F. Temporal Logic of Programming. Springer Verlag Heidleberg. 1987
149. Lafontaine C, Ledru Y and Schobbens P. Use of the B-Theorem Prover. Comm. ACM 43(5). 1991
150. Larsen PG and Plat N. Standards for Non-executable Specification Languages. The Computer Journal, 35(6) pp.567-573, 1992
151. Larsen PG Fitzgerald J and Brookes T. Applying Formal Specification in Industry. IEEE Software 13(3) pp. 48-56. 1996
152. Lind RK and Vairavan K. An experimental investigation of software metrics and their relationship to program effort. IEEE Transactions on Software Engineering 15(5) pp. 649-653. 1989
153. Liu S and Stavridou V. The practice of formal methods in safety critical systems. Journal Systems Software 28, pp.77-87. 1995
154. Lightfoot D. Formal Specification using Z. Macmillan. 1991
155. Loomes MJ and Vitner RJ. Formal Methods: no cure for Faulty Reasoning, University of Hertfordshire, Division of Computer Science Technical Report 265 September 1996
156. Macro A and Buxton J. The Craft of Software Engineering. Addison Wesley. 1987
157. Mandrioli D. Applying Research Results in the Industrial Environment: the Case of the TRIO Specification Language. Proceedings 5th International Conference Algebraic Methodology and Software Technology. Munich, Germany. Wirsing M and Nivat M (eds). pp33-42 1996
158. Martin JJ. Data Types and Data Structures. Prentice-Hall International 1986.
159. Mataga P and Zave P. A Formal Specification of Some Important 5ESS Features: Part III: Connections and Provisioning, Technical Report BL0112650-931001-24 AT&T Bell Laboratories USA 1993
160. Mataga P and Zave P. Multiparadigm Specification of an AT&T Switching System. In Applications of Formal Methods. Hinchey MG and Bowen JP (eds) pp. 375-398. Springer-Verlag 1995
161. May D and Shepherd DE. Formal Verification of the IMS 8000 microprocessor. Proceedings Electronic Design Automation, pp. 605-615. London UK. 1987

162. McCabe TJ. A Complexity Measure. IEEE Transactions on Software Engineering, 2(4) pp. 308-320. 1976
163. McCall JA, Richards PK and Walters GF. Factors in Software Quality RADC TR-77-396 1977. Vols I,II,III US Rome Air Development Center Reports NTIS AD/A-049 014, 015, 055 1977
164. Middleburg CA. VVSL: A Language for Structured VDM Specifications Formal Aspects of Computing, 1(1) pp.115-135. 1989
165. Milner RA. Calculus of Communicating Systems. Lecture Notes in Computer Science 92. Springer Verlag New York. 1980
166. Milner R. Communication and Concurrency. Prentice-Hall International Englewood Cliffs NJ. 1989
167. Milner R, Tofte M and Harper R. The Definition of standard ML. MIT Press Cambridge Mass. and London England. 1990
168. Minitab reference manual. Minitab Inc.3081 Enterprise Drive, State College PA 16801 USA
169. Mitchell R, Loomes M and Howse J. Structuring formal Specifications - a lesson relearned. Microprocessors and Microsystems, 18 (10) pp. 593-599. 1994
170. Naur P. Formalization in Program Development. BIT 22 pp.437-452. 1982
171. Naur P. Understanding Turing's Universal Machine---personal style in program description. Computer Journal, 36 (4) pp.3 51-371. 1993
172. Neil MD and Bache RM. Data linkage maps. Journal of Software Maintenance, 5(3). 1993
173. Neilson DS and Sørensen IH. The B-Technologies: A System for Computer Aided Programming. In Proceedings of the 6th Nordic Workshop on Programming Theory B-Core (UK) Ltd. BRICS Notes Series University of Aarhus Denmark pp. 18-35 1994
174. Parnas DL. Using Mathematical Models in the Inspection of Critical Software. In Applications of Formal Methods. Hinchey MG and Bowen JP (eds) pp. 17-31 Springer-Verlag 1995
175. Parnas D. On the criteria to be used in Decomposing Systems into Modules. Classics in Software Engineering. Yourdon e (ed). Yourdon Press New York 1979
176. Peterson I. Fatal Defect- Chasing Killer Computer Bugs. Vintage Books New York. 1996
177. Peterson J. Petri Nets. Computing Surveys, 9 (3) pp.223-252. 1977
178. Pfleeger SL and Hatton L, Investigating the influence of Formal Methods IEEE Computer 30(2) pp. 33-43 1997
179. Pfleeger SL, Page S, Fenton NE, Hatton L. Case study results for the SMARTIE project. Deliverable 2.2.4 (Additional Study), CSR, City University. March 1995.
180. Phillips M. CICS/ESA 3.1 Experiences. Proceedings of the 4th Annual Z User Meeting. Springer Verlang Workshops in Computing, pp. 179-185. 1989
181. Plat N and Larsen PG. An Overview of the ISO/VDM-SL Standard. SIGPLAN Notices 7(8) pp. 76-82. 1992
182. Potter B. Formal Methods: Needs Benefits and Pitfalls. Advanced Information Systems, pp. 205-210. 1991
183. Rann D, Turner J and Whitworth J. Z: A Beginner's Guide. Chapman and Hall London. 1994
184. Radcliffe B and Rollo AL. Adapting function point analysis to Jackson System Development. Software Engineering Journal, 5(1) pp. 79-84. 1990

185. Ritchie B, Bicarregui J and Haughton H. Experiences in Using the Abstract Machine Notation in a GKS Case Study. Proceedings FME'94:Industrial Benefit of Formal Methods eds Naftalin M, Denvir T and Bertran M pp. 93-104 Springer-Verlag 1994
186. Roche JM. Software Metrics and Measurement principles. ACM SIGSOFT Software Engineering notes, 19(1) pp. 77-85. 1994
187. Roy S. No Maths Please. The Bulletin of the IMA 28(1) pp. 26-27. 1992
188. Rushby J. Formal Methods and the Certification of Critical Systems. Technical Report CSL-93-7 Computer Science Lab, SRI International, Menlo Park, California, USA. 1993
189. Rushby J Formal Methods and their Role in the Certification of Critical Systems. Technical Report CSL-95-1 Computer Science Lab, SRI International, Menlo Park, California, USA. 1995
190. Saiedian H. (with contributions from 15 authors) An invitation to Formal Methods. Computer 29(4) pp. 16 – 30. 1996
191. Stavridou V. Formal Methods and VSLI Engineering Practise. The Computer Journal 37(2) pp. 96-113. 1994
192. Schneidewind NF. Methodology for validating software metrics. IEEE Transactions on Software Engineering, 18 (5) pp. 410-422. 1992
193. Shepherd DE. Verified microcode design. Microprocessors and Microsystems, vol. 14 pp. 623-630. 1990
194. Shepperd MJ and Ince DC. The use of metrics in the early detection of design errors. Proceedings Software Engineering, 90, pp. 67-85, 1990.
195. Shepperd MJ. Design Metrics: an empirical analysis. Software Engineering Journal, 5(1) pp. 3-10. 1990
196. Soloway E, Bonar J and Ehrlich K. Cognitive strategies and looping constructs: an empirical survey Comm ACM 26 (11) pp. 853-860 1983
197. Spivey JM. A specification Language and its Formal Semantics, vol 3 of Cambridge Tracts in Theoretical Computer Science Cambridge University Press 1988 (Published version of 1985 Dphil thesis)
198. Spivey JM. The Z notation: A reference manual. Series in Computer Science Prentice-Hall International, 2nd Edition. 1992
199. Spivey JM. The fUZZ Manual Computer Science Consultancy, 34 Westlands Grove, Stockton Lane, York YO3 0EF, UK 2nd edition. 1992
200. Srivas MK and Miller SP. Formal Verification of AAMP5 Microprocessor. . In Applications of Formal Methods. Hinchey MG and Bowen JP (eds) pp. 125-180 Springer-Verlag 1995
201. Storey A. A Specification Case Study using the B-Methodology. Software Testing , Verification and Reliability, vol 2 pp. 187-202. 1992
202. Sullo G and Williams M. Formal Where? Even if you aren't designing an airplane or a spaceship, formal methods can help you create your own defect free system.. Computer Language 9(7) pp. 51-58. 1992
203. Symons CR. Function Point Analysis: Difficulties and improvements. IEEE Transactions on Software Engineering, 14 (1) pp. 2-11. 1988
204. Takang AA, Grubb PA, MaCredie RD, The effects of comments and identifier names on program comprehensibility: an experimental investigation. Journal of Programming Languages 4 pp. 143-167. 1996.
205. Tenny T. Program Readability: Procedures Versus Comments. IEEE Transactions on Software Engineering, 14(9) pp. 1271-1279. 1988

206. van den Brand M , van Deursen A, Klint P , Klisener S and van der Meulen E Industrial Applications of ASF+ SDF. Proceedings 5th International Conference Algebraic Methodology and Software Technology, Munich, Germany. Wirsing M and Nivat M (eds) pp. 9-18. 1996
207. Ventouris KP and Pintelas PE. A Practical Assessment of Formal Specification Approaches for Data Abstractions. Journal Systems Software, vol. 17 pp. 169-188. 1992
208. VDM Specification Language- proto standard Technical Report, British Standards Institution BSI IST/5/19 March 1991 for full standard see [Andrews 1996]
209. Vinter RJ. A Review of Twenty Four Formal Specification Notations. University of Hertfordshire, Division of Computer Science Technical Report 240. February 1996. 1996a
210. Vinter RJ. Transfer of Logical Tendances to Formal Reasoning, University of Hertfordshire, Division of Computer Science Technical Report 252. July 1996. 1996b.
211. Vinter RJ, Loomes MJ and Kornbrot DE. Reasoning about Formal Software Specifications: An Initial Investigation. University of Hertfordshire, Division of Computer Science Technical Report 249. March 1996. 1996
212. Voss K. Using predicate/transition nets to modeland analyze distributed database systems. IEEE Transactions on Software Engineering, 6 (6) pp. 539-544. 1980
213. Ward NJ. The Rigorous Retrospective Static Analysis of the Sizewell 'B' Primary Protection System. Software Proceedings SAFECOMP'93, pp. 171-181. 1993
214. Wegner P. The Vienna definition language. ACM Comput. Surveys 4 pp. 5-63. 1972
215. Weissman L. A methodology for studying the psychological complexity of computer programs. Technical Report CSRG-37, University of Toronto. 1974
216. Weyuker EJ. Evaluating software complexity measures. IEEE Transactions on Software Engineering , 14(9) pp. 1357-1365. 1988
217. Whitty R. Structural Metrics for Z Specifications. Proceedings 4th Annual Z User Meeting 1989, pp. 186-191. Springer-Verlag. 1990
218. Whitty RW and Lockhart R. Structural metrics. ESPRIT 2 project COSMOS document GC/WP1/REP/7.3 Goldsmiths' College, London. 1990
219. Wing J. A study of 12 Specifications of the Library Problem. IEEE Software 5(4) pp. 66-76. 1988
220. Woodcock JCP. Structuring specifications in Z. Software Engineering Journal 3(1) pp. 51-66. 1989
221. Woodcock JCP. The Rudiments of Algorithmic Refinement. Computer Journal, 35(5) pp. 441-450. 1992
222. Wordsworth JB. Software Development With Z: A Practical Approach to Formal Methods. Software Engineering, Addison-Wesley. 1992
223. York Software Engineering The CADiZ User Guide YSE Report (details from yse@minster.york.ac.uk.) 9 July 1993
224. Yourdon E and Constantine LL. Structured Design. ,Prentice-Hall. 1979
225. Zave P and Jackson MA. Where do Operations Come From? A Multiparadigm Specification Technique. Draft available from the authors. 1994
226. Zave P and Mataga PA. Formal Specification of Some Important 5ESS Features: Part I : Overview, Technical Report BL0112610-931001-33 AT&T Bell Laboratories USA 1993

227. Zwanzig K. Handbook for Estimating Using Function Points. GUIDE project DP-1234 GUIDE Int. November 1984
228. Zweben SH, Edwards SH, Weide BW and Hollingsworth JE. The Effects of Layering and Encapsulation on Software Development Cost and Quality. IEEE Transactions on Software Engineering, 21(3), pp. 200-208. 1995

APPENDIX A A brief description of some of the most common software metrics

DeMarco / Bang metrics

These are now known as specification weight metrics and based on De Macro's structured analysis and design methods. They incorporate counts taken from both the data flow diagrams (DFD) and the entity relation diagrams (ER)that form part of his methods of software specification. [DeMarco 1979]

Function Bang Measure

This is based on the data flow diagram, the pictorial representation of the system. In this picture the flow of the data is modelled with its component pieces and all interfaces between the components shown. It looks like a series of connected circles where each circle represents a transformation performed and each connection follows the flow of data. The DFD is developed using a top down approach and each original process within a its circle is expanded layer by layer until no further breakdown of is possible.

The function bang measure is based on counting the functional primitives i.e. the number of circles when the model is fully expanded. This count is weighted according to the type of functional primitive and the number of data tokens it uses.

Data Bang Measure

This is based on the entity relationship model which is a diagram showing all entities involved in the specification with connections showing the relationships between them and distinguishing the type of relationship as one to one, one to many or many to many.

The data bang measure is based on the number of entities in the ER diagram and is weighted by looking at the number of relationships involving each entity.

McCabe Complexity/cyclomatic number

The complexity of software has generally been associated with

- the number of components
- their maturity
- the relationships between them

A large number of complexity measures have been suggested over recent years but McCabe's [McCabe 1976] is one of the best known. He defines the complexity of a program on the basis of the program's flowgraph, the graphical representation of the structure using directed graphs. In these graphs the nodes represent the program statements and where edges join 2 nodes, they represent a flow of control between the statements.

He defines the complexity as $v = e - n + 2p$

where e is the number of edges, n the number of nodes and p the number of connected components.

This number is derived from graph theory and measures the number of linearly independent 'walks' through the graph.

If $p = 1$ and predicate nodes have outdegree 2, then it can be shown that $v = \pi + 1$ where π is the number of predicates in the graph.

McCabe used empirical evidence to suggest that any module giving a value of v higher than 10 would cause problems.

Coupling and Cohesion

In a generalisation of McCabe, which looks at the internal structure of a single module, we can consider features of the design of the whole system which is composed of many modules. Two features of this modular design the coupling between modules and the cohesion within them are common features of the move towards object oriented design methods.

Coupling

This is a measure of the connections between each module and Fenton [Fenton 1991] mentions six different classifications.

1. No coupling where two modules are completely independent.
2. Data coupling where data is passed between modules.
3. Stamp coupling where a record type is shared.
4. Control coupling where a parameter is passed as a control.

5. Common coupling where there is shared global data .
6. Content coupling where one module has an impact on the content of another.

This is an ordered list. The further down the list the type of coupling is, the tighter the classification of coupling.

Fenton and Melton [Fenton and Melton 1990] propose a measure of cohesion between two modules x and y:

$c(x,y) = i + \frac{n}{n+1}$ where i is the worst coupling between them (1-6) and n is the number of interconnections between them .

They give a measure of the overall cohesion C of the system S which consists of n modules D_1 to D_n as

$C(S)$ = median value of the set $\{ c(D_i, D_j) : 1 \leq i < j \leq n\}$

Cohesion

There can be various definitions of cohesion but one of the most common is functional cohesion. This is a measure of the relatedness of the internal components so that a highly cohesive module has one basic function which is indivisible.

Yourdon and Constantine [Yourdon and Constantine 1979] proposed 7 classes of functional cohesion

1. Functional cohesion where the module carries out a single function.
2. Sequential cohesion where more than one function is performed by the module but in a prescribed order.
3. Communications cohesion where the module has a number of functions to perform on a single body of data.
4. Procedural cohesion where a number of functions are carried out related only to a general procedure (not sequentially).

5. Temporal cohesion has the functions of a module related by the time in which they must occur.
6. Logical cohesion where the relation between the functions performed by the module is one of logic.
7. Coincidental cohesion is where the functions contained within a module have no relationship to one another.

The order here (1-7) denotes the extent of the functional strength of the modules from 1 (the most desirable) to 7 (the least desirable).

Bieman and Ott [Bieman and Ott 1994] noted that using this model there was no scale for the cohesion so that a module was either functionally cohesive or not. They tried to develop quantitative measures of functional cohesion based on the slice abstraction of a program. based on data slices. In their paper they give the definition of a slice:

A slice of a procedure at statement s with respect to a variable v is the sequence of all statements and predicates that might affect the value of v at s.

This is modified to a *metric slice* when both the used and used by relationships are taken into account and then further refined to a *data slice* when the metric slice is applied to data tokens which they define as variable and constant definitions and references.

The other two aspects of their assessment of cohesion are *glue* and *super glue* to give a measure of *adhesiveness.* This looks at the number of slices that each data token appears in and classifies the tokens as superglue if they appear in every slice of a module, or glue if they appear in more than one but not every slice of a module.
Strong Functional Cohesion (SFC) and Weak Functional Cohesion (WFC) are definitions based on the relative numbers of superglue and glue tokens respectively.
In their example of a three slice abstraction their figures are given as

Tokens	11
Glue	5
Super glue	2

From their diagram 2 tokens occur on 3 slices and 3 on 2 so their calculations become

$$\text{WFC} = \frac{\text{glue tokens}}{\text{total tokens}} = 5/11 \qquad \text{SFC} = \frac{\text{super glue tokens}}{\text{total tokens}} = 2/11$$

$$\text{A} = \text{Adhesiveness} = \frac{\sum slices \times tokens}{tokens \times slices} = \frac{2\times 3 + 3\times 2}{11\times 3} = 12/33$$

They show WFC$\leq$A$\leq$SFC and conclude that Adhesiveness is the most sensitive measure of program modifications. It has the advantage that a lot of the computations could be automated and with the right tool support the information on slices and tokens could be generated .

There is no need for any subjective judgement as tokens are clearly defined and implicated in slices by clear definition.

Macro and Buxton [Macro and Buxton 1987] looked at abstract or data cohesion. This was an extension of functional cohesion to the field of abstract data types, But instead of concentrating on functional cohesion it looked at the cohesion of data.

Henry and Kafura's information flow measures

This is based on the flow of data between modules and was one of the earliest design metrics [Henry and Kafura 1981].

To understand the calculation of their metric several terms to do with the information flow through the system must be defined:

- local direct flow: when two modules pass data directly between them,
- local indirect flow: when data is passed through a module onto another,
- global flow: flow of data between two modules via a global data structure.

Fan in and fan out are properties of the modules.

- Fan in of a module: number of local flows terminating there + number of data structures used by the module.
- Fan out of a module: number of local flows starting there + number of data structures modified by the module.

This gives the information flow measurement on a module:

$$\text{Information Flow Complexity} = \text{Length} \times (\text{Fan in} \times \text{Fan Out})^2$$

and the total information flow complexity is the sum of these measurements over all modules.

Shepperd's refinement to Information Flow Metric

Shepperd refined the measures used by Henry and Kafura by tightening the definitions of some types of data flow [Shepperd and Ince 1990]. Their main change was to ignore the module length in the formula as it meant the metric could be applied at the design stage before coding information was available . This Shepperd names IF0 so:

$$IF0 = (\text{Fan In} \times \text{Fan Out})^2$$

Problems that arise with information flow metrics are mainly to do with the definition of the data flows and the way that they are counted. In general Shepperd and Ince proposed that a lot of the repeated calls and duplicate flows should only be counted once. They felt that these refinements distinguished better between data flow which was for control purposes and that which was passing information. In his experimental work Shepperd [Shepperd 1990] uses two more refinements of the Henry and Kafura metric which he labels IF3 and IF4.

$$IF3 = (\text{fan\_in} \times \text{fan\_out})^2$$

In this metric he is using his own definition of Fan in and Fan out

fan_in : the total number of local and global flows terminating at a module

fan_out : the total number of local and global flows emanating from a module

The refinement for IF4 is to consider the uniqueness of the flows so that

$$\text{IF4} = (\text{unique\_fan\_in} \times \text{unique\_fan\_out})^2$$

unique_fan_in : the total number of local and global unique flows terminating at a module

unique_fan_out : the total number of local and global unique flows emanating from a module

All these measures are computed module by module and then summed over all modules.

He notes his uneasiness with the modules which make no contribution to the overall total because they have a zero value for either fan-in or fan-out. He concludes that these must be sources or sinks for information and that these must add little to the complexity but that it would need further investigation.

Band width

This is a simple measure of the level of nesting in a program which could also be used from a design stage if there was a flow chart of control. Band Width (BW) is given as an indicator of the average level of nesting of a program .

$$\text{BW} = \frac{i \times L(i)}{n} \text{ where}$$

i is the level , L(i) is the number of nodes at level i and n = number of nodes in the graph.

So a straight line program results in a band width of 1 but a deeply nested one has a higher BW value.
In their work Lind and Vairavan found this metric to be poorly correlated with development effort [Lind and Vairavan 1989].

Halstead Software Science metric

This is a size metric developed by Maurice Halstead in the early 1970's to predict size of code; he also derives an expression for the effort required to generate the program [Halstead 1975]. Evaluating the formulas involves counting the number of unique operands and operators as well as the total number of each. Using these figures Halstead derives several relationships including the predicted length, a number representing the maximum length of the program. Any program exceeding this length would be susceptible to features which may include ambiguity, redundancy in expressions or unfactored expressions.

Let a = number of unique operators and N_a the number of occurrences of these operators, and b = number of unique operands and N_b the number of occurrences of these operands; then the estimated program length is given by

$a\log_2 N_a + b\log_2 N_b$

then Halstead defines the			
	estimated length	:	$N = N_a + N_b$
	vocabulary	:	$c = a+b$
	volume	:	$V = N\log_2 c$
	estimated level	:	$L = \frac{2}{a} + \frac{b}{N_b}$
	estimated difficulty	:	$D = \frac{1}{L}$

The effort required to generate the program is given by E where

$$E = \frac{aN_bN\log c}{2b}$$

This effort is measured in numbers of elementary mental discriminations. To convert this to a time scale the Stroud number is used, which is named after a psychiatrist who estimated that humans can make up to 20 mental discriminations per second.

In Halstead's examples he uses 18 as the divisor of E, finally giving a time in seconds.

Jenson program length

Jensen [Jensen and Vairavan 1985] refined the estimated program length by using
$\log_2 a! + \log_2 b!$ rather than $a\log_2 N_a + b\log_2 N_2$

In recent years there has been less support for some of Halsteads theories and in their paper Lind and Vairavan conclude that cruder measures such as total lines in code were some of the best to correlate to development effort [Lind and Vairavan 1989].

Function Points

This is intended as a measure of product size that can be derived as early as the specification stage since they deal with data sources, inputs and outputs rather than coding details. However it can also be used at later stages of the development when more accurate detail can be incorporated into the factors. They were first described in the late 70's by Allen Albrecht [Albrecht 1979] and were subsequently revised by him and others. [Albrecht and Gaffney 1983], [Symons 1988]. Albrecht proposed them initially as measures to:

- study factors affecting productivity , taking the size of the system in isolation from its environmental factors,
- give a technology independent view of the system from the user's point of view,
- aid estimation by being easy to obtain early in the lifecycle,
- be understood by non-technical users.

The Function Point count

This is based on a system of categories:

External Inputs
External Outputs
External Inquiries
External Files
Internal Files

After counting each category a weight is assigned to it according to a 3 point scale:

	simple	average	complex
External Inputs	3	4	6
External Outputs	4	5	7
External Inquiries	3	4	6
External Files	7	10	15
Internal Files	5	7	10

The unadjusted function count UFC is given by the formula

$$\sum_{i=1}^{15} \text{(number of items of type i)} \times \text{(weight of i)}$$

This sum allows for all 15 possible categories

e.g. given a small system with 4 simple inputs, 5 average inputs, 2 complex outputs, 5 average external inquiries and 1 simple external file and 2 complex internal files would give

$UFC = 4\times3 + 5\times4 + 2\times7 + 5\times4 + 1\times7 + 2\times10 = 93$

The function point FP is derived by multiplying the UFC by a technical complexity factor TCF

Factors contributing to TCF -technical complexity factor

F1	Reliable back up and recovery	F2	Data Communications
F3	Distributed functions	F4	Performance
F5	Heavily used configuration	F6	On- line data entry
F7	Operational ease	F8	On-line updating
F9	Complex interface	F10	Complex processing
F11	Reusability	F12	Installation ease
F13	Multiple sites	F14	Facilitate change

Each of the above factors is given a rating 0,1,2,3,4,5, with

0 Irrelevant, not present or no influence

1 Insignificant influence

2 Moderate influence

3 Average influence

4 Significant influence

5 Strong influence, throughout, essential

then the $TCF = 0.65 + 0.01 \sum_{1}^{14} F_i$

his gives a range of TCF from 0.65 when all 14 factors are rated irrelevant to 1.35 when all are given the rating essential.

e.g. in our above example if we decided that this system had only 2 essential factors reusability and operational ease and nothing else mattered (an unlikely scenario) then

$TCF = 0.65 + 0.01(10) = 0.75$

Finally this gives a FP in our example of $93 \times 0.75 = 69.75$

Issues

It can be seen that although it would be easy to count categories, the distinction of internal and external is not always clear cut and assigning values to each category could be difficult. The definitions of simple, average and complex might be open to wide interpretation which would affect the total UFC although these distinctions are based on factors such as the number of data elements in each type. The range allowed between these categories also does not allow very much differentiation so for example an increase of four in data elements in an external input may still only double the value given to it as simple but regarding the initial four as complex would have the same effect. Similar problems occur when assigning ratings to the fourteen technical complexity facors.

It is easy to see that two different assessors of a project could end up with very different Function Point answers. The figure for the answer cannot be derived until a full specification is in place; the user requirements documentation is insufficient.

By comparison with a finished system the estimates can be out from 400 to 2000%. This is mainly due to a coarse initial view in comparison to a final product with extra complexity and enhanced functions appearing .

Subjectivity means that the estimation cannot be fully automated. There can also be confusion over the distinction between inputs and enquiries.

The counting is not independent of the system analysis and design method used so it is reliant on the style of design. It has been used successfully for data processing applications that have low procedural complexity but its use in real time and scientific applications has been controversial. This is mainly because it does not deal with internal complexity or complex mathematical algorithms but concentrates on external functionality.

The GUIDE project tried to clarify the detailed rules for counting the FP to make it easier to apply but did not alter the underlying weaknesses involved because of the weighting [Zwanzig 1984].

Mark 2 Function points

These were developed by Symons as an alternative way of calculating function points to overcome the weaknesses he lists in the paper [Symons 1988]. The main area he was concerned with was the validity of the method for general application.

His Unadjusted Function Points (UFP) are given by:

$UFP = N_I W_I + N_E W_E + N_O W_O$ where

N_I = number of input data elements

W_I = weight of an input data element type

N_E = number of entity type references

W_E = weight of an entity type reference

N_O = number of output data elements

W_O = weight of output data element type

and N_I ,N_E and N_O are summed over all transaction types.

He calibrated his weightings by using practical data from two clients who supplied 6 systems each. Having arrived at his W values he scaled them so that they would be comparable to Albrecht's for values of UFP < 500.

This gives

$UFP = 0.44\ N_I + 1.67 N_E + 0.38 N_O$

He then modifies the Technical Complexity factor by using

$TCF = 0.65\,(1+\frac{Y}{X})$ where

Y = man hours devoted to technical complexity factors

X = man hours devoted to information processing size as defined by FP

He notes that comparing this with Albrecht's TCF, the weighting should take into account the technology involved with each project. The original work was based on the projects around in the late 1970's and his main conclusion is that weightings should be sensitive to the type of technology involved in each project and cannot be routinely applied across different application areas.

The following merasurements of LOC, Development Effort, Cocomo and other cost/effort predictors can only be used on the basis of lines of code written so are applied at a much later stage of development.

LOC

Lines of code, a seemingly simple measurement of source code program length has been one of the most common metrics used. However it is open to many interpretations, Jones has identified 11 main variations in ways of defining LOC [Jones 1988]. They differ most in the handling of:

- comments (extra lines added in natural language to help explanation but which are not essential to the running of the software),
- declaration lines (declaring data types or procedure headings),
- empty lines (added to give clarity to the presentation),
- artificial line breaks (writing x=3 y=4 on separate new lines for better style).

This can lead to a difference in LOC for 2 similar programs which may be almost entirely accounted for by the style and layout of the programmers rather than the nature of the program. It is fairly easy to eliminate blank lines and comments in counting as the latter must have some type of marker to avoid being considered as code by the compiler. Using these tighter definitions of LOC have led to NCLOC (non commented lines of code) also sometimes known as ELOC(effective lines of code). The most widely accepted model just uses a simple code listing with comments and blanks removed.

The drawback of using this last definition, as Fenton points out, is that information about the proportion of total lines of code made up from comments is lost [Fenton 1991]. When studying issues of comprehensibility this information is vital. Some first year programming courses even insist on a fixed commenting percentage (40% has been known) to encourage new programmers to explain the workings of their programs.

To overcome this loss of data Fenton suggests recording both separately:
Total Length (LOC) = NCLOC + CLOC
where CLOC is defined as commented lines of code.
And giving, incidentally, another metric concerned with the commenting level by considering the fraction $\frac{\text{CLOC}}{\text{LOC}}$

You can change to a language dependant LOC from a function point count by using an expansion factor for that language.

As an example in Cobol you might use factors 110 lines / function point whereas it might be 330 lines of assembler code. So a system with 50 FP will code up to 5500 lines of Cobol and 16500 lines of assembler.

This is an attractive proposition for systems written in 4GL languages because LOC can be difficult to use in systems suited to 4GL as they have more emphasis on external rather than internal functionality.

Development Effort

Metrics based on counting lines of code have been the simplest and most traditional used for estimating software development effort and measuring productivity. However as we have seen LOC is not a straightforward metric to define.

Effort estimation usually comprises of two parts, the first being a base estimate as a function of software size. This is usually given in man months the form

Effort = A + B $(KLOC)^C$ where various values of A, B and C have been proposed and KLOC is the abbreviation for thousands of lines of code.

The second part of the estimation model usually modifies the first estimate by taking into account the effect of environmental factors such as hardware and personnel.

Conte et al [Conte et al. 1986] give some of the most common of these models including

Watson - Felix	A = 0	B= 5.2	C = 0.91	
Bailey-Basili	A = 5.5	B= 0.73	C = 1.16	
Doty	A = 0	B= 5.288	C = 1.047	For KLOC>9
Boehm Simple	A = 0	B= 3.2	C = 1.05	
Boehm Average	A = 0	B= 3.0	C = 1.12	
Boehm Complex	A = 0	B= 2.8	C = 1.20	

These last three have become the basis for the COCOMO model.

COCOMO

Short for the Constructive Cost Model and proposed by Boehm [Boehm 1981], this is a widely used cost estimation model developed with information from applications written mainly in assembly, Fortran PL/I and Cobol.

There are three Cocomo models: basic, intermediate and detailed, each of which can be used at a different stage of the software cycle.

The basic model can be used for initial estimates and leaves out the factors attributed to cost drivers. The intermediate model is used when major components have been identified and the detailed model used when the design has got to the level of identifying individual components.

The formula used to find effort depends on a number of factors.

Cost drivers

There are 15 cost drivers incorporated into Cocomo and default values are given for each, but they should be adjusted to fit the circumstances of the particular project.

They refer to four categories

3 Product attributes (reliability, complexity, database size)

5 Personnel attributes (experience and capability of analysts and programmers)

4 Computer attributes (execution time, storage, virtual machine volatility, turnaround time)

3 Project attributes (programming practises, use of tools, development schedule)

Each cost driver is given a level of importance which must be estimated on a six point ordinal scale very low, low, nominal high, very high, extra high

and an adjustment factor is assigned to each of these points of the scale. For all factors the nominal value is 1.

Modes of development

There are 3 modes of development which will affect the effort formula:

organic (small to medium in-house data processing projects),

embedded (ambitious and tightly constrained projects),

semi-detached (somewhere between the two).

The effort estimation used by Cocomo is given in man months as

Effort = $a(\text{size})^b \times$ (product of cost drivers)

where size is the number of source instructions measures in thousands (usually equated with LOC) and a and b depend on the mode of development.

Organic	$a = 2.4$	$b = 1.05$
Embedded	$a = 3.6$	$b = 1.20$
Semi-detached	$a = 3.0$	$b = 1.12$

As well as being used as an estimation of effort Boehm uses the same equation with different values of a and b to give estimates of duration. It is intended to give the best estimate of the project duration given the effort.

The equation is adapted for duration by fixing a at 2.5 and varying b:

Organic	$b = 0.38$
Embedded	$b = 0.35$
Semi-detached	$b = 0.32$

Cocomo is a mixture of prescribed values, i.e. those for a and b and estimated ones. It relies on the weighting being assigned correctly to each product driver after assessments of the importance are made. In this respect it has the drawbacks of Function Points, relying on a subjective judgement to decide whether, for example, the analyst capability on a particular project should be rated as of extra high importance or just very high.

APPENDIX B Validation studies of software metrics

Lind and Vairavan

This study had two main aims; the authors were interested in the relationship of metrics to software development effort but also in the comparison of different metrics[Lind and Vairavan 1989]. They compared 11 different metrics:

Total Lines (Including comments)

Code Lines

Total Characters

Comments

Comment Characters

Halstead's N

Halstead's N_H

Jenson's N_J

McCabe's MC

Bandwidth BW

The software they used for the comparison was a large medical imaging system. It consisted of 4500 routines and 400 000 LOC of which 58% was in Pascal and 29% in Fortran. In the first part of the experiment a sample of 390 routines were used as a random sample. In the second part the System Performance reports were analysed with relation to a sample of 10 software features which together accounted for just over 1000 routines.

They found no major differences in the results and conclusions that could be drawn from these two approaches.

Summary

High comment level could indicate major future development effort as programmers tend to put more comments in to difficult code.

The simple LOC correlated to the development effort just as well or better than the more complex metrics. The other two well correlated metrics were MC complexity and Halstead's program length.

Defect density (program changes per 100 lines of code) changes in relation to LOC, declining to a minimum and then increasing; it seems to be related to the MC value.

Schneidewind

Schneidewind tested a metrics validation methodology using six criteria: association, consistency, discriminative power, tracking, predictability and repeatability [Schneidewind 1992]. These were chosen to support the quality functions of assessment, control and prediction. He tested his metrics on a set of 4 Pascal programs whose application areas were string processing, data base management and 2 directed graph analysis projects.

He looked at a total of 112 procedures and included 1600 source statements in his analysis. He aimed to see if the cyclomatic number (a combination of complexity C and size S) could be used to control reliability as measured by a factor error count E.

Summary

He concluded that C and S are valid with respect to his 'Discriminative Power Criterion' and could be used to distinguish quality. He notes that, as they are strongly correlated, only one needs to be collected and recommends that a size metric be used for economy reasons.

Kitchenham, Pickard and Linkman

In their study of 226 programs the aim was to look at the relationships between a set of design metrics and their ability to identify change prone, error prone and/or complex programs [Kitchenham, Pickard and Linkman 1990]. The set were compared with simple code metrics to assess their usefulness. To measure complexity a subjective test was used rather than a 'side effect' measure such as observed faults. The data was collected manually.

The quality indicator metrics that were collected from design and code were:

- information fan out (IFO),
- information fan in (IFI),
- information flow complexity (IFC),

- lines of code (LEN),
- control flow in branches (CF),

The quality characteristic metrics that would be used to assess the performance of the first set were:

- number of known errors (KE),
- number of planned changes (CHNG),
- subjective complexity (SC),

Summary

They found that

- IFO was correlated to the quality characteristic metrics but IFI was not.
- Code metrics had greater correlation with KE and SC than IFO.
- Overall, although large values of IFO, CF and LEN coincide with high values of KE, CHNG and SC it is a weak relationship.
- Large IFI values were related to small KE and CHNG.
- General correlation is weaker for programs with high values of CHNG than those with high values of KE.

So they did not find the information flow metrics significantly better as indicators over the simple code metrics. However some of the findings did indicate that judicial use of development effort to concentrate on those programs with high IFO values might be efficient and effective.

Harold

The principal aim of this experimental work was to look at the impact of structured programming techniques on program quality [Harold 1986]. In order to measure the quality of the programs Harold developed a set of metrics and refined them after comments and suggestions from a panel of experts (Baker, Basili, Boehm, Gilb and Wienberg). He aimed to make them free of structured programming bias and his final list of attributes together with metrics were :

Readability
Comments
Data and procedure names
Sequential flow of logic
Module size
Indentation
Logical Simplicity
Modifiability
Program modularity
Logical linkages
Program size
Empirical Modifiability
Verifiability
Satisfaction of Specification
Debugging difficulty

He gave very specific ways of obtaining each of these metrics and went to considerable lengths to train the assessors. He tested 122 programs which had each been assessed 3 times giving 366 metric score sheets; 16 variables were evaluated.

Summary

He found his metrics adequate to assess program quality but thought that a reduced number might yield equally good results.

Chidamber and Kemerer

In their paper the authors aimed to look in particular at metrics which could be applied to object oriented design [Chidamber and Kemerer 1994]. They had three objectives in mind:

- to propose metrics incorporating the experiences of software engineers,
- to test these against established criteria for validity,
- to obtain empirical data from commercial products to illustrate the use of the metrics on real applications.

They designed metrics to capture the complexity of the classes in an object oriented design. They did not apply them to the dynamic behaviour of the system. They validated their metrics against a subset of six of Weyuker's original nine properties [Weyuker 1988]. The explanations are in terms of classes P,Q and R and a metric m where P+Q stands for the combination of the two classes.

Noncoarseness	($\forall P\ \exists Q \ni mP \neq mQ$)
Non uniqueness	(it is possible for some P and Q that $mP = mQ$)
Design details are important	(same functionality does not imply equal m values)
Monotonicity	($mP< m(P+Q)$ and $mQ<m(P+Q)$)
Non equivalence of interaction	($mP = mQ \not\Rightarrow m(P+R) = m(Q+R)$)
Interaction increases complexity	($\exists P, \exists Q \ni mP+mQ < m(P+Q)$)

The proposed metrics were

Weighted methods per class	WMC
Depth of inheritance tree	DIT
Number of children	NOC
Coupling between object classes	CBO
Response for a class	RFC
Lack of cohesion in methods	LCOM

In relation to Booch's OOD Steps [Booch 1991] they note that these metrics only apply to the first 3 stages and give a table showing their position his OOD steps.

Metric	Identification	Semantics	Relationships
WMC	x	x	
DIT	x		
NOC	x		
CBO		x	x
RFC			x
LCOM		x	

Data from two sources was used to validate the metrics. The first source consisted of two C++ libraries consisting of 634 classes used in the design of graphical user interfaces and the second was 1459 classes from the libraries used in a CAD application for the production of VSLI circuits.

Summary

They found that there were only a few exceptions when their metrics were tested against Weyuker's criterion. Property 6 was not met by any of them and they argued that it could imply that their complexity metric could increase rather than reduce as a class is subdivided.
Property 4 was not satisfied by DIT and, under certain circumstances, LCOM failed too.

They saw their metrics as an aid to the management and design of projects and as a predictive tool. Using WMC , DIT and NOC they could judge whether the application was top heavy and RFC and CBO could keep a check on the interconnections.

They suggested tracking the metrics through the life of the project (as they will change as implementation takes place) and correlating them with some of the managerial performance indicators.

Shepperd

Shepperd aimed to look at design metrics and identify those which could be used diagnostically to pinpoint weaknesses with a view to minimising effort [Shepperd 1990]. He considered 8 metrics:

M1	Number of modules
M2	Program Size ELOC
M3	Module global data structure reads
M4	Module global data structure writes
M5	Indirect unique flows (H&K)
M6	Number of duplicate information flows (IF3)
M7	Number of unique information flows (IF4)
M8	Connect time (a measure of development effort)

He wanted to find out if the size metrics (M1-M4) and the structural metrics (M5-M7) are good predictors of development effort as measured by M8, the time students spent at the computer.

The metrics were tested on a subset of 27 versions of the same software written by student teams from the same specification. In size the programs varied from 313 ELOC to over 2000 ELOC but Shepperd eliminated those at either end of the spectrum, i.e. those which contained no error handling and those which added a lot of unspecified additional features, so that a subset of 13 programs constructed by 50 students was used for the testing . These had between 14 and 33 modules.

Summary

He found that size metrics were weakly associated with development effort but that IF4 (the Henry and Kafura metric adapted to consider unique information flows) gave a strong association with effort. However the size of the project and the teams associated with them may have had a bearing on these results and Shepperd acknowledges that this might be the case.

Radcliffe and Rollo

In this work an attempt was made to adapt the use of function point analysis (FPA) to the context of Jackson Systems development (JSD) [Radcliffe and Rollo 1990]. They gave their interpretation of the counting rules from two versions of FPA , those of Albrecht and Gaffney and by Symons. [Albrecht and Gaffney 1983, Symons 1988] as mapped to its interpretation in JSD.

They tested their metrics on a JSD based pensions project carried out by the Abbey National Building Society. Cobol was the target language and Effort and ELOC were calculated using Albrecht and Gaffney formulas. The results are given in the table below

Adaptation of	FC(UFP)	CAF	Adjusted FP	EFFORT work hrs	KSLOC Cobol
Albrecht and Gaffney	1132	0.84	951	37445	106.4
Symons	1009	0.94	948	37293	106

They found that the estimates of Effort and LOC were considerably higher than the actual figures for the project which were 16000 hrs and 57 KSLOC. They attribute this overestimate partly to the automated tool support available on the project. This meant that Cobol code was generated automatically affecting both the effort and also the style of code. With automatic generation the code is considerably more compact and so will have an impact on the KSLOC figure.

However they do cite their previous study when a similar reduction from 26KSLOC to 12-14 KSLOC was estimated .

Summary

The FP methods were adapted to JSD with mixed results. As a predictor of size and effort the two adapted methods seemed to overestimate both the size and the effort by a factor of two. They suggest further adaptations to the FP method to make it more accurate.

Abran and Robillard

This study looked at the validity of Function Point Analysis by considering intermediate measurement of some of the components that make up the metric and the implicit relationships behind them [Abran and Robillard 1996].

They tested 23 of the independent variables on a historic database of 37 projects of a finance organisation. These 37 were reduced to a set of 21 for data collection to avoid major differences between projects. The variables collected were:

17 Primary Components including:
data elements, logical groups of data elements,
data in Data Measurements Process,
data in Transaction Measurement Process,
data elements in internal files,
data elements in external files,
data on inputs,
data on outputs,
data on inquiries.

6 Variables with weights including:
AFP & UFP,
subtotals of UFP for each of 5 types- internal files, external files, inputs, outputs and enquiries.

They did a great deal of statistical analysis looking at the regression equations for a single dependent variable, Work-Effort, in terms of combinations of the 23 independent variables. They also considered the correlation between the 5 different FP types.

Summary

They found that the component parts of the FPA model in this homogeneous environment gave almost as good an answer in relation to the Work-Effort as the full model. By choosing suitable combinations of the 23 factors the model was almost equivalent to full FPA. However any generalisation would be dangerous in view of the nature of the data used for testing. Looking at the correlation of the 5 different FP components they claimed that there was no pairwise correlation so that all 5 could have been used to build their model.

Basili and Hutchens

This study looked at a family of syntactic complexity measures and derives two further metrics[Basili and Hutchens 1983]. They defined *slope,* a measure skill of a programmer in handling complexity, and *r square,* an indirect measure of the consistency of a programmer. The five metrics collected were:

STMT- counting the executable statements,
SynC -syntactic complexity involving nesting, length, structure and organisation counts,
CALL- the number of calls to procedures and functions,
v(g)- measures cyclomatic complexity by adding decisions to segments,
DecS- a count related to If, WHILE and CASE statements.

The metrics were collected from 19 compilers written by student teams. Comparisons were made between the metrics and their efficacy in predicting the number of changes made.

Summary

The measurements of *slope* and *r square* gave contradictory indications about the team approach to programming although they do suggest a specific disciplined team as opposed to a specific ad hoc team will behave in a more predictable and capable way when dealing with complexity.

They found that the statement count, STMT, correlated best with the program changes and that metrics which count specific parts of the code, like CALL and DecS, seem to be poorer predictors of program changes than those which count a feature spread all through the code.

Basili , Selby and Phillips

In this work the authors validated a collection of metrics looking particularly at Halstead's Software Science and McCabe's Cyclomatic Complexity [Basili , Selby and Phillips 1983]. They were looking at the effectiveness of the metrics in predicting effort , as measured in man hours, spent by programmers and managers counting the time from functional design through to acceptance testing. They also looked at the metrics as predictors of the quality of the software where this was to be measured in errors reported during development.

They broaden the number of metrics considered by looking at some of the estimators of the basic parameters to the Software Science metrics and also some of the more common metrics from the code. They used a code analysing program (SAP)to carry out the counting for the metrics and the data was drawn from ground support software for satellites which consisted of projects containing up to 112000 lines of Fortran code with 200- 600 modules (subroutines) in each project. Effort data was obtained from Component Status Reports filled out weekly by programmers and Resource Summary Forms filled out weekly by managers.

Internal and External Validation of the metrics

They carried out a number of tests and comparisons mostly on a total of about 1800 modules and considered:

- the correlation between the different estimators used for program length and level,
- the correlation between Software Science metrics and others,
- the relationship between the metrics and effort.
- the relationship between the metrics and errors

The table shows a list of metrics collected or calculated

Traditional Metrics	
calls	number of calls in a module to a function or subroutine
calls and jumps	total calls and decisions
source lines	source code include comments excluding blanks
source lines - comments	difference between source lines and comment lines
executable statements	Fortran Executable statements
cyclomatic complexity	1+sum of the constructs (do-loop etc. + AND ,OR)
cyclomatic complexity 2	1+ sum of the constructs (without AND ,OR)
revisions	number of versions of a module generated in the program library
changes	changes affecting the module (9 types)
weighted changes	effort spent making changes as reported by programmers on a 4pt scale
Software Science metrics	
$\eta = \eta_1 + \eta_2$	vocabulary metric measures of numbers of unique operators and operands
$\eta^* = \eta_1^* + \eta^*_2$	potential vocabulary metric, minimum numbers of operators and operands
error	(fault) a 'mistake' in code caused by a misconception or document discrepancy originating with the programmer. Counted by number of system changes citing error correction.
weighted error	effort spent fixing errors on a 4pt scale
program length $N= N_1+N_2$	total number of operators and operands
N^	estimator $\eta_1 \log_2\eta_1 + \eta_2 \log_2\eta_2$
program volume V	$N\log_2 \eta$
potential volume V*	$(2+\eta^*_2) \log_2 (2+\eta^*_2)$
program level L	V*/V
estimated program level L^	$2\eta_2 / \eta_1 N_2$
program difficulty D	1/L
program difficulty D_2	$1/L\text{^} = \eta_1 N_2 / 2\eta_2$
Effort E	$V/L = V^2 / V^*$
Approximated effort E^	$V/L\text{^} = \eta_1 N_2 N\log_2\eta / 2\eta_2$
Refined approximated effort E^^	$N\text{^} \log_2\eta / L\text{^}$
Program bugs B	V/E_0 E_0 the mean of elementary discriminations between potential errors
Program bugs B^	$E^{2/3} /3000$

The metrics considered by Basili, Selby and Phillips

They considered several sets of projects written by different sets of programmers for comparisons. They also broke down the validating metrics on effort into different levels

Summary

They found that considering the internal validation:

- the relationship of the Software Science metrics with their estimators seemed to be size dependent. Their accuracy seemed to vary with the size of the modules,
- N and V correlated well to the traditional program measures,
- program size metrics N_1 N_2 N V and correlated well with lines of code,
- none of the Software Science metrics correlated well with the number of revisions or the sum of procedure and function calls.

When looking at the effort and error data they found the correlation was greatly affected by the reliability of the report data . In general

- E^ and E^^ the estimators correlated better to actual effort than E but it was still not a very good correlation.
- There were disappointing results for correlation to errors except when restricting the projects to a particular programmer.

Errors are correlated best to the number of revisions.

Appendix C Specification of a Stack in VDM, ADT and Z

Operations

CREATE	creates an empty stack.
PUSH	puts an item on the stack.
TOP	reads the value of the item at the top of the stack.
POP	removes an element from the stack.
ISMT	checks to see if the stack is empty.

Signatures

CREATE:		$\rightarrow$ Stack
PUSH :	Item x Stack	$\rightarrow$ Stack
TOP :	Stack	$\rightarrow$ Item
POP :	Stack	$\rightarrow$ Stack
ISMT :	Stack	$\rightarrow$ Boolean

VDM

Stack :: s : Item*

CREATE

ext wr s:Item*

post s = []

PUSH(e:Item)

ext wr s:Item*

post s = [e] _ s'

TOP()e:Item

ext rd s:Item*

pre s ≠ []

post e = **hd** s

POP()

ext wr s:Item*

pre s ≠ []

post s = **tl** s'

ISMT()b:Boolean

ext rd s:Item*

post b = ↔ s = []

ADT

Axioms

e:Item s : Item*

TOP(**CREATE**) ::= error

TOP(**PUSH** (e,s)) ::= e

POP(**CREATE**) ::= **CREATE**

POP(**PUSH** (e,s)) ::= s

ISMT(**CREATE**) ::= T

ISMT(**PUSH** (e,s)) ::= F

Note the underscore denoting concatenation is a variant from the standard VDM.

Z

Stack

s:Item*

Create

Stack

s = <>

Push

Δ Stack

e? : Item

s' = <e?> _ s

Pop

Δ Stack

s' = **tail** s

____Top___________

Ξ Stack

e!:Item

e! = **head** s

____Ismt__________

Ξ Stack

b! : Boolean

b! ⇔ s = <>

Appendix D Results of the counting metrics on 9 initial specifications

ATTRIBUTE	VDM			VDM	Z	ADT	VDM	Z	ADT
Attribute/metric	P1	P2	P3	S1	S2	S3	A1	A2	A3
Readable									
brackets / line	1.48	1.33	1.31	1.00	0.43	2.25	0.95	0.36	3.33
Modifiable(ent)	7	7	7	7	3	2	7	4	2
Modifiable(op)	3	4	5	3	4	4	3	4	6
Standard	1	1	1	1	2-5	>5	1	2-5	>5
Modular	yes	yes	yes	yes	poss	no	yes	Poss	no
Size									
Vocabulary(not)	6	4	9	6	6	3	5	6	3
Vocabulary(mat)	4	4	5	0	13	0	5	5	0
Read time- mins	25	25	25	25	25	25	25	25	25
Write time	100	100	100	100	100	100	100	100	100
Hidden detail	3	0	6	2	2	0	1	1	1
Symbols	14	12	19	6	32	3	15	16	3
Base types	1	1	1	1	1	1	1	1	1
Length	23	24	29	18	14	8	19	22	12

Appendix E Results of the counting metrics on 3 larger specifications

ATTRIBUTE	VDM	Z	ADT
	SP1	SP2	SP3
total of symbols used	237	82	102
brackets/lines	2.22	1.22	4.02
symbols/lines	1.55	0.66	0.98
Language comments/line	30/153	30/123	15/104
Modifiable(ent)	7	2	2
Modifiable(op)	15	10	6 or 35
Standard	1	2-5	more
Modular	yes	no	no
Abstract	some	some	more
Vocabulary(not)	8	14	8
Vocabulary(mat)	4	5	6
Read time	70 min	70 min	70 min
Write time	560 min	560 min	560 min
Built in notation	8	4	0
specifier defined notation	8	12	4
overall symbols count	14 x 26	14 x 18	14 x 3
Repetition	15	0	3
Base types	2	2	2
Length	153	123	104

Appendix F The cleaned data set

The cleaned data set consisted of:

NLVL,	course level (C1 to C6) .
student id number,	from 1-147
SPEC;	one of the five specifications 1-5
CMT(Comment),	these three
MNFUL(Meaningful_names),	were dummy variables
STRCT(Structured)	indicating the type of specification given
TREAD	time to read to intro (in seconds),
TOTQ1,TOTQ2,TOTQ3,	times to answer Questions 1,2, and 3 respectively,
TOTQT	total time to answer the questions;
	rankings of the 5 specs from least comprehensible to most; coded as -2 -1 0 1 2
S1,S2,S3	Scores on the three questions; either 0 or 1
TOTS	Total score out of 3.

Summary of variables

Variable	Mean	Std Dev	Minimum	Maximum	N
Scores for questions 1,2,3, and total-score:					
S1	.46	.50	.00	1.00	146
S2	.55	.50	.00	1.00	146
S3	.32	.47	.00	1.00	146
TOTS	1.33	1.23	0	3	146
Indicator variables for subjects with zero and three total-score					
IZERO	.37	.48	0	1	146
ITHREE	.27	.44	0	1	146
Times to read the introduction, and the three questions					
TREAD	168.24	129.29	30.00	780.00	95
TQ1	149.23	110.65	27.00	660.00	111
TQ2	105.79	73.21	25.00	480.00	139
TQ3	98.47	66.35	17.00	480.00	127
TOTQT	341.52	174.30	120.00	960.00	98

Appendix G Details of the Factor Analysis of the timing data.

The matrix of factor loadings are:

	Factor 1
TQ1	**.67773**
TQ2	.45367
TQ3	**.80060**
TREAD	**.36831**

Note that all the loadings are positive, with the times for questions 1 and 3 having the highest loadings on the Common factor. The loadings would have to be divided by the response standard deviations to obtain the relevant correlations.

Final Statistics:

Variable	Communality	*	Factor	Eigenvalue	Pct of Var	Cum Pct
TQ1	.45932	*	1	1.44174	36.0	36.0
TQ2	.20581	*				
TQ3	.64096	*				
TREAD	.13565	*				

Appendix H Analysis of Variance of total scores

Anova of TOTS by NLVL and SPEC, with TOTQT.

HIERARCHICAL sums of squares; Covariates entered AFTER main effects

Source of Variation	**Sum of Squares**	**DF**	**Mean Square**	**F**	**Sig of F**
Main Effects	**25.016**	**9**	**2.780**	**2.235**	**.030**
NLVL	**10.355**	**5**	**2.071**	**1.665**	**.155**
SPEC	**14.661**	**4**	**3.665**	**2.947**	**.026**
Covariates	**17.283**	**1**	**17.283**	**13.895**	**.000**
TOTQT	**17.283**	**1**	**17.283**	**13.895**	**.000**
2-Way Interactions	**18.020**	**18**	**1.001**	**.805**	**.688**
NLVL SPEC	**18.020**	**18**	**1.001**	**.805**	**.688**
Explained	**60.320**	**28**	**2.154**	**1.732**	**.035**
Residual	**83.337**	**67**	**1.244**		
Total	**143.656**	**95**	**1.512**		

147 cases were processed.

51 cases (34.7 pct) were treated as missing through incomplete data.

Appendix I The Analysis of Variance for the experimental design

```
* * * * * * A n a l y s i s   o f   V a r i a n c e * * * * *


Tests of Significance for TOTS using Cov Adj SEQUENTIAL Sums of Squares
Source of Variation              SS      DF      MS       F          Sig of F


WITHIN+RESIDUAL                 119.70   91     1.32
REGRESSION                      12.90     1    12.90     9.81          .002
CMT                             2.53      1     2.53     1.92          .169
MNFUL                           5.80      1     5.80     4.41          .039
STRCT                           .24       1     .24      .19           .667
CMT * MNFUL                     .76       1     .76      .58           .449


(Model)                         25.74     5     5.15     3.91          .003
(Total)                         145.44    96     1.52


R-Squared =          .177
Adjusted R-Squared =  .132


- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Regression analysis for WITHIN+RESIDUAL error term
--- Individual Univariate .9500 confidence intervals
Dependent variable .. TOTS


COVARIATE          B        Beta   Std. Err.    t-Value   Sig. of t


TOTQT        -.00214    -.30393      .001      -3.132      .002


COVARIATE   Lower -95%   CL- Upper


TOTQT          -.003      -.001
```

Appendix J Logistic Analysis of question success

```
LOGISTIC ANALYSIS OF THE PROBABILITY OF
GETTING 3/3

    Number of selected cases:              147
    Number rejected because of missing data:  50
    Number of cases included in the analysis: 97

Dependent Variable..  ITHREE

Initial -2 Log Likelihood   110.71022

Fitted model
 -2 Log Likelihood      79.978
 Goodness of Fit       78.204

                       Chi-Square  df      Significance

 Model Chi-Square        30.733     9       .0003

Classification Table for ITHREE
                   Predicted
                    0     1              Percent Correct
                    0  I  1
Observed        +-------+-------+
  0   0         I  67  I  5  I            93.06%
               +-------+-------+
  1   1         I  13  I  12  I           48.00%
               +-------+-------+
                   Overall                81.44%
--------------------- Variables in the Equation ----------------------

Variable        B      S.E.    Wald   df    Sig

C1        -3.3108  1.4739  5.0462   1  .0247
C2        -2.2444  1.3642  2.7070   1  .0999
C6        -3.7509  1.2891  8.4669   1  .0036
C4        -1.5814  1.2642  1.5649   1  .2109
C5         -.4114  1.3227   .0967   1  .7558
CMT         1.7208    .7630  5.0872    1  .0241
MNFUL        .6734    .7109   .8973   1  .3435
STRCT       -.6645    .7724   .7402   1  .3896
TOTQT       -.0111    .0036  9.5154  1  .0020
Constant    3.7577   2.0728  3.2864  1  .0699
```

Appendix K Perceived Comprehensibility Rating data

Frequency tables for the comprehensibility of specifications, vertically ordered by spec 1,2,3,4,5; and horizontally by comprehensibility coded as:

least comprehensible......				....most
-2	-1	0	1	2
1	2	3	4	5

The first three tables are for each of the total scores 3,2,1. The final table is for all subjects. Tots=3 (N=39)

1	1	1	7	29
2	17	14	3	2
30	5	1	2	1
4	2	18	14	2
2	14	5	13	5

Tots=2 (N=24)

1	0	3	4	3
1	9	9	5	9
19	2	0	1	0
1	5	8	5	8
2	8	4	9	4

Tots=1 (N=28)

0	1	3	6	3
6	8	5	6	5
12	10	4	1	4
7	3	11	2	11
3	6	5	13	5

Tots=0 (N=47)

2	5	8	8	8
3	15	10	15	10
25	9	5	3	5
9	5	16	10	16
8	13	8	11	8

All Subjects (N=138)

4	7	15	25	87
12	49	38	29	8
86	26	10	7	8
21	15	53	31	21
15	41	22	46	14

The staff telephone directory contains names, phone numbers and office labels.

1 [*NAME, NUMBER, OFFICE*]

Each member of staff is identified by *NAME* has an *OFFICE* and a phone extension *NUMBER*. Each member of staff can have only one phone number and one office. However several people can share an office and share phones. A member of staff is only recorded in the phone directory if they have a phone number. Although a phone number can be reassigned to a different physical telephone that phone can only be in one office at any given time. Telephones may be in offices even though people are not assigned to the phone or the office.

Operations

The operations on the phone directory are:

2 *OPERATION* ::= InitPhoneDir | AddPerson | MovePerson | DeletePerson | Move Phone
3 DisconnectPhone | ConnectPhone | QueryPersonNumber | QueryPersonOffice |
4 ListNumberPeople | ListOfficePeople | ListOfficeNumbers

- *InitPhoneDir* - **Initially the directory is emptied.**
- *AddPerson* - **Assign a person a phone and an office.**
- *DeletePerson* - **Remove a person from the directory.**
- *MovePerson* - **Reassign a person to a different office and phone.**
- *MovePhone* - **Move the location of a phone.**
- *ConnectPhone* - **Associate a phone number with an office.**
- *DisconnectPhone* - **Remove the phone number from the directory.**
- *QueryPersonNumber* - **Check which phone extension a person has.**
- *QueryPersonOffice* - **Check which office a person is in.**
- *ListNumberPeople* - **List the people on a given phone number.**
- *ListOfficePeople* - **List the people who share a given office.**
- *ListOfficeNumbers* - **List the phones in a given office.**

Responses

These are responses given as a result of the operations. If all is well the normal operation response is a return message of 'ok'.

5 *RESPONSE* ::= ok | adderror | notfound | connecterror

Basic State

The telephone directory consists of three functions relating the sets *NAME, NUMBER,* and *OFFICE*.

```
┌─ PhoneDir ──────────────────────────
│ $phone : NAME \rightarrow NUMBER$
│ $addr : NAME \rightarrow OFFICE$
│ $location : NUMBER \twoheadrightarrow OFFICE$
├───────────────
│ $\mathrm{dom}\ addr = \mathrm{dom}\ phone$
│ $\mathrm{ran}\ phone \subseteq \mathrm{dom}\ location$
└─────────────────────────────────
```

This schema includes setting up an empty phone directory, plus all operations and the errors arising from the failure of the operations: AddPerson, DeletePerson, QueryPersonNumber, QueryPersonOffice, ListNumberPeople, ListOfficePeople, ListOfficeNumbers, MovePerson, MovePhone, ConnectPhone and DisconnectPhone.

Note as both Moving operations use the override function we need not produce error conditions it will be the trivial override.

```
┌─ PhoneOperations ──────────────────
│ $\Delta PhoneDir$
│ $name? : NAME$
│ $office?, office! : OFFICE$
│ $number?, number! : NUMBER$
│ $operation? : OPERATION$
│ $resp! : RESPONSE$
│ $people! : \mathbb{P}\ NAME$
│ $phones! : \mathbb{P}\ NUMBER$
├───────────────
│ (   $resp! = ok$
│ (   (   $operation? = InitPhoneDir$
│           $addr' = phone' = location' = \varnothing$
│       ) $\vee$
│       (   $operation? = AddPerson$
│           $name? \notin \mathrm{dom}\ phone$
│           $phone' = phone \cup \{name? \mapsto number?\}$
│           $addr' = addr \cup \{ name? \mapsto office? \}$
│           (   (   $number? \notin \mathrm{dom}\ location$
│                   $location' = location \cup \{ number? \mapsto office? \}$
│               ) $\vee$
│               (   $number? \in \mathrm{dom}\ location$
│                   $location\ number? = office?$
│                   $location' = location$
│       )   )   ) $\vee$
│       (   $name? \in \mathrm{dom}\ phone$
│           (   $operation? = MovePerson$
│               $phone' = phone \oplus \{name? \mapsto number?\}$
│               $addr' = addr \oplus \{ name? \mapsto office? \}$
```

```
            location′ = location ⊕ { number? ↦ office? }
        ) ∨
        (   (   operation? = DeletePerson
                phone′ = phone \ { name? ↦ phone name? }
                addr′ = addr \ { name? ↦ addr name? }
            ) ∨
            (   (   operation? = QueryPersonNumber
                    number! = phone name?
                ) ∨
                (   operation? = QueryPersonOffice
                    office! = addr name?
                )
                phone′ = phone
                addr′ = addr
            )
            location′ = location
        )
    ) ∨
    (   operation? = ListNumberPeople
        number? ∈ ran phone
        people! = { p : NAME | phone p = number? }
    ) ∨
    (   operation? = ListOfficePeople
        office? ∈ ran addr
        people! = { p : NAME | addr p = office? }
    ) ∨
    (   operation? = ListOfficeNumbers
        office? ∈ ran location
        phones! = { p : NUMBER | location = office? }
    ) ∨
    (   phone′ = phone
        addr′ = addr
        (   operation? = ConnectPhone
            (   (   number? ∉ dom location
                    location′ = location ∪ { number? ↦ office? }
                ) ∨
                (   number? ∈ dom location
                    location number? = office?
                    location′ = location
        )   )   ) ∨
        (   operation? = DisconnectPhone
            number? ∈ dom location
            location′ = location \ { number? ↦ location number? }
        ) ∨
        (   operation? = MovePhone
            location′ = location ⊕ { number? ↦ office? }
        ) ∨
```

```
            (   (   operation? = QueryPersonNumber ∨
                    operation? = QueryPersonOffice ∨
                    operation? =  ListNumberPeople ∨
                    operation? = ListOfficePeople ∨
                    operation? = ListOfficeNumbers
                )
                location' = location
        )   )
    ) ) ∨
    (   phone' = phone
        addr' = addr
        location' = location
        (   operation? = AddPerson ∨  operation? = ConnectPhone
            location number? ≠ office?
            resp! = connecterror
        ) ∨
        (   operation? = AddPerson
            name? ∈ dom phone
            resp! = adderror
        ) ∨
        (   (   (   operation? = QueryPersonNumber ∨
                    operation? = QueryPersonOffice ∨
                    operation? = DeletePerson ∨
                    operation? = MovePerson
                )
                name? ∉ dom phone
            ) ∨
            (   operation? = ListNumberPeople
                number? ∉ ran phone
            ) ∨
            (   operation? = ListOfficePeople
                office? ∉ ran addr
            ) ∨
            (   operation? = ListOfficeNumbers
                office? ∉ ran location
            )
            resp! = notfound
    )   )
```

Appendix M The six main parts of Specification B

The staff telephone directory contains names, phone numbers and office labels

1 $[NAME, NUMBER, OFFICE]$

Each member of staff is identified by *NAME* has an *OFFICE* and a phone extension *NUMBER*. Each member of staff can have only one phone number and one office. However several people can share an office and share phones. A member of staff is only recorded in the phone directory if they have a phone number. Although a phone number can be reassigned to a different physical telephone that phone can only be in one office at any given time. Telephones may be in offices even though people are not assigned to the phone or the office.

Operations

The operations on the phone directory are:

2 $QUERY$ ::= QueryPersonNumber | QueryPersonOffice | ListNumberPeople |
3 ListOfficePeople | ListOfficeNumbers
4 $UPDATE$ ::= AddPerson | DeletePerson | MovePerson | DisconnectPhone |
5 InitPhoneDir | MovePhone | ConnectPhone

Responses

These are responses given as a result of the operations. If all is well the normal operation response is a return message of 'ok'.

6 $RESPONSE$ ::= ok | adderror | notfound | connecterror

Basic State

The telephone directory consists of three functions relating the sets *NAME, NUMBER,* and *OFFICE*

7 —— *PhoneDir* ——————
8 $phone : NAME \rightarrow NUMBER$
9 $addr : NAME \rightarrow OFFICE$
10 $location : NUMBER \twoheadrightarrow OFFICE$
————
11 $\operatorname{dom} addr = \operatorname{dom} phone$
12 $\operatorname{ran} phone \subseteq \operatorname{dom} location$
——————

This schema sets up an empty directory :

13 —— *InitPhoneDir* ——————
14 $PhoneDir'$
————
15 $addr' = phone' = location' = \varnothing$
——————

The operations described here on the phone directory are:

- *AddPerson* **- Assign a person a phone and an office.**
- *DeletePerson* **- Remove a person from the directory.**

```
┌── BasicOperations ──────────────
│ ΔPhoneDir
│ name? : NAME
│ office? : OFFICE
│ number? : NUMBER
│ operation? : UPDATE
│ resp! : RESPONSE
├──────────────────
│ (   operation? = AddPerson
│     name? ∉ dom phone
│     phone' = phone ∪ { name? ↦ number? }
│     addr' = addr ∪ { name? ↦ office? }
│     (   (   number? ∉ dom location
│               location′ = location ∪ { number? ↦ office? }
│         ) ∨
│         (   number? ∈ dom location
│               location number? = office?
│               location′ = location
│ )   )   ) ∨
│ (   operation? = DeletePerson
│     name? ∈ dom phone
│     phone' = phone \ { name? ↦ phone name? }
│     addr' = addr \ { name? ↦ addr name? }
│ )
│ resp! = ok
└──────────────────────────────
```

This schema shows the errors arising from the failure of the operations *AddPerson* **and** *DeletePerson.*

```
┌── BasicFailures ──────────────
│ Ξ PhoneDir
│ name? : NAME
│ number? : NUMBER
│ operation? : UPDATE
│ resp! : RESPONSE
├──────────────────
│ (   operation? =DeletePerson
│     name? ∉ dom phone
│     resp! = connecterror
│ ) ∨
│ (   operation? = AddPerson
│     name? ∈ dom phone
│     resp! = adderror
│ )
└──────────────────────────────
```

The operations described here on the phone directory are::

- *QueryPersonNumber* **- Check which phone extension a person has.**
- *QueryPersonOffice* **- Check which office a person is in.**
- *ListNumberPeople* **- List the people on a given phone number.**
- *ListOfficePeople* **- List the people who share a given office.**
- *ListOfficeNumbers* **- List the phones in a given office.**

```
┌── PhoneQueries ──────────────
│ ΞPhoneDir
│ name? : NAME
│ office?, office! : OFFICE
│ number?, number! : NUMBER
│ operation? : QUERY
│ resp! : RESPONSE
│ people! : ℙ NAME
│ phones! : ℙ NUMBER
├──────────────
│ (  (   operation? = QueryPersonNumber
│        name? ∈ dom phone
│        number! = phone name?
│    ) ∨
│    (   operation? = QueryPersonOffice
│        name? ∈ dom addr
│        office! = addr name?
│    )
│    ) ∨
│    (   operation? = ListNumberPeople
│        number? ∈ ran phone
│        people! = { p : NAME | phone p = number? }
│    ) ∨
│    (   operation? = ListOfficePeople
│        office? ∈ ran addr
│        people! = { p : NAME | addr p = office? }
│    ) ∨
│    (   operation? = ListOfficeNumbers
│        office? ∈ ran location
│        phones! = { p : NUMBER | location p = office? }
│ )  ) ∧
│ resp! = ok
└──────────────
```

This schema shows the errors arising from the failure of the operations *QueryPersonNumber*, *QueryPersonOffice*, *ListNumberPeople*, *ListOfficePeople* **and** *ListOfficeNumbers.*

```
86     ┌─ FailQueries ──────────────
87     │ ΞPhoneDir
88     │ name? : NAME
89     │ office? : OFFICE
90     │ number? : NUMBER
91     │ operation? : QUERY
92     │ resp! : RESPONSE
       ├──────────────
93     │ (  (  operation? = QueryPersonNumber
94     │       name? ∉ dom phone
95     │    ) ∨
96     │    (  operation? = QueryPersonOffice
97     │       name? ∉ dom addr
98     │    ) ∨
99     │    (  operation? = ListNumberPeople
100    │       number? ∉ ran phone
101    │    ) ∨
102    │    (  operation? = ListOfficePeople
103    │       office? ∉ ran addr
104    │    ) ∨
105    │    (  operation? = ListOfficeNumbers
106    │       office? ∉ ran location
107    │ )  ) ∧
108    │ resp! = notfound
       └──────────────────────────
```

The operations described here on the phone directory are::

- *MovePerson* - **Reassign a person to a different office and phone.**
- *MovePhone* - **Move the location of a phone.**
- *ConnectPhone* - **Associate a phone number with an office.**
- *DisconnectPhone* - **Remove the phone number from the directory.**

```
109    ┌─ ModifyOperations ──────────────
110    │ ΔPhoneDir
111    │ name? : NAME
112    │ office? : OFFICE
113    │ number? : NUMBER
114    │ operation? : UPDATE
115    │ resp! : RESPONSE
       ├──────────────
116    │ (  (  operation? = MovePerson
117    │       phone' = phone ⊕ {name? ↦ number?}
118    │       addr' = addr ⊕ { name? ↦ office? }
119    │       location' = location ⊕ { number? ↦ office? }
120    │    ) ∨
|       (   operation? = ConnectPhone
|           phone' = phone
|           addr' = addr
|           (   number? ∉ dom location
|               location′ = location ∪ { number? ↦ office? }
|           ) ∨
|           (   number? ∈ dom location
|               location number? = office?
|               location′ = location
|           )
|       ) ∨
|       (   number? ∈ dom location
|           (   operation? = DisconnectPhone
|               location′ = location \ { number? ↦ location number? }
|           ) ∨
|           (   operation? = MovePhone
|               location′ = location ⊕ { number? ↦ office? }
|           )
|           phone' = phone
|           addr' = addr
|   )   ) ∧
|   resp! = ok
|______________________________
```

This schema shows the errors arising from the failure of the operations *MovePerson*, *MovePhone*, *ConnectPhone* **and** *DisconnectPhone.*

```
┌── FailModify ──────────────
|   ΞPhoneDir
|   office?  : OFFICE
|   number? : NUMBER
|   operation? : UPDATE
|   resp! : RESPONSE
|──────────────────
|   operation? = DisconnectPhone
|   resp! = connecterror
|   (   office? ≠ location number?
|       ∨
|       number? ∉ dom location
|   )
|______________________________
```

Note as both Moving operations use the override function we need not produce error conditions it will be the trivial override.

$PhoneOperations \triangleq InitPhoneDir \wedge (BasicOperations \vee BasicFailures) \wedge (PhoneQueries \vee FailQueries) \wedge (ModifyOperations \vee FailModify)$

The staff telephone directory contains names, phone numbers and office labels.

1 $[NAME, NUMBER, OFFICE]$

Each member of staff is identified by *NAME* has an *OFFICE* and a phone extension *NUMBER*. Each member of staff can have only one phone number and one office. However several people can share an office and share phones. A member of staff is only recorded in the phone directory if they have a phone number. Although a phone number can be reassigned to a different physical telephone that phone can only be in one office at any given time. Telephones may be in offices even though people are not assigned to the phone or the office.

Responses

These are responses given as a result of the operations. If all is well the normal operation response is a return message of 'ok'.

2 $RESPONSE$ ::= ok | adderror | notfound | connecterror

Basic State

The telephone directory consists of three functions relating the sets *NAME, NUMBER,* and *OFFICE*

3 *PhoneDir*

4 $phone : NAME \rightarrow NUMBER$

5 $addr : NAME \rightarrow OFFICE$

6 $location : NUMBER \twoheadrightarrow OFFICE$

7 $\mathrm{dom}\ addr = \mathrm{dom}\ phone$

8 $\mathrm{ran}\ phone \subseteq \mathrm{dom}\ location$

This schema sets up an empty directory :

9 *InitPhoneDir*

10 $PhoneDir'$

11 $addr' = phone' = location' = \emptyset$

The operation described here on the phone directory is: *AddPerson*
Assign a person a phone and an office.

```
12 ┌─ AddPerson ──────────────
13 │ ΔPhoneDir
14 │ name? : NAME
15 │ office? : OFFICE
16 │ number? : NUMBER
17 │ resp! : RESPONSE
   ├──────────────────────
18 │ name? ∉ dom phone
19 │ (   (   number? ∉ dom location
20 │         location´ = location ∪
21 │             { number? ↦ office? }
22 │     ) ∨
23 │     (   number? ∈ dom location
24 │         location number? = office?
25 │         location´ = location
26 │ )   ) ∧
27 │ phone' = phone ∪ {name? ↦ number?}
28 │ addr' = addr ∪ { name? ↦ office? }
29 │ resp! = ok
   └──────────────────────────────
```

This schema shows the errors arising from The failure of the operation *AddPerson*

```
30 ┌─ AddFailures ────────────
31 │ Ξ PhoneDir
32 │ name? : NAME
33 │ resp! : RESPONSE
   ├──────────────────────
34 │ name? ∈ dom phone
35 │ resp! = adderror
   └──────────────────────────────
```

The operation described here on the phone directory is: *DeletePerson*
Remove a person from the directory.

```
36 ┌─ DeletePerson ────────────
37 │ ΔPhoneDir
38 │ name? : NAME
39 │ resp! : RESPONSE
   ├──────────────────────
40 │ name? ∈ dom phone
41 │ phone' =phone \{ name? ↦ phonename?}
42 │ addr' = addr \ { name? ↦ addr name? }
44 │ resp! = ok
   └──────────────────────────────
```

This schema shows the errors arising from The failure of the operation *DeletePerson*

```
45 ┌─ DeleteFailures ──────────
46 │ Ξ PhoneDir
47 │ name? : NAME
48 │ resp! : RESPONSE
   ├──────────────────────
49 │ name? ∉ dom phone
50 │ resp! = connecterror
   └──────────────────────────────
```

The operations described here on the phone directory is: *QueryPersonNumber* **and** *QueryPersonOffice*

This schema shows the errors arising from the failure of the operations

QueryPersonNumber **- Check which phone extension a person has.**
QueryPersonOffice **- Check which office a person is in.**

```
┌── QueryPersonNumber ──────
│ ΞPhoneDir
│ name? : NAME
│ number! : NUMBER
│ resp! : RESPONSE
├──────────────────
│ name? ∈ dom phone
│ number! = phone name?
│ resp! = ok
└──────────────────────
```

```
┌──FailQueryPerson ──────
│ ΞPhoneDir
│ name? : NAME
│ resp! : RESPONSE
├──────────────────
│ name? ∈ dom phone
│ resp! = notfound
└──────────────────────
```

```
┌── QueryPersonOffice ──────
│ ΞPhoneDir
│ name? : NAME
│ resp! : RESPONSE
├──────────────────
│ name? ∈ dom addr
│ number! = addr name?
│ resp! = ok
└──────────────────────
```

The operation described here on the phone directory is: *ListNumberPeople*

This schema shows the errors arising from the failure of the operation

ListNumberPeople **- List the people on a given phone number .**

```
┌── ListNumberPeople ──────
│ ΞPhoneDir
│ number? : NUMBER
│ people! : ℙ NAME
│ resp! : RESPONSE
├──────────────────
│ number? ∈ ran phone
│ people! = { p : NAME | phone p = number? }
│ resp! = ok
└──────────────────────
```

```
┌── FailListNumberP ──────
│ ΞPhoneDir
│ number? : NUMBER
│ resp! : RESPONSE
├──────────────────
│ number? ∉ ran phone
│ resp! = notfound
└──────────────────────
```

The operations described here on the phone directory is: *ListOfficePeople* **and** *ListOfficeNumbers*

This schema shows the errors arising from the failure of the operations

ListOfficePeople **- List the people who share a given office.**
ListOfficeNumbers **- List the phones in a given office.**

```
86  ┌── ListOfficePeople ─────────
87  │  ΞPhoneDir
88  │  office? : OFFICE
89  │  people! : ℙ NAME
90  │  resp! : RESPONSE
    ├──────────────────
91  │  office? ∈ ran addr
92  │  people! = { p : NAME | addr p =
office? }
93  │  resp! = ok
    └──────────────────────────
```

```
94  ┌── FailListOffice ──────────
95  │  ΞPhoneDir
96  │  office? : OFFICE
97  │  resp! : RESPONSE
    ├──────────────────
98  │  office? ∉ ran addr
99  │  resp! = notfound
    └──────────────────────────
```

```
100┌── ListOfficeNumbers ─────────
101│  ΞPhoneDir
102│  office? : OFFICE
103│  phones! : ℙ NUMBER
104│  resp! : RESPONSE
   ├──────────────────
105│  office? ∈ ran location
106│  phones! = { p : NUMBER | location p = office? }
107│  resp! = ok
   └──────────────────────────
```

The operations described here on the phone directory are:

MovePerson **- Reassign a person to a different office and phone.**
MovePhone **- Move the location of a phone**

Note as both Moving operations use the override function we need not produce error conditions it will be the trivial override

```
108┌─ MovePerson ──────────────
109│  ΔPhoneDir
110│  name? : NAME
111│  office? : OFFICE
112│  number? : NUMBER
113│  resp! : RESPONSE
   ├──────────────────
114│  phone' = phone ⊕ {name? ↦ number?}
115│  addr' = addr ⊕ { name? ↦ office? }
116│  location′ = location ⊕ { number? ↦ office? }
117│  resp! = ok
   └──────────────────────────────
```

```
118┌─ MovePhone ──────────────
119│  ΔPhoneDir
120│  office? : OFFICE
121│  number? : NUMBER
122│  resp! : RESPONSE
   ├──────────────────
123│  location′ = location ⊕ { number? ↦ office? }
124│  phone' = phone
125│  addr' = addr
126│  resp! = ok
   └──────────────────────────────
```

The operation described here on the phone directory is: *ConnectPhone*

ConnectPhone - **Associate a phone number with an office.**

```
┌── ConnectPhone ──────────────
│ ΔPhoneDir
│ office? : OFFICE
│ number? : NUMBER
│ resp! : RESPONSE
├──────────────────
│ number? ∉ dom location
│ location′ = location ∪ { number? ↦ office? }
│ resp! = ok
└──────────────────────────
```

This schema shows the errors arising from the failure of the operation

```
┌── FailConnectPhone ──────────────
│ ΞPhoneDir
│ number? : NUMBER
│ resp! : RESPONSE
├──────────────────
│ number? ∈ dom location
│ resp! = errorconnect
└──────────────────────────
```

The operation described here on the phone directory is: *DisconnectPhone*

DisconnectPhone - **Remove the phone number from the directory.**

```
┌── DisconnectPhone ──────────────
│ ΔPhoneDir
│ number? : NUMBER
│ resp! : RESPONSE
├──────────────────
│ number? ∈ dom location
│ location′ = location \
│       { number? ↦ location number? }
│ phone' = phone
│ addr' = addr
│ resp! = ok
└──────────────────────────
```

This schema shows the errors arising from the failure of the operation

```
┌── FailDisconnectPhone ──────────────
│ ΞPhoneDir
│ number? : NUMBER
│ resp! : RESPONSE
├──────────────────
│ (   office? ≠  location number? ∨
│     number? ∉ dom location
│ )
│ resp! = errorconnect
└──────────────────────────
```

Operations

The operations on the phone directory are therefore:

PhoneOperations ≙ *InitPhoneDir* ∧ (*AddPerson* ∨ *AddFailures*) ∧ (*DeletePerson* ∨ *DeleteFailures*) ∧ (*QueryPersonNumber* ∨ *FailQueryPerson*) ∧ (*QueryPersonOffice* ∨ *FailQueryPerson*) ∧ (*ListNumberPeople* ∨ *FailListNumberP*) ∧ (*ListOfficePeople* ∨ *FailListOffice*) ∧ (*ListOfficeNumbers* ∨ *FailListOffice*) ∧ *MovePerson* ∧ *MovePhone* ∧ (*ConnectPhone* ∨ *FailConnectPhone*) ∧ (*DisconnectPhone* ∨ *FailDisconnectPhone*)

Appendix O An example of the set of 20 questions to accompany Specification A

1. What information is contained in the line numbered '1' of the specification?
2. What is indicated by the difference between *number*? and *number*! in line 16?
3. What information is given in line 28 of the specification?
4. Describe the purpose of line 68.
5. Show how lines 27 and 28 would appear if a new person, 'Jones', is to be given office number '48B' and a phone number '376'.
6. List all the lines of the specification that would need to be changed if it was decided to rename the function *addr*?
7. Describe the condition(s) which give rise to the response '*adderror*'.
8. Why is the type of '*phones*!' in line 20 $\mathbb{P}$*NUMBER* and not just *NUMBER*?
9. Which 5 conditions in the telephone system give rise to the response '*notfound*'?
10. Explain the significance of the '⊕' symbol in line 89.
11. Which line informs you that a phone remains in the same office when a person is removed from the directory?
12. Which line in the specification tells you that a person can only have one phone?
13. Which line or lines inform you that the telephone directory is unchanged by the operation *QueryPersonNumber*?
14. Which variable(s) remain unchanged if a phone is disconnected?
15. Which variables are modified when a person is added to the phone directory?
16. Can an office appear in the directory if it does not have any phones in it? On which line(s) of the specification did you find the answer?
17. Can an office appear in the directory if it does not have any people in it? On which line(s) of the specification did you find the answer?
18. When adding a person to the directory what two things happen if the phone number is already associated with an office? (give line references)
19. Write the additional lines that would be required if the phone directory specification were to be modified so that it stores the department that a person works in. State where these new lines would be inserted into the specification.
20. Using your answer to question 19, write the additional lines that would be needed to add an operation called '*QueryPersonDept*' that, given the name of a person, checks which department that person is in. State where these new lines would be inserted

Appendix P Published papers

Correspondence

Mathematical Notation in Formal Specification: Too Difficult for the Masses?

Kate Finney

Abstract—The phrase "not much mathematics required" can imply a variety of skill levels. When this phrase is applied to computer scientists, software engineers, and clients in the area of formal specification, the word "much" can be widely misinterpreted with disastrous consequences. A small experiment in reading specifications revealed that students already trained in discrete mathematics and the specification notation performed very poorly; much worse than could reasonably be expected if formal methods proponents are to be believed.

Index Terms—Formal specification, mathematics, reading Z.

1 Introduction

THERE has been a recognition in recent years that education in the uses of formal methods is vital, and that the role of mathematics is central in this [1], [2]. It is encouraging to see the increasing emphasis on discrete mathematics as a basis for software engineering and the early insertion of general mathematics teaching in undergraduate courses for computer science. One of the difficulties caused by the sequential nature of mathematics is that the previous level of understanding is constantly overwritten as new skills are acquired, making it hard to recall a previous stage. Experienced practitioners of formal methods also pass along this learning curve assimilating concepts and notation so that reading specifications become as easy as if it were written in natural language. In promoting formal methods to new users the strangeness of the notation as well as the underlying mathematical background must be taken into account.

The early pioneers in the field of programming had excellent mathematical training and background and often held the view that programming was essentially an application of mathematics [3]. The development of formal methods with the rigorous proofs and stringent rule bases has by definition its origin in mathematical constructions and its first practitioners from centers such as IBM and Oxford University Programming Research Group were operating with a high skill base in this respect. It is often people from this background who propose that the level of mathematics required for the understanding of formal specification is not great. Hall in his Seven Myths [4] asserts "mathematics for specification is easy." A simple formal specification will contain as a minimum, concepts and notation from set theory and logic, and these are often quoted as all that are required.

The formalization of problems into abstract statements and the manipulation of symbolic expressions seem to be real stumbling blocks to the vast majority of the population. Even teachers of mathematics at A level (the standard course for 16–18 year olds interested in mathematics), are reporting problems with the algebraic competence of their pupils and as schools are moving away from the teaching of algebra and removing geometric proofs from the curriculum lower down the school this situation will not improve. [5].

Against this background it is not surprising that problems are experienced by users in the area of formal methods. Our conjecture was that people find formal specifications difficult to read because of the large use of symbols. When they attempt to write a specification the need for great attention to detail and correct use of mathematical statements add to the notation difficulties. These users may include those who write the specifications, those who interpret and try to implement them in code, and those who need to read the specification to check it has captured the client's requirements. However, the assumptions made are that with a short period of training, as little as two days according to Potter [6], they will quickly feel comfortable with the concepts and expressions and be reading and writing specifications.

On the basis of our own experimental work we show that the language of mathematics of the type required in formal methods is not widely known or easily used even among interested groups. We also contend that assumptions about general mathematical experience are becoming increasingly out of touch.

The experiment that was carried out involved 62 students, undergraduate and postgraduate, in reading a very small portion (less than 20 lines) of a specification in Z. All were attending computing courses and most had been through a basic grounding in discrete mathematics in addition to separate tuition in the use of Z. The hours of tuition varied between student groups with all undergraduate classes completing a semester of formal methods while the postgraduates had a shorter time concentrating on the reading and writing of Z specifications. Details of the background to the experiment can be found in [7]. Each student was asked three questions to test their ability to read and understand the specification.

The context of the experiment was to assess the effects on comprehensibility of adding meaningful variable names and annotated English to Z. The full results are described elsewhere [7]. What we concentrate on here is a different outcome of the experiment, namely that in general the students found it difficult to understand any of the very simple Z specifications. (An example of one such specification appears in Fig. 1.)

The results given here (Fig. 2) show the numbers of questions each student answered correctly. The point to note is that 19 students, nearly a third of the group, could not answer a single question and found the specification incomprehensible.

These results must be treated with caution, and we recognize that this is not a rigorous treatment, the experience, ability, and motivation of the subjects could all be factors affecting this outcome. This initial study, nevertheless, gives cause for concern and could provide a pointer to areas for further investigation. Assumptions are being made about the ability of your "average" software engineer, programmer, and even client to interpret specifications which may be false. In their recent paper Fenton et al. [8] strongly advocate empirical validation of methods used in software engineering. While these results only give an indication of some of the underlying difficulties of applying formal methods, it has been an attempt to put some statistics to an argument otherwise based on opinion and feelings.

2 Conclusion

In conclusion, if formal specifications are to become an acceptable and useful aspect of software engineering those who would be their champions should recognize:

- the level of mathematics even among interested parties may be some way below the "not much" category they have in mind.

• *K. Finney is in the School of Computing and Mathematical Sciences, University of Greenwich, Wellington Street, Woolwich, London SE18 6PF. E-mail. k.finney@greenwich.ac.uk.*

Manuscript received November 1995; revised December 1995.
For information on obtaining reprints of this article, please send e-mail to: transactions@computer.org, and reference IEEECS Log Number S95119.

- the familiarity with notation and structure that comes naturally to them takes time, training, and practice to acquire.

SPECIFICATION 1

The new users name is taken in and an identity number is assigned from the pool of unused numbers. The unused number set is amended and the new pair of user and their number are added to the existing users.

$$
\begin{array}{|l}
\text{— } \mathit{Add} \text{ —}\\
\Delta \mathit{System}\\
\mathit{name?} : \mathit{Person}\\
\mathit{n?} : \mathbb{N}\\
\mathit{message!} : \mathit{Response}\\
\hline
\mathit{n?} \in \mathit{Unused\_Ids}\\
\mathit{name?} \notin \operatorname{dom} \mathit{Users}\\
\mathit{Unused\_Ids}' = \mathit{Unused\_Ids} \setminus \{\, \mathit{n?} \,\}\\
\mathit{Users}' = \mathit{Users} \cup \{\, \mathit{name?} \mapsto \mathit{n?} \,\}\\
\mathit{message!} = \mathit{OK}\\
\hline
\end{array}
$$

Here error messages are generated by the failure of either of the two preconditions in the Add schema.

$$
\begin{array}{|l}
\text{— } \mathit{AddFail} \text{ —}\\
\Xi \mathit{System}\\
\mathit{name?} : \mathit{Person}\\
\mathit{n?} : \mathbb{N}\\
\mathit{e\_message!} : \mathit{Response}\\
\hline
(\mathit{name?} \in \operatorname{dom} \mathit{Users} \land \mathit{e\_message!} = \mathit{name\_in\_use}) \lor\\
(\mathit{Unused\_Ids} = \varnothing \land \mathit{e\_message!} = \mathit{no\_id\_available})\\
\hline
\end{array}
$$

Finally the behaviors are combined.

$$\mathit{AddUser} \mathrel{\hat=} \mathit{Add} \lor \mathit{AddFail}$$

Questions

1) What conditions give rise to error messages?
2) The size of which set would give you the number of current users on the network?
3) Which set or sets give you information about the total number of users the network will support?

Fig. 1. A sample specification with its accompanying questions.

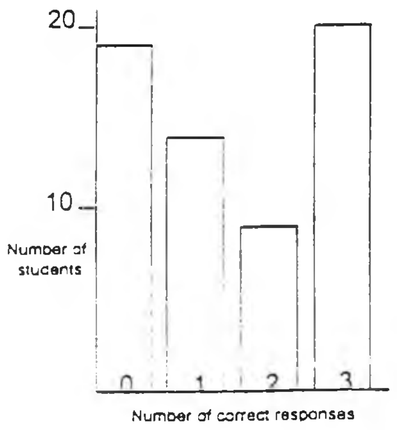


Fig. 2. A chart showing scores obtained.

ACKNOWLEDGMENTS

I would like to thank N. Fenton for his encouragement and advice and A. Fedorec for his collaboration with the experimental work.

REFERENCES

[1] J.B. Wordsworth, "Education in formal methods for software engineering," *Information and Software Technology*, vol. 29, Jan.-Feb. 1987.

[2] "Educational Issues relating to formal methods," *Education Issues Session of Z Users Meeting '94*.

[3] C.A.R. Hoare, "Maths adds safety to computer programs," *New Scientist*, pp. 53-56, Sept. 1986.

[4] J.A. Hall, "Seven myths of formal methods," *IEEE Software*, vol. 7, pp. 11-19, Sept. 1990.

[5] K. Hurst, "Consequences of GCSE in mathematics for degree studies," *The Mathematical Gazette*, vol. 79, no. 484, pp. 61-63, Mar. 1995

[6] B. Potter, "Formal methods: Needs, benefits and pitfalls," *Proc. Advanced Information System*, pp. 205-210, London, Mar. 1991.

[7] K. Finney and A. Fedorec, "An empirical study of specification readability," *Educational Issues of Formal Methods*, M. Hinchey and N. Dean, eds., Academic Press to appear in 1996.

[8] N. Fenton, S. Lawrence, P. Fleeger, and R.L. Glass, "Science and substance: A challenge to software engineers," *IEEE Software*, pp. 89-95, July 1994.

Kate M. Finney and Alex M. Fedorec **8**

An empirical study of specification readability

8.1 Introduction

Experience in the teaching of mathematics at any level reveals the difficulty that a large proportion the population have with the use of abstracted symbolic notations (Hackney, 1991). Some, who have mastered arithmetic and can cope with geometry, reach an insurmountable barrier with algebra. A recent Assessment of Performance Unit (APU) report shows decreases in U.K. pupil's ability to cope with algebra and number, whilst reporting small increases in ability in geometry and data handling (Burghes, 1992). Reasons cited include: the sudden introduction of modern mathematics; a serious cutback in work involving natural numbers; a massive reduction in the basic algebraic content of the GCSE syllabus and indiscriminate use of calculators both in the classroom and in all examinations (Roy, 1992). The ability to read and interpret information condensed into equations and to be at ease with abstract notation seems to be increasingly rare among the student body. Similar problems occur in the teaching of programming languages when introducing concepts of variables and parameters.

In formal methods the aim is to express the essence of a specification in a mathematically precise form using a concise, well-founded notation, therefore some of these same difficulties will occur. There are many claims and counter claims about formal methods and the ease with which they can be used but very little published empirical work. This paper reports the results of an initial experiment to test some aspects of specifications that may be pertinent to these claims. The aim was to try to quantify the effect of various factors on the ability to read and understand formal specifications and is part of broader on-going work examining and identifying metrics for them. The initial results and some experimental observations are reported in this paper.

8.2 Experimental context

The main factors to consider when undertaking this work were: the formal method to be examined, the attributes to be measured and the subjects of the experiment.

TEACHING AND LEARNING FORMAL METHODS ISBN 0-12-349040-5

One of the frustrations of the existing studies of formal methods is the tendency of researchers to invent new versions of existing notations or even entirely new notations and methodologies. It was decided to use the Z specification language (Spivey, 1992) as the basis for the study to give the results wide applicability. The Z notation is becoming a *de facto* lingua franca within the software engineering and computer science communities. This breadth of applicability in both academia and industry is evidenced by the rapid rise in the number of publications about Z or using Z as a notation and the number of quoted applications. The schema structure of Z also lent itself to the presentation of part of a larger specification whilst still being a meaningful fragment.

As an initial study the main attribute of concern was the readability of the specification. This becomes quite a complex attribute because aspects of comprehension, interpretation and inference can all be considered as part of the skill of reading. This is reflected in the learning process which involves two distinct phases: comprehension and articulation. The testing of reading literature also brings these issues into focus. When children learn to read they are initially tested by reading out loud to see that they have mastered the mechanics of putting the letters together and joining the words into sentences. At this stage no comprehension is tested although some interpretation could be implied by phrasing and inflexion. A more advanced stage of reading is tested by a comprehension test which requires the child to answer questions to see if the passage has been understood, it being assumed that they have already mastered the initial mechanistic stage (Harris and Sipay, 1975). The experiment included both aspects of reading.

The comprehension of a language may be understood in three phases or levels: lexical, syntactic and semantic. By analogy, the way in which the child's reading material is written can make a difference to their ability to understand it. Familiar names which they can easily recognize give them key words to work round. Anyone reading a long novel with complicated Russian names will be familiar with the difficulty of maintaining the sense of the narrative whilst recalling the different characters' names. The presentation of the material also has some bearing on the ease with which it is read. The use of paragraphs and short sentences in the structure of the page will aid the reading process. Finally, illustrations in a book give valuable extra information in a visual form about the meaning of the text and give the child confirmation that their interpretation is correct or cues the child into the correct interpretation.

Formal specifications can become more user-friendly if they are presented in a way which increases readability. In an analogy with a child's reading three aspects of a specification were examined: the use of variable names, the inclusion of explanatory comments and text, and the decomposition of the schemas. These are each factors which have been considered essential to the production of quality software where, as in English prose, the style of writing can illuminate or obscure the ideas which are expressed. It is intuitively appealing and has long been generally accepted that the use of meaningful variable names and appropriate explanatory comments enhance program readability (Rushby, 1980) and the partitioning of large or complex programmes into smaller modules is essential to their intellectual manageability (Wirth, 1974). This has to some extent been confirmed by experiment (Harold, 1986) and (Tenny, 1988).

8.3 The experiment

The subjects of the experiment were undergraduate and postgraduate students at the University of Greenwich. They all had some knowledge of Z and were willing to spend time taking part in an experiment. The uniformity of the background and experience of the students also helped in the control of some of the bias to the results.

The students came from four classes which were, for the purposes of the experiment, called A, B, C and D. The students in A, B, and C were part-time evening computer science undergraduates. Those in Class A were nearing the end of a one semester unit in formal specification using Z and were at level 2, that is, the equivalent of the second year of a full time degree. Nine students participated in the experiment. Five of these where direct entrants having successfully completed an HND with appropriate grades. The sixteen students of Class B were also level 2 but had completed the formal methods unit the previous semester and were now on a theory of computation unit. Class C was in a level 3 software engineering unit, that is equivalent to the final year of a full-time degree. Of these, eleven students participated in the experiment. Their experience of Z had been part of a formal methods course in the previous year covering much of the same ground as the new formal specification unit taught to the current level 2 students. Twenty six students of Class D took part in the experiment. These were a mix of day-release part-time and full-time MSc software engineering students who had had one three-hour lecture on Z with tutorial exercises three weeks prior to the experiment. Five mini-specifications were prepared varying only in the naming of

Table 8.1: Experimental specifications

Specification	Comments	Meaningful names	Single schema
1	Yes	Yes	No
2	Yes	No	No
3	No	No	No
4	Yes	Yes	Yes
5	No	Yes	No

variables, the use of comments and the decomposition into more than one schema. Each specification was randomly assigned a number from a set of random number tables (Murdoch, 1974). These are summed up in Table 8.1. An example of a specification used in the experiment with comments but without meaningful variable names is given in Figure 8.1 and the same specification but without the the comments but with meaningul names is presented in Figure 8.2. The students were each given an instruction sheet and a sheet to record their responses. The

The new user's name is taken in and an identity number is assigned from the pool of unused numbers. The unused number set is amended and the new pair of user and their number are added to the existing users;

$$
\begin{array}{|l}
\hline
\textit{Add} \\
\Delta \mathit{System} \\
x? : \mathit{Ngrp} \\
n! : N \\
m! : \mathit{Res} \\
\hline
n! \in \mathit{Avids} \\
x? \notin \operatorname{dom} \mathit{Net} \\
\mathit{Avids}' = \mathit{Avids} \setminus n! \\
\mathit{Net}' = \mathit{Net} \cup \{x? \mapsto n!\} \\
m! = \mathit{OK} \\
\hline
\end{array}
$$

Here error messages are generated by the failure of either of the two preconditions in the Add schema:

$$
\begin{array}{|l}
\hline
\textit{AddFail} \\
\Xi \mathit{System} \\
x? : \mathit{Ngrp} \\
n! : N \\
e\_m! : \mathit{Res} \\
\hline
(x? \in \operatorname{dom} \mathit{Net} \wedge e\_m! = \mathit{error\_type}1) \vee \\
(\mathit{Avids} = \varnothing \wedge e\_m! = \mathit{error\_type}2) \\
\hline
\end{array}
$$

Finally the behaviors are combined:

$$\mathit{AddUser} \mathrel{\widehat{=}} \mathit{Add} \vee \mathit{AddFail}$$

Figure 8.1: Specification 2

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{Add}} \\
\hline
\Delta \mathit{System} \\
\mathit{name}? : \mathit{Person} \\
n! : N \\
\mathit{message}! : \mathit{Response} \\
\hline
n! \in \mathit{Unused\_Ids} \\
\mathit{name}? \notin \operatorname{dom} \mathit{Users} \\
\mathit{Unused\_Ids}' = \mathit{Unused\_Ids} \setminus n! \\
\mathit{Users}' = \mathit{Users} \cup \{\mathit{name}? \mapsto n!\} \\
\mathit{message}! = \mathit{OK} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{AddFail}} \\
\hline
\Xi \mathit{System} \\
\mathit{name}? : \mathit{Person} \\
n! : N \\
\mathit{e\_message}! : \mathit{Response} \\
\hline
(\mathit{name}? \in \operatorname{dom} \mathit{Users} \land \mathit{e\_message}! = \mathit{name\_in\_use}) \lor \\
(\mathit{Unused\_Ids} = \varnothing \land \mathit{e\_message}! = \mathit{no\_id\_available}) \\
\hline
\end{array}$$

$$\mathit{AddUser} \mathrel{\widehat{=}} \mathit{Add} \lor \mathit{AddFail}$$

Figure 8.2: Specification 5

instruction sheet is reproduced in Figure 8.3. The students were thanked by the experimenter and told the experiment was being conducted as part of research into styles and metrics in formal specification. It was explained that the results would be treated confidentially and not used as part of any assessment. The instruction sheet was read through and any arising questions answered. The experiment was then conducted in two phases. In the first phase the students were asked to answer three questions about a randomly allocated specification and record the time as they responded to each question. These times were later translated into elapsed time. The questions were:

- ♦ What conditions give rise to error messages?
- ♦ The size of which set would give you the number of current users on the network?
- ♦ Which set or sets give you information about the total number of users the network will support?

In the second phase, when this task was completed, each student was given copies of all five specifications and asked to rank them in order of comprehensibility and record the ranking.

8.4 Results

The first analysis of the results was a simple examination of the differences in scores and times between the classes. Class D, as might be expected having had only one brief lesson in Z, were on average slower in completing the first phase of answering three questions. They took an average of 577 seconds. These students were postgraduates and therefore with a higher educational level, but they had also received the shortest amount of teaching time which seems to have led to a longer reading time. Class B, who had completed the course and taken an examination in Z a few months previously performed the best, taking an average time of 363 seconds. One outlier distorts the figures and ignoring this reduces their mean time to just 317 seconds. Class C, surprisingly, did better than A with an average time of 391 seconds as opposed to 541. However class A's times were distorted by an outlying result which if ignored reduced the class average time to 480 seconds. A graph showing the total time taken to complete the reading of the specification by the individual students, and grouped by class, is given in Figure 8.4. Only 58 students are represented here as four failed to provided either a start time or a finish time on their answer sheet and therefore their results had to be ignored. The scores obtained in answer to the three questions are shown in a bar chart (Figure 8.5). This reveals a high proportion (19 out of 62) of the students that could not answer any of the questions correctly, demonstrating a poor understanding of the specification. The bar chart also shows that a similar proportion (20 out of 62) could correctly answer every question implying that with Z, as with a lot of mathematics, it is common that students divide into those who

Instruction Sheet

Do not turn over the question paper until you are asked to.

Thank you for agreeing to participate in this experiment. It is being conducted as part of research into styles and metrics in formal specification. The results will be treated confidentially and will not be used as part of any assessment of you or any other person.

As well as this instruction sheet, you will be given an answer sheet to record your answers on and a question sheet which you must not look at until you are asked to do so. The experiment will take part in two phases.

Phase I
On the other side of the question sheet is a simple Z specification and three questions for you to answer about the specification. We would like you to answer those questions and time your response at each step.

When you are asked to start, turn over the question sheet and record the start time on the answer sheet. There are five different specifications of which you have been allocated one on a random basis. Working as quickly as you can record the number of your specification on the answer sheet and then read through the specification.

When you have finished reading through the specification and you are ready to answer the questions record the time again. Answer each question in turn noting the time as you complete each answer. Raise your hand when you have finished. Do not attempt to modify your answers after you have recorded your finish time.

Phase II
After you have looked at one specification you will be given all five and asked to compare them for comprehensibility.

Taking as much time as you need put the five specifications into order on the basis of how comprehensible you feel they are. Carefully sort the specifications until you feel you have ranked them with the least comprehensible first, and the one which is most understandable last. Write down the numbers of the specifications in order on the answer sheet.

Once again, thank you for your time and effort.

have finished a problem and those who cannot even start. In Figure 8.6 the scores for answering the questions are plotted against the time taken to answer them. The mean times have been marked showing an inverse relationship suggesting that if students could read and understand the specification then they did not need a long time to do so, whereas those who took a long time to answer the questions got more of them wrong. Regression analysis was performed on the scores against the three attributes: comments, names and structure. We would like to note that this was done as an exploratory rather than definitive analysis. Since the response variable is ordered and discrete, a polytomous logistic regression would be an appropriate form of analysis. However, for indicative purposes we restrict ourselves to linear regression fitting by the ordinary least squares method.

The regression equation became:

$$\mathit{score} = 0.974 + 0.259c + 0.676n - 0.059s$$

where:

c is the variable showing the presence or absence of comments;

n is the variable showing the presence or absence of helpful names;

s is the variable showing the presence or absence of blocked schemas.

As a result of the analysis only the use of meaningful or helpful identifier names in the schemas seemed to have a significant effect on the score with a p value of 0.05.

Finally the cumulative rankings of the five specifications in terms of their comprehensibility is shown in Figure 8.7. This clearly shows the specification with comments and meaningful variable names is rated as most comprehensible and that with neither is the least. These match closely the predicted order 1, 4, 5, 2, 3, only differing in the order of specifications 4 and 5. This implies that in order of importance for the readability of a specification, helpful names are more important than comment levels and least important by comparison is the blocking of one or more schemas together. It is clear that different comments may produce different rankings, similarly decomposition of schemas may be more important on large or complex specifications and further investigation does need to be conducted. The data was analyzed using Minitab[1] and any incomplete data was not included. As only four students did not have either scores or total times this had little overall effect. The different classes taking part were used as a blocking factor in the model.

8.5 Conclusions

Large claims for statistical validity and inferences of a far-reaching nature about the readability of Z could not be justified by such a small-scale experiment. However,

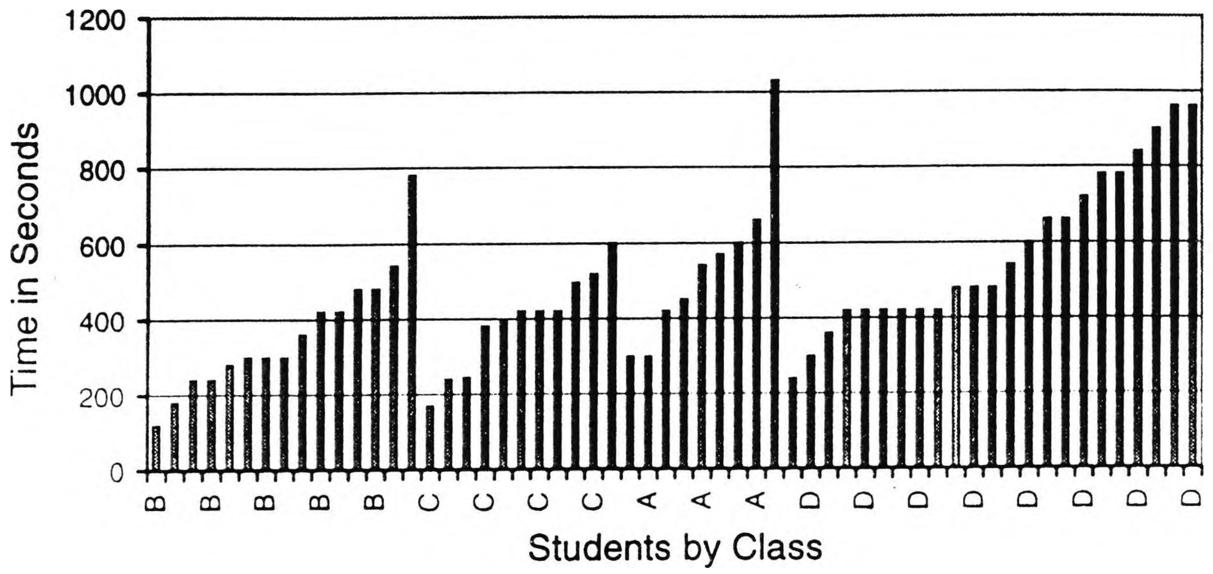


Figure 8.4: Individual student times

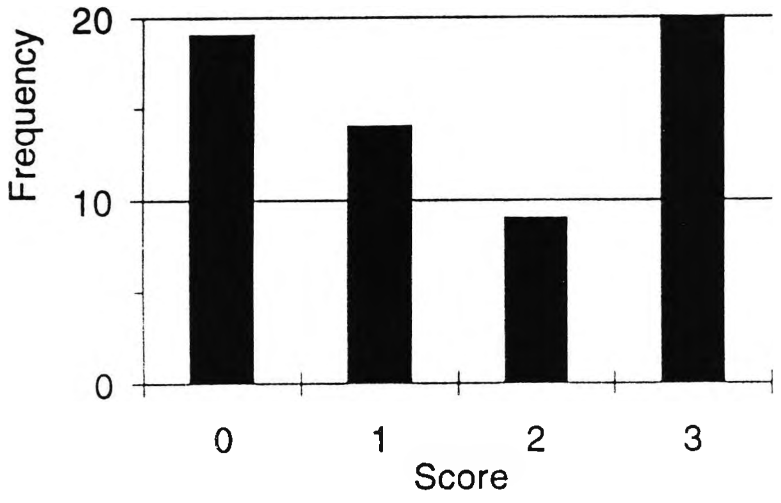


Figure 8.5: Student scores

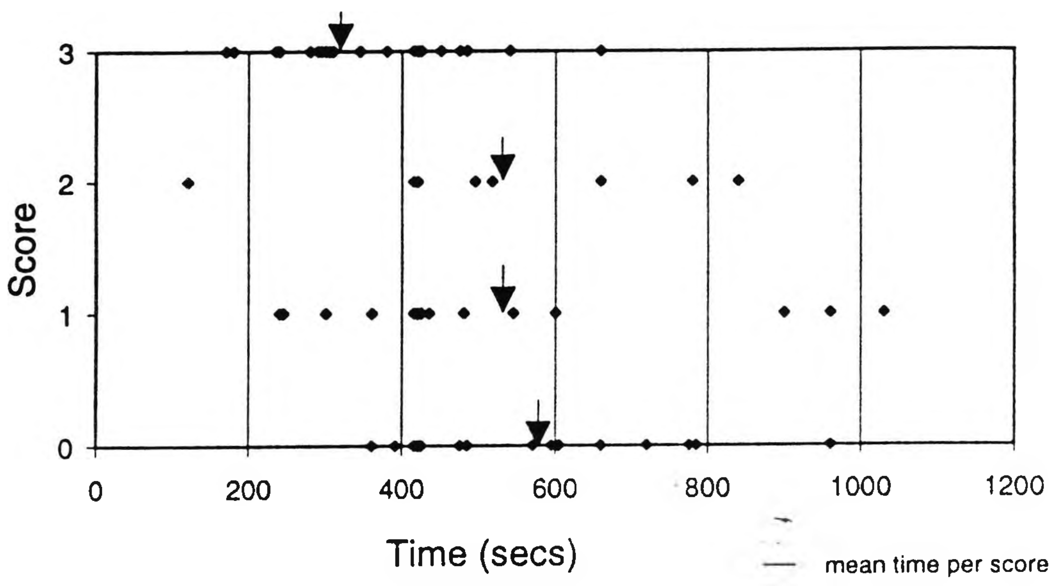


Figure 8.6: Scores and times

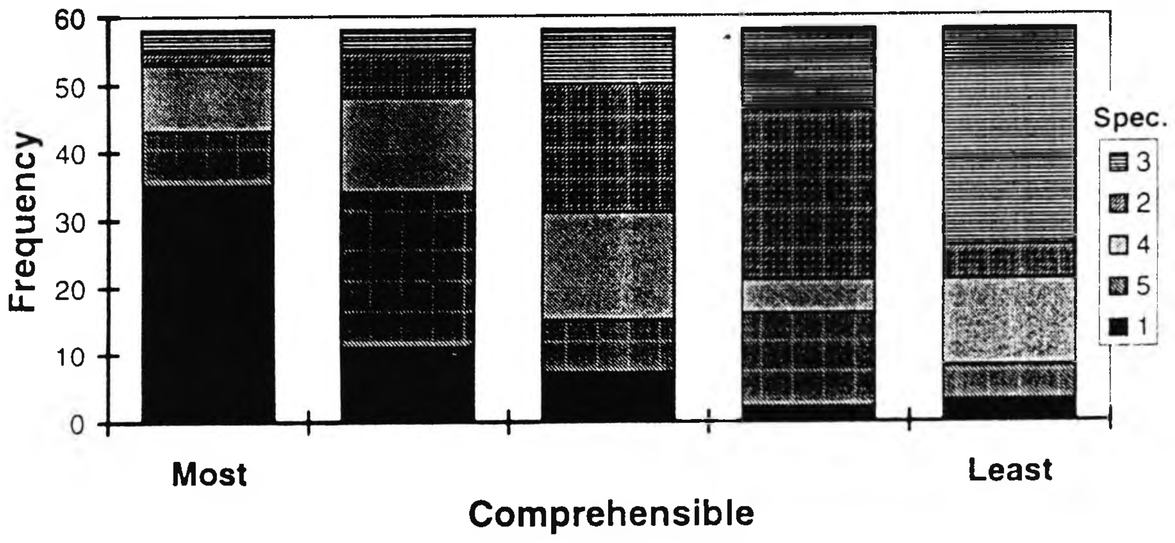


Figure 8.7: Specification rankings

there were sufficient subjects to make some limited observations and to give pointers for further work.

However, it can be concluded that a positive difference can be made to a specification in Z by the use helpful variable naming and the attachment of appropriate comments. Although there are compelling reasons to believe that large specifications are more readable when partitioned by judicious use of schemas, the scale of the specification used in this experiment was not large enough to show this effect.

From this experiment it seems valid to say that several lessons can be learnt for the teaching of formal methods. One should not underestimate the difficulty of reading a formal specification written in a mathematical notation. Specification must be recognized as a unique form of communication between human beings. Every opportunity should be taken to make the reading easier particularly by suitable naming of variables and data. Time to assimilate the techniques involved is important and reading does not always imply semantic comprehension or the higher skill level — the ability to articulate. As it was clear that 19 of the subjects did not understand the specification sufficiently to answer any of the questions correctly despite their background, then we should not expect clients and software engineers to master Z and other formal methods without training.

It is not only for the benefit of the client that the specifications should be read easily but also the author. It is well known in programming that a time lapse can create difficulties for the writers themselves with their own code. Indeed, the poor style of many programmers, use of C, for example, has given it the undeserved label of a write-only language.

Training practitioners from the start to have appropriate names for identifiers and to include suitable explanatory text will ensure good comprehension from clients, fellow software engineers and implementors.

8.6 Discussion

In the Educational Issues session of the 1994 Z User meeting Wordsworth (1994)[2] argued that almost everything that the software engineer does has a mathematical theory that can explain and illuminate it and therefore, that mathematics should be made the teaching medium and basis of an entire education program in software development. He acknowledges that software engineers may not be good at teaching mathematics and their students not good at understanding it. He concludes that universities have a critical part to play in changing this situation by finding ways to improve their teaching and enthuse their undergraduates, or improve selection of undergraduates.

There is no option of changing student selection criteria given the pressures political or otherwise. However, it is possible, as Wordsworth suggests, to look for new ways of improving our teaching and inspiring the students. We have, for example, found the use of Logica's Formalizer tool for writing and type checking Z specifications valuable in providing direct feedback to the student, in much the same way that a compiler provides feedback to a programming student.

This experiment suggests that in teaching formal methods it is crucial that appropriate time and effort is put into the early lexical and syntactic comprehension of the notation, supporting this through semantic cues in good choice of variable names and comments.

Whilst good software engineering does rest on mathematical rigor and formalism, teaching it in the abstracted way of many mathematicians with terse term identifiers will do nothing to enthuse the students, instead it will create feelings of confusion, demotivation and inevitably failure.

8.7 Future developments

The experiment has now been repeated with a further 71 students and although the analysis is not yet complete the preliminary results appear to confirm those presented here. To examine this further more research is needed, on a more in-depth and broader scale.

- A fully interactive approach could be used. This would entail writing further specifications so that all combinations of attributes are covered.

- More diverse subjects could be tested including representatives from industry who employ formal specifications.

- The experiments could be replicated in other institutions to see whether similar results occur.

Other attributes could be included or investigated separately in these further experiments.

The size of the Z specification needs to be increased to test further the effect of schema decomposition on readability and give a more realistic simulation of the type of specification normally encountered. Mitchell *et al.* (1994) suggest that different Z specification structurings can lead either to versions where it is easy to understand the component parts but harder to see how they fit together or to those where it is easier to get an overall view of the system's behavior but harder to understand the individual component parts. This implies that different styles of specification writing might be appropriate for different levels of abstraction and would need to be explored further.

Acknowledgements

The authors would like to thank K. Rennolls of the University of Greenwich for his invaluable help and encouragement with this work.

Notes

1. Minitab is a trademark of Minitab, Inc.
2. A revised version of that paper is to be found as Chapter 1 of this book.

References

Burghes, D. (1992) Recent changes in school curricula and their effect on university mathematics courses: a view from mathematics education, *Bulletin of the IMA*, **28**(1):17–20, January.

Hackney, J. (1991) The internalisation of symbolism, *Bulletin of the IMA*, **27**(10):214–216, October.

Harold, F.G. (1986) Experimental evaluation of program quality using external metrics. In E. Soloway and S. Iyengar, editors, *Empirical Studies of Programmers*, pp 153–167, Ablex Publishing Corp., New Jersey.

Harris, A.J. and Sipay, E.R. (1975) *How to Increase Reading Ability, A Guide to Developmental and Remedial Methods*, 6th Edition, McKay.

Mitchell, R., Loomes, M. and Howse, J. (1994) Structuring formal specifications — a lesson relearned, *Microprocessors and Microsystems*, **18**(10):593–599.

Murdoch, J. and Barnes, J.A. (1974) *Statistical Tables for Science, Engineering, Management and Business Studies*, 2nd Edition, Macmillan, London.

Roy, S. (1992) *No Maths Please*, *Bulletin of the IMA*, **28**(1):26–27, January.

Rushby, N. (1980) Other people's programs. In B. Meek and P. Heath, editors, *Guide to Good Programming Practice*, pp 152–159, Ellis Horwood, Chichester.

Spivey, J.M. (1992) *The Z notation: A Reference Manual*, 2nd Edition, Prentice Hall International Series in Computer Science, Hemel Hempstead.

Wirth, N. (1974) On the composition of well-structured programs, *Computing Surveys*, **6**(4):247–259.

Wordsworth, J.B. (1994) Educational issues relating to formal methods. In Proceedings of the Educational Issues of Formal Methods Session of the 1994 Z User Meeting, St. John's College, Cambridge, June, pp 29–36; included in revised form as Chapter 1 of this book.

NORTH-HOLLAND

Evaluating the Effectiveness of Z: The Claims Made About CICS and Where We Go From Here

Kate Finney
University of Greenwich, Wellington Street, Woolwich, London SE18 6PF England

Norman Fenton
Centre for Software Reliability, City University, Northampton Square, London EC1V 0HB England

There have been few genuine success stories about industrial use of formal methods. Perhaps the best known and most celebrated is the use of Z by IBM (in collaboration with Oxford University's Programming Research Group) during the development of CICS/ESA (version 3.1). This work was rewarded with the prestigious Queen's Award for Technological Achievement in 1992 and is especially notable for two reasons: 1) because it is a commercial, rather than safety- or security-critical, system, and 2) because the claims made about the effectiveness of Z are quantitative as well as qualitative. The most widely publicized claims are: less than half the normal number of customer-reported errors, and a 9% savings in the total development costs of the release. This paper provides an independent assessment of the effectiveness of using Z on CICS based on the set of public domain documents. Using this evidence, we believe that the case study was important and valuable, but that the quantitative claims have not been substantiated. The intellectual arguments and rationale for formal methods are attractive, but their widespread commercial use is ultimately dependent upon more convincing quantitative demonstrations of effectiveness. Despite the pioneering efforts of IBM and PRG, there is still a need for rigorous, measurement-based case studies to assess when and how the methods are most effective. We describe how future similar case studies could be improved so that the results are more rigorous and conclusive. *© 1996 by Elsevier Science Inc.*

Address correspondence to Norman Fenton, Centre for Software Reliability, City University, Northhampton Square, London ECIV 0HB United Kingdom.

1. INTRODUCTION

In the last 10 years, there has been fierce debate within the software engineering community about the merits of using formal methods. The authors of this paper have been actively involved in this debate: on the one hand by doing research and technology transfer in this field (Fenton and Mole, 1988; Fenton and Hill, 1992), and on the other by doing empirical assessment of formal methods (Pfleeger et al., 1995, Finney, 1996). We therefore feel well placed to comment on the state-of-the-art of industrial use of these methods. While there is a grudging acceptance of their usefulness in safety and security critical applications, there is no such consensus for commercial applications. Here the perceived benefits in terms of improved reliability are outweighed by perceived overheads in development costs and training. In fact, the present position can be summarized as follows.

- Most formal methods have not penetrated far from their academic roots.
- Very few companies in the world use any formal methods systematically.
- Most major IT companies have no plans to use formal methods.
- The rare industrial uses of formal methods are restricted to formal specification; there is almost no evidence of any serious attempt at formal development or program proof. Thus, the original raison-d'être of formal methods has not been seriously tested.

0164-1212/96/$15.00
PII S0164-1212(96)00122-7

- Although not widely used, there is now reasonably widespread awareness of two notations, Z and VDM, thanks mainly to the existence of training resources in these.
- The need to apply formal methods on security- and safety-critical systems is beginning to be recognized by some regulatory and standards authorities (Bowen and Stavridou, 1993), but there are extremely few documented examples of such use.

Part of the reason for the lack of penetration is the near absence of convincing empirical evidence to support the benefits of using formal methods (Fenton et al., 1994). However, one ongoing study has been routinely cited in many papers on formal methods as a rare example of how Z has been used effectively on a large and important commercial system. This is IBM's CICS/ESA (version 3.1) which was released in 1990. The project was prominent in the international survey of industrial uses of formal methods (Gerhart et al., 1993; Craigen et al., 1995) as the largest commercial application.

The quantitative claims made about the effectiveness of using Z on CICS are impressive. The most notable are

- 2.5 times fewer customer-reported errors;
- 9% saving in the total development costs of the release.

The cost savings is particularly important because, while reliability improvements are an expected benefit of using formal methods, there has been no such expectation of economic benefits. Indeed, Bowen and Hinchey (1995) cited this cost savings in the CICS study as the main counterexample to the 'myth' that 'formal methods delay the development process'. Because of the high profile nature of the project, the claims about CICS have had far-reaching ramifications on the public perception of formal methods (it is also likely that UK research funding agencies have been influenced by the claims). For example, in 1992, IBM together with the Oxford University Programming Research Group won the highly prestigious Queen's Award for Technological Achievement. The citation includes the following text (our italics).

> Oxford University Computing Laboratory gains the Award jointly with IBM United Kingdom Laboratories Limited for the development of a programming method based on elementary set theory and logic known as the Z notation, and its application in the IBM Customer Information Control System (CICS) product. *The use of Z reduced development costs significantly and improved reliability and quality.*

In light of the claims made about the use of Z in CICS, it seems reasonable to examine the evidence rigorously. There are a number of public domain reports about the study, although none have appeared in the major software engineering publications. Of these, three focus to some extent on the evidence to support the effectiveness of the use of Z (Phillips, 1989; Collins et al., 1991; Houston and King, 1991). These reports are not easily accessible (appearing mainly in relatively minor conference proceedings). Our first aim is therefore to bring together these fragmented results from an independent perspective. Thus, in Section 2 we describe the background to CICS/ESA and the way that Z was used during the development. In Section 3 we describe the specific claims that have been made, and we analyze these in the light of the evidence that is publicly available. We conclude that, while there are some grounds for optimism, the material in the public domain does not support the strong quantitative claims that have been made. The problems with the quantitative evidence on CICS are a direct result of an inadequate case study set-up and an ad-hoc approach to measurement. These weaknesses in turn are probably explained by the fact that the project was set up before the issues of empirical validation and software measurement were as widely accepted as they are today. The empirical problems we identify are solvable for future similar studies simply by adhering to well-known best practice in software experimentation and measurement. With a small amount of additional investigation, the problems could even be partly resolved retrospectively for CICS. This improved approach to assessment is discussed in Section 4. It is our firm belief that the future of formal methods is critically dependent on these kinds of quantitative studies revealing benefits. The claims made for Z in CICS are exceptionally strong but need to be substantiated. Studies (conducted along the rigorous lines we propose) that produce even modest quantifiable improvements advance the cause of formal methods. Indeed, in our own work on the SMARTIE project (Pfleeger et al., 1995) we applied such an approach to demonstrate small but significant quality benefits of using formal methods on a major commercial system.

2. BACKGROUND TO THE CICS STUDY

CICS is the Customer Information Control System developed by IBM. It is an on-line transaction processing system with many thousands of users worldwide. It was originally developed in 1968, with new releases approximately every two years since then.

In 1983 IBM decided to invest in a major restructuring of the internals of CICS. Two years earlier Professor Tony Hoare of the Oxford University Programming Research Group had by chance met Tony Kenny, the IBM CICS manager at Hursley Park. This resulted in a research contract between IBM and PRG in 1982 to study the application of formal techniques to large-scale software development. In the initial studies undertaken by PRG, two notations were used for comparison, CDL (IBM's internal Common Design Language) and Z (the specification language developed at PRG). Primarily because of this research it was decided that Z would be used for most of the new code required in the restructuring of CICS.

In the event when CICS/ESA version 3.1 came out in June 1990 it included (Houston and King, 1991)

- 500,000 lines of unchanged code;
- 268,000 lines of new and modified code including

37,000 lines 'produced from Z specifications and designs' and

11,000 lines partially specified in Z

(2,000 pages of formal specifications in total).

In none of the published reports is it made clear what proportion of the 'Z code' was modified (and hence, respecified based on a previous version) compared with new code. We shall see that this is one of several omissions that have serious implications on the results.

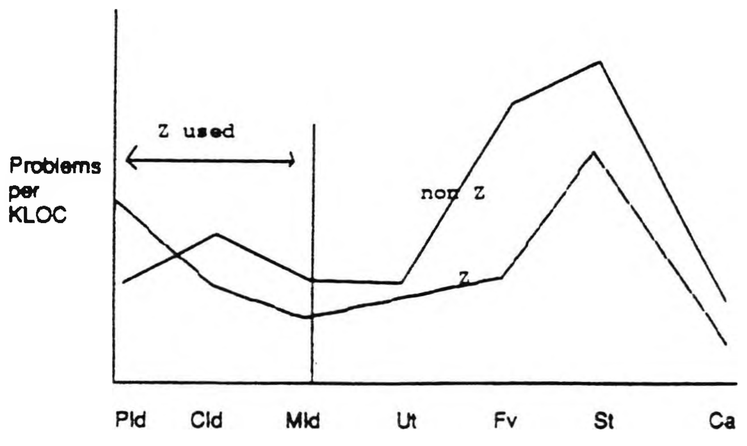
There had been a thorough program of education and training in the use of Z for the IBM personnel, and this was concentrated on the specification and design techniques. Throughout the project, help was available from the Oxford team; workshops, tutorials, and a specially written Z manual were provided. There was only limited use of refinement techniques and very little proof. Specifications and one or two levels of design were written in Z, but a notation based on Dijkstra's guarded commands (Dijkstra, 1975) was used to express designs and bridge the gap to procedural code. In most cases, there was no formal relationship between the stages, and noting the preconditions was the only attempt at rigour.

3. THE QUANTITATIVE CLAIMS MADE

The claims for the benefits of the use of Z in this project seem to be based on the comparison between the development of the code from the specifications where Z has been used and that where no Z was involved (see Figure 1). This still leaves a number of doubts about exactly what is being compared with what (see below), but no matter what assumptions are being made, it is clear that the 'Z data' is defined on a relatively tiny amount of code compared with the 'non-Z data'. It is also clear that the Z modules were not chosen at random. From an experimental viewpoint, these are major (related) problems that could have been easily averted.

Claim 1. Fewer Problems Overall During Development

Figure 1 (which is copied from the only graph given in the literature) is a comparison of the reported 'problems per KLOC' (Thousands of Lines of Code) at each key phase of the project development. These phases are defined as part of the standard IBM



Pld	Product level design	Ut	Unit Test
Cld	Component level Design	Fv	Function Verification
Mld	Module level design	St	System test
		Ca	Customer Availability

Figure 1. Comparison between Z use and non-Z use.

development life cycle. Phillips (1989) draws the graph up to the system test phase, but Houston and King (1991) adds the customer availability data (eight months after release).

The problem rate appears to be larger in the early stages of development with the code derived from Z specifications, but in the later stages of the life cycle of the software, it has fewer errors. This is consistent with our own findings when we analyzed the effectiveness of formal methods on another industrial system as part of the SMARTIE project (Pfleeger et al., 1995). It is proposed that the reason for this is that by using Z, specifiers are forced by the rigour of the notation to tackle all the complexities of the problem and therefore make a larger proportion of their mistakes at this stage. In contrast, the non-Z users have developed code which has errors in it right up to the final stages of the process when fixing errors will be more expensive.

The most serious concerns about the validity of this claim are

- The omission of a scale on the vertical axis of the graph in Figure 1.
- It is not clear what is being compared

37K lines with Z: 268–37K lines without Z;
37K lines with Z: 268–37–11K lines without full Z;
37K lines with Z: 268 + 500K lines without Z.

Houston and King (1991) assert that 'Since similar measurements were made on the non-Z code on this release and on all the code in previous releases, meaningful comparisons are possible.' From this assertion, it is our understanding that the most likely comparison being made is between 37KLOC of Z and 768KLOC without Z, but the public papers do not make this clear. This basis for comparison is the most worrying scenario since the non-Z code is heavily biased with old code which has problem reports dating back many years. In particular, the number of post release problems for such code must inevitably be higher, but these cannot have been included in the Z code from the timescale involved in the graph.

- The fact that we do not know what proportion of the Z specification relates to new modules (as opposed to just changed) is really crucial. Where an existing module is being respecified, we would expect to see a significant drop in problem reports compared to both the previous version and the baseline. This is true irrespective of the specification method used. After all, IBM has many years of experience with the existing code and has extensive knowledge of where the problems lie; this is a major rationale for making changes. If, as it seems likely, the proportion of changed code in the Z specified portion is significantly different from the proportion of changed code in the non-Z portion, then this is further evidence that the problem density data between the Z and non-Z code cannot be meaningfully compared.
- As any involvement from Z stops after the 3rd stage (module level design), it is not clear what effect Dijkstra's language had and whether it was used on the non-Z code.

It is not clear if the two types of code were subjected to the same types of inspections and testing (it is highly unlikely this would be possible, given the very different nature of the documents). Consequently, it is not clear if the class of problems being reported (especially prior to Ut) are comparable. Nor do we know how problems were counted. If an error originated in a schema, was it counted with every inclusion? It is also certain that not all errors are equally serious (there appears to have been no attempt to weight them by fixing time or severity); because of this, the absence of a scale in Figure 1 is an even more serious omission.

- It is not clear what KLOC means for non-code documents. Every phase prior to Ut (Unit Test) in Figure 1 involves a document which is not code. The most likely explanation is that the KLOC at, say Pld (product level design), is measured as the KLOC in the code that is eventually implemented from the design. Also, it is not clear if comments are included in KLOC and if this is significantly different in the two types of code.
- The increased supervision of those writing in Z may have been a factor as they were employing a completely new technique.
- The expertise available from PRG on the Z parts may have been a critical factor.

Claim 2. Development Savings

Of all the claims made about the benefits of using Z on CICS, the most impressive and surprising is the well publicized 9% reduction in overall project costs. Specifically, Houston and King (1991) assert

> 'IBM has calculated that there is a reduction in the total development cost of the release. Based on the reduction in programmer days spent fixing problems, they estimate a 9% reduction as compared to developing the 37,000 lines without Z specifications.

The 'problems' on which the figure is based are restricted to those discovered during development;

we believe that they do not include post-release user reported problems. Houston and King (1991) do not state this, but a very similar quote using the 9% figure appears in Phillips (1989) which was published before the new release (in fact, Phillips' version of Figure 1 only goes as far as the system test).

The concerns we have already expressed about the basic comparison between Z and non-Z derived code make the 9% figure already seem less convincing. However, we are also concerned about how the figure was calculated. The second sentence in the Houston and King (1991) quote is as much detail as is presented in all the papers about this, so we can only speculate on what it really means. There is a suggestion that the extra time spent on doing the Z specifications (not to mention training, etc.) is not accounted for. This is a gross omission. We are also concerned about the word 'estimation'. It suggests that no actual data on time to fix the problems was recorded. Rather, it appears that an IBM standard 'cost to fix a problem' was used. Let this cost be c. Now suppose the problem density of the portion of code derived from the Z specification was x KLOC and the problem density of the portion of code derived from the non-Z specification was y KLOC. Then it appears that they have computed

$$c*37*(y - x)$$

and that this comes to 9% of the actual total cost of the release.

It is still unclear from the literature how much comparative measurement went on and what statistics were collected.

Claim 3. Customer Benefits

Houston and King (1991) assert

'...the figures on number of problems reported by customers are extremely encouraging: in the first 8 months after the release was made available, the code which was specified in Z seems to have approximately 2.5 times fewer problems than the code which was not specified in Z. These figures are even more encouraging when it is realised that the overall number of problems reported is much lower than on previous releases. There is also evidence to show that the severity of the problems for code specified in Z is much lower than for the other problems'.

There are actually three claims being made here that we address in turn.

1. That the Z code has 2.5 times fewer problems than the non-Z code. The 2.5 figure is, presumably the differential shown in the last phase of the graph of Figure 1. The authors do warn that

   'the length of time and the size of the sample mean that figures available so far should be treated with some caution...that many customers do not change immediately to a new release when it is made available'

   This is a very obvious drawback to the 2.5 claim. We have already noted that the non-Z code contains a very high proportion of old and well-used code, which would inevitably attract more problem reports. All of the Z code is either new (hence, contains new functionality not yet much used on IBM's own admission) or changed (hence, inevitably reducing the number of known problems). It is therefore our view that the 2.5 figure should be treated with more than just 'some caution'.
2. The overall number of problems reported is much lower than on previous releases. In fact, it is not clear what is being compared. The problem reports for the new release only cover the first eight months; the problem reports for previous releases cover their entire lifetime (in each case this is at least two years). It is therefore inevitable that the number of problems reported is much lower. However, if the comparison is genuine (that is, if the comparison is with the number of problems reported in previous releases restricted to their first eight months), then it is strange that IBM has not provided details of the data. There seem to be no subsequent reports to describe what happened after the first eight months.
3. The severity of the problems for code specified in Z is much lower than for the other problems. Unfortunately, the 'evidence' is not presented at all. For example, we do not know if this is based on a subjective classification of the problems or on something more scientific, like the relative time to fix the problems.

4. COMPARING THE CASE STUDY WITH BEST PRACTICE

IBM does not claim to have run a scientific experiment to evaluate the effectiveness of Z. To a certain extent this is understandable because when the project began, the necessary knowledge of empirical software engineering and software measurement were not widely understood. However, it is important to investigate just how the evaluative study was conducted, and to identify areas where it could have been improved (and indeed may still be improved).

An important contribution has recently been made by Kitchenham et al., in the study of experimental validation of software tools and techniques. Kitchenham et al. (1995) argue that in order to be able to draw significant inferences from a case study, certain basic steps must be taken so that the conclusions have a valid statistical basis. In carrying out such a case study, they have suggested guidelines to follow in the design and administration stage, and a checklist to run through in the planning stage. With these in mind, we shall reassess the CICS project and its importance as a case study in the use of formal methods.

Aims and Objectives

The first stage in planning a project to assess the use of Z in CICS should have been to define clearly the objectives and to state the hypothesis that was to be tested.

To some extent this was done. The literature states that the incorporation of Z into the new CICS release was intended to improve the quality of the product and extend its life (Phillips, 1989). The aim of the statistics collected should have been to lead to a conclusion about whether this had been achieved. However, the baseline against which any comparisons were made is not at all clear. Some of the literature seems to imply comparisons were only made within the project (Phillips, 1989), while other writers hint that measurements made on other similar projects were used for comparison (Houston and King, 1991).

The lack of clarity over the subjects for the comparison reinforces the idea that the aims and objectives were not really thought out at the start. External project constraints affecting the aims are not mentioned. The effect of working very closely with an academic group, PRG, whose expertise was on call throughout and whose aims might not have been primarily commercial is not taken into account. There is mention of the shift in the emphasis of the project with the introduction of Z, but no details are given for comparison (Collins et al., 1991). It is also not clearly stated if the time constraints imposed by the extra training in Z are taken into account in evaluating the project budget.

Collins et al. (1991) widen the aims of the case study to say that its purpose is to address the issues of

whether aspects of large and complex systems could be captured using mathematics;

whether there would be practical benefits from doing that;

who in the development team should use the methods;

what training should be given;

what tools would be required.

Analysis of these issues is vague, however. Collating the various views and intentions together, the simplest hypothesis adopted seems to have been by implication:

> Using formal methods in the CICS project would improve the quality of the resulting software and reduce development and maintenance costs.

Measurements

The response variables and how they are to be measured should be the next items on a case study checklist. Here the literature is divided on what was intended before the study and what is reported to have taken place.

Collins' paper, written for a conference in July 1988 at a stage when the implementation of the new version of CICS was still underway, clearly states their intentions (listed below).

(1) To collect data to consider for comparison

the CICS process before the introduction of Z;

the CICS process using Z;

processes used by similar products using other methods.

(2) To measure in the specification and design phase

time spent on producing documentation;

size of documentation;

resources spent on inspections;

number of problems found and their severity.

(3) To measure at the coding stage

time spent writing and testing code;

number of lines of code produced;

problems at each stage (unit test, function test, etc.);

problems after shipping;

Categorization of problems by type (specification, design...);

Categorization of problems by severity.

There does not appear to be a follow up paper where the results of these measurements are reported.

As discussed earlier, Phillips (1989) gives a single graph which shows results of a comparison of problems per KLOC through the development process as far as the system test. The other data tables in the paper show lines of code and schema variables counted, but no conclusions are drawn from this data. Houston and King (1991) use the same graph but extend the horizontal scale to show problems per KLOC up to the customer availability stage.

They also mention that measurements were taken relating to

time spent producing specification and design documents;

the size of documents;

the resources spent on inspection;

the number of problems found;

the severity of problems found.

No results are given, however, and all claims seem to be made on the number of problems found. There is also mention made of similar measurements made on non-Z developed code on this release and comparisons with all code in previous releases, but again, details are elusive.

From the diagrams available, it is clear that the measurements were made at several development stages, but the latest stage reported seems to be eight months after the release, and in fact, the 9% statistic quoted is in the paper written before that release, so customer feedback could not have been included.

Practical Issues

In Section 3 we raised the concern about the fact that the 'Z data' is defined on a relatively tiny amount of code, compared with the 'non-Z data'. From an experimental viewpoint, this is a major problem that could have been easily averted. Ideally, a set of similar modules should have been randomly assigned in equal numbers to Z and non-Z developments. Literature on the statistical design of experiments would provide models for this (Montgomery, 1984). Even retrospectively it may be possible to salvage something by selecting a 'similar' non-Z module for each of the Z modules and then restricting the comparative analysis with non-Z code to the chosen modules.

The basis under which Z was incorporated into the CICS project is not clear. The criteria for the selection of the modules to be specified using Z is not made explicit. The complexity, length, functionality, and isolation of these modules are factors which should have been taken into account. Houston notes that unless this is clear, any comparison of error rates is meaningless.

The team that worked on Z (originally a team of about 12) were separate from the other project workers, but we are not told how the team was chosen and whether they had particular specialisms or expertise which would affect the outcomes. Collins does say that accelerated development of selected modules was achieved by using two senior and experienced developers.

There seems to have been an initial intention to use Z for specification and take this through design to code. However, in practice, no rigor was applied to the interface between the different levels of abstraction, and in particular, the links between Z and Dijkstra's language of guarded commands do not appear to have been formalized.

So far we have looked at the aims, hypothesis, and some possible confounding factors. As a basis for a study which makes claims about the improvement derived from using Z, there seem to be deficiencies. The project did not have a clear basis for the factors that were to be measured, and in the published work, the single relevant statistic quoted is based on an ill-defined comparison.

The only measurement given on which to base the claims is the problem per KLOC of the Z specified code compared with that of non-Z. It is not clear how far this goes in validating the first part of the hypothesis. Using problems per KLOC as a single 'quality' measure has many well-known pitfalls (see Fenton (1996) for a comprehensive discussion).

If the second part of the hypothesis concerns maintenance and the long term future and development of the code, it is unfortunate that there have been no studies published about customer reaction and experience. Houston and King (1991) remind us that when a new release takes place, many customers do not change over immediately, and even those that do, will not use all the new functionality straight away. They also state that the overall number of problems reported is lower than on previous releases. The fact that well over 50% of the code was unchanged from the previous release may invalidate any argument based on that statistic. Also given is a comment on the severity of the problems, but there are no figures or details to back this up.

5. CONCLUSIONS AND RECOMMENDATIONS

The CICS experience is widely regarded as the most significant application of formal methods to an industrially sized problem. The claims made about the effectiveness of using Z on this project are highly impressive and often quoted. It is therefore essential that the claims are substantiated with rigorous quantitative evidence. We have found that the public domain articles do not provide such evidence. We believe the following are needed if any firm conclusions are to be drawn.

- An update with the results for the 1990 release under further customer experience.
- Clarification of the basis under which the measurements were made.
- More details of the analysis of the comparative statistics mentioned in the papers.

The formal methods community should address the problems of measuring the effect of incorporating their ideas into large-scale projects if there is to be a significant shift towards their adoption. Business needs hard evidence to convince them that the outlay in training and front loading of effort are worth the benefits in the resulting software. We believe this can be achieved without the prohibitive expense of conducting a formal experiment. We adopted such an approach in the SMARTIE project (Pfleeger et al., 1995; Pfleeger and Hatton, 1996) with the effect that we were able to quantify rigorously the moderate benefits (in terms of improved operational reliability) achieved with formal specification using VDM and CCS. Thus, we recommend a careful case study approach, which involves the following.

Have clear aims.

Set up a hypothesis.

Minimize external factors.

Minimize confounding factors.

Plan and collect relevant measurements.

Analyze and report the results.

For all its obvious qualitative benefits, the CICS study only managed to be moderately successful in one or two of the above case study criteria. This partially explains why the claimed quantitative benefits of using Z in this study have not had the expected impact on current industrial practice.

ACKNOWLEDGMENTS

We are indebted to Martin Neil and Bev Littlewood for suggesting improvements to the first draft of this paper. This work was supported in part by the EPSRC and DTI via the DATUM project (grant GR/H89944) which formed part of the Safety Critical Systems Programme, and in part by the CEC under the ESPRIT project DEVA.

REFERENCES

Bowen, J. P., and Stavridou, V., Safety Critical Systems, Formal Methods and Standards, *Software Eng J.* 8(4), 189–209 (1993).

Bowen, J. P., and Hinchey, M. G., Seven More Myths of Formal Methods, *IEEE Software.* 12(3), 34–41 (July, 1995).

Collins, B. P., Nicholls, J. E., and Sorensen, I. H., Introducing formal methods: the CICS experience with Z, *Mathematical Structures for Software Engineering*, Clarendon Press, Oxford, 1991, pp. 153–164.

Craigen, D., Gerhart, S., Ralston, T., Formal Methods Reality Check: Industrial Usage, *IEEE Transactions on Software Eng.* 21(2), 90–98 (1995).

Dijkstra, E. W., Guarded Commands, Nondeterminancy and Formal Derivation of Programs, *Communications of the ACM.* 18(8), 453–457 (August 1975).

Fenton, N. E., and Hill, G., *Systems Construction and Analysis: A Mathematical and Logical Approach*, McGraw Hill, 1992.

Fenton, N. E., and Mole, D., A Note on the Use of Z for Flowgraph Decomposition, *J. Information & Software Tech.* 30(7), 432–437 (1988).

Fenton, N. E., Pfleeger, S. L., and Glass, R., Science and Substance: A Challenge to Software Engineers, *IEEE Software.* 11(4), 86–95 (July, 1994).

Finney, K., Mathematical Notation in Formal Specification: Too Difficult for the Masses?, *IEEE Transactions on Software Engineering.* 22(2), 158–159 (1996).

Gerhart, S., Craigen, D., and Ralston, T., Observation on industrial practice using formal methods. *Proc. 15th Int. Conf. Software Eng.*, 24–33, *IEEE Computer Soc.*, 1993.

Houstan, I., and King, S., CICS Project Report: Experiences and results from the use of Z in IBM. In: Proceedings of the 4th International Symposium of VDM Europe Springer-Verlag Vol 1: Conference Contribution pp. 588–596, 1991.

Kitchenham, B., Pickhard, L., and Pfleeger, S. L., Case Studies for Method and Tool Evaluation, *IEEE Software.* 52–62 (July, 1995).

Montgomery, D. C. *Design and Analysis of Experiments*, Wiley, 1984.

Pfleeger, S. L. and Hatton, L., How do Formal Methods Affect Code Quality?, *IEEE Computer.* (May, 1996).

Pfleeger, S. L., Page, S., Fenton, N. E., and Hatton, L., Case study results for the SMARTIE project, Deliverable 2.2.4 (Additional Study), CSR, City University, March, 1995.

Phillips, M. H., CICS/ESA 3.1 Experiences, In: Proceedings of the 4th Annual Z User Meeting. Springer-Verlag Workshops in Computing, pp. 179–185, 1989.

ELSEVIER

The Journal of Systems and Software 42 (1998) 3–15

The Journal of Systems and Software

Measuring the comprehensibility of Z specifications

Kate Finney *, Keith Rennolls, Alex Fedorec

School of Computing and Mathematical Sciences, University of Greenwich, Wellington Street, Woolwich, London SE18 6PF, UK

Received 19 November 1996; received in revised form 6 February 1997; accepted 13 February 1997

Abstract

The effects of natural language comments, meaningful variable names, and structure on the comprehensibility of Z specifications are investigated through a designed experiment conducted with a range of undergraduate and post-graduate student subjects. The times taken on three assessment questions are analysed and related to the abilities of the students as indicated by their total score, with the result that stronger students need less time than weaker students to complete the assessment. Individual question scores, and total score, are then analysed and the influence of comments, naming, structure and level of student's class are determined. In the whole experimental group, only meaningful naming significantly enhances comprehension. In contrast, for those obtaining the best score of 3/3 the only significant factor is commenting. Finally, the subjects' ratings of the five specifications used in the study in terms of their perceived comprehensibility have been analysed. Comments, naming and structure are again found to be of importance in the group when analysed as a whole, but in the sub-group of best performing subjects only the comments had an effect on perceived comprehensibility. © 1998 Elsevier Science Inc. All rights reserved.

1. Introduction

Formal methods have been proposed as part of the software development lifecycle for many years but there is still much debate on the benefits their inclusion will produce and the claims that can be made, (Hall, 1990; Saiedian, 1996; Bowen, 1995; Cohen, 1989). Their widespread adoption in industry will only take place when there is convincing evidence of their advantages and when the methods and notation are more accessible to a wide range of users. Comprehensibility in terms of reading and understanding a specification written in a formal notation is one of a number of factors that may affect the willingness of practitioners to adopt these methods. To add to the debate it is necessary that well designed empirical studies should be conducted to demonstrate aspects of the use of formal methods within a software development project. Their results will help identify those factors which produce benefits, or drawbacks, and give some quantifiable evidence to augment the arguments based on the valuable experiential knowledge we already have.

There are obvious difficulties in carrying out experimental work in the context of a large software project. The development costs of software are high enough without supporting duplication or control projects. Nevertheless if evidence is to be produced to support the claimed improvements of a particular method, software engineers must face the issue of how this is to be obtained (Fenton et al., 1994).

As Craigen et al. (1995) have noted, studies that are carried out on the application of formal methods within large scale projects frequently fail to collect relevant data on the benefits and drawbacks. Those case studies which have collected data are often statistically inadequate, not following the good practice in the design of factorial experiments as shown by Kitchenham et al. (1995).

Within a large project a relatively small group may be involved in writing a specification, but if that specification is to be useful for refinement, as a basis for testing or for verification of a client's requirements, a larger group of people should be able to read the documents and understand the notation. The appropriate education of future software engineering practitioners in this area will be vital if there is to be widespread adoption of formal methods in the industry beyond specialised research and development departments. It is necessary to equip these future practitioners with a broad knowledge of the latest techniques which may be required, including the use of a formal requirements specifi-

* Corresponding author. Tel.: +44 181 331 8700; fax: +44 181 331 8665; e-mail: k.finney@greenwich.ac.uk.

cation notation, for example Z. (Spivey, 1992; Woodcock, 1989). Z is model oriented, and its use involves identification of key parts of a proposed system and the provision of predicate logic statements to describe their structure and behaviour under certain conditions. A knowledge of relational set theory and first order predicate logic and a facility with symbolic manipulation are all essential in interpreting the specification. In Z, declarations about sets and variables, together with the logic predicates which apply to them, can be grouped together in structures called schemas represented diagrammatically rather like a small table. These schemas often refer to the effect of a particular operation on some part of the overall state or system. Z therefore requires a considerable conceptual framework, and, for fluent manipulation and comprehension, a relatively high level of mathematical ability, (Finney, 1996).

We have observed that just the reading of specifications written in Z causes great difficulty to a wide range of software engineering students, and conjecture that the same problems will be found in the software development industry.

In this paper we extend the work of a pilot study reported elsewhere, (Finney and Fedorec, 1996). In order to quantify the difficulties that arise in reading specifications written in Z, an experiment was carried out with the aim of identifying the effect of three factors, (commenting, variable naming, and structure), on the comprehensibility of a small portion of a specification, (see Appendix D for sample specification and questions). The work is also an attempt to reflect similar studies carried out on the comprehensibility of programming languages (Tenny, 1988). It is hoped that the insights gained may improve our approach to the teaching of formal methods, and also form a basis for future research on a larger scale with members of the software engineering community.

2. The experiment

2.1. Design

The two primary factors considered important in affecting the comprehensibility of a formal specification in Z were: (i) the use of *meaningful identifying names* within the schemas, and (ii) the effect of *comments* in natural language between schemas. Thus the experimental design was a 2 × 2 factorial, involving four versions of the same specification, each having a different combination of these factors, as indicated in Fig. 1.

It was conjectured that version D (with meaningful variable names and additional natural language comments) would be the most comprehensible and A (without either) the least. All these specifications contain a certain degree of structuring although they are only small fragments of a larger specification. To enable an assessment of the effect of this structuring, a monolithic form of D was produced, E, by combining the original schemas into one. There are two possible effects of this:

1. the lack of structure will not affect the expected advantage of D, or,
2. the lack of structure will undermine the advantage expected in specification D.

The five specifications were assigned random numbers so that the subjects of the experiment could not deduce anything from the rank. (see Table 1).

The hypothesised ranking, in terms of comprehensibility, from easy to difficult, was **1 4 (5 2) 3.** It was conjectured that the lack of structure in such small specifications would be small, and the bracketing of 5(B) and 2(C) indicates our

		Meaningful Names	
		0	1
Comments	0	A	B
	1	C	D

Fig. 1. Specifications in terms of the two primary factors.

Table 1
Showing the coding of specification labelling

Specification	1(D)	2(C)	3(A)	4(E)	5(B)
Meaningful names	1	0	0	1	1
Comments	1	1	0	1	0
Structure	1	1	1	0	1

prior lack of view of the relative importance of comments and naming in isolation. The possibility of an interaction between comments and names was also of interest.

2.2. Response variables

The response variables were in three forms.

Times: The times taken to read the introductory document to the test, and the times taken to answer each of three questions.

Scores: The scores obtained on these three questions, where 1 was awarded for a correct answer and 0 for an incorrect one. The nature of the questions made it easy to decide on the marks and an answer sheet of acceptable answers was prepared before marking. All marking was done by one person so that there was consistency.

Rankings: Subjective rating of the comprehensibility of the specifications, by each of the subjects. A symmetric five point scale was used with −2 corresponding to 'unclear', and 2 corresponding to 'clear'.

2.3. Conduct of the experiment

2.3.1. Subjects

These were 147 undergraduate and postgraduate students at the University of Greenwich. They came from six different classes studying at levels from Higher National Certificate to Masters, and are, for the purposes of the experiment, labelled C1–C6. It was expected that there might be significant differences in the performances of these classes. However in this paper 'class' is largely treated as an experimental 'blocking' factor to improve the sensitivity of the main factors of the experiment.

2.3.2. Administration

The experiment took place in the normal course of the students' attendance at the university but the participation was voluntary. The students were assigned one of the five specifications randomly. They were given an instruction sheet and a separate question/answer sheet. The experiment consisted of two phases. In the first phase each subject studied one of the specifications, answered three questions on it and recorded their times to read and complete the questions. In the second phase each subject was issued with all five specifications and asked to rank them according to the ease with which they could read them.

2.4. Data 'cleaning'

All missing values have been treated as such and no missing value imputation has been used. Also responses such that the time spent on either reading the introductory details or answering any of the questions were zero have been treated as erroneous, probably due to rounding, and the associated times have also been treated as missing values. The data obtained from the experiment, and a summary can be found in Appendix A.

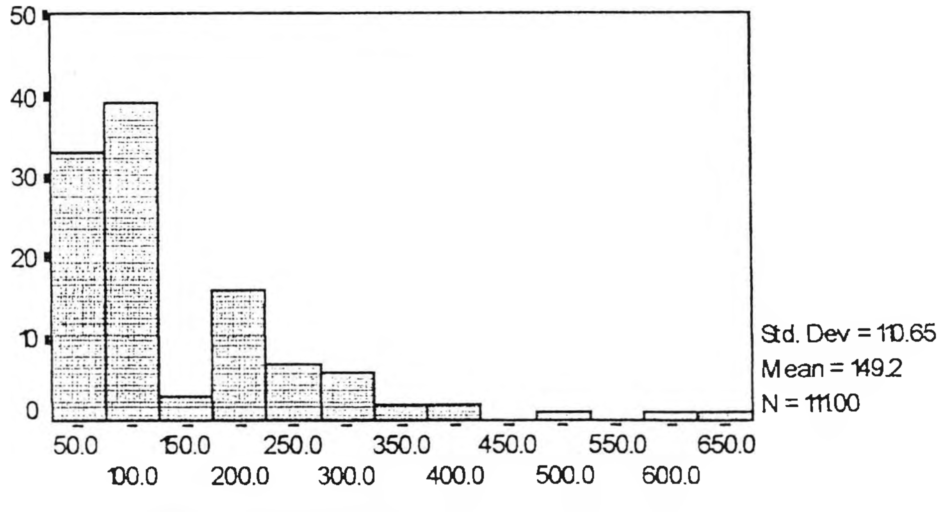


Fig. 2. Frequency distribution of times taken in seconds to answer Question 1.

3. Analysis

3.1. Analysis of timing data

The times taken (in seconds) to answer each question and the total time taken for all three questions are displayed in Figs. 2–5.

Note that the multi-modality in Figs. 2–4 is an artefact resulting from the tendency of subjects to round times to the nearest minute. This has been smoothed out in Fig. 5 for TOTQT. It would be possible, in principle, to take into account the 'rounding' tendency, in a model-based analysis of this timing data. However, such an analysis would be unnecessarily complicated in view of the relatively limited information contained in the timing data. The raw (cleaned) timings have therefore been used in the analyses reported in this paper. The correlation matrix between the timing variables was:

	TQ1	TQ2	TQ3	TREAD
TQ1	1.0000	0.1360	0.2700	0.0384
	(111)	(108)	(101)	(90)
P	-	0.161	0.006	0.719
TQ2	0.1360	1.0000	0.2371	0.0483
	(108)	(139)	(122)	(93)
P	0.161	-	0.009	0.646
TQ3	0.2700	0.2371	1.0000	0.1093
	(101)	(122)	(127)	(87)
P	0.006	0.009	-	0.313
TREAD	0.0384	0.0483	0.1093	1.0000
	(90)	(93)	(87)	(95)
P	0.719	0.646	0.313	-

(Coefficient/(Cases)/2-tailed Significance)

The times to complete Q1 and Q2 are significantly correlated with the time for Q3, but not with each other. None of the times to do the questions is significantly correlated with the time to read the introduction.

A Factor Analysis of the timing data was carried out using the correlation matrix, based on an initial Principal Components Analysis. Only one principal component had an eigenvalue greater than one, and hence only one factor was extracted. Hence, we conclude that there was no internal structure to the timings of the components of the test. We note that, the proportion of variance accounted for, (the commonalties), are largest for Q1 and Q3 and that the time to read the introductory material is not significantly related by the common time factor. Details are given in Appendix B.

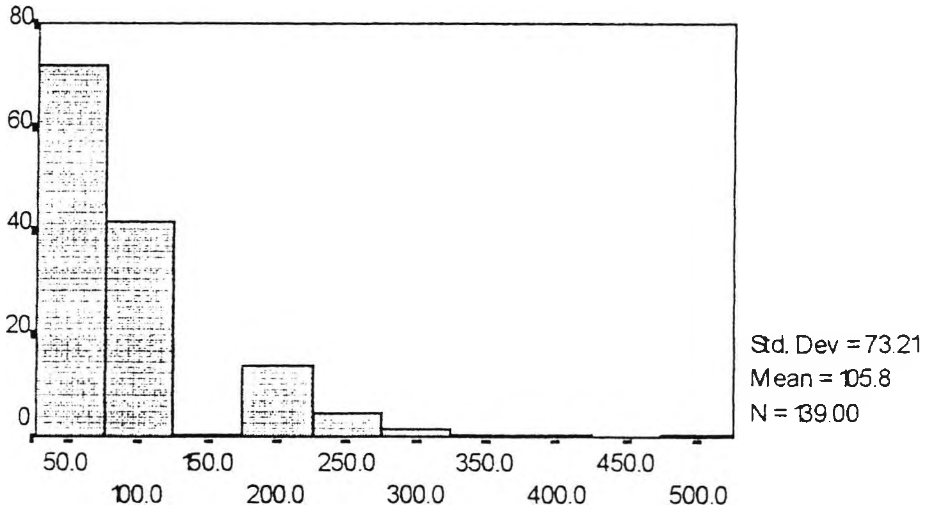


Fig. 3. Frequency distribution of times taken in seconds to answer Question 2.

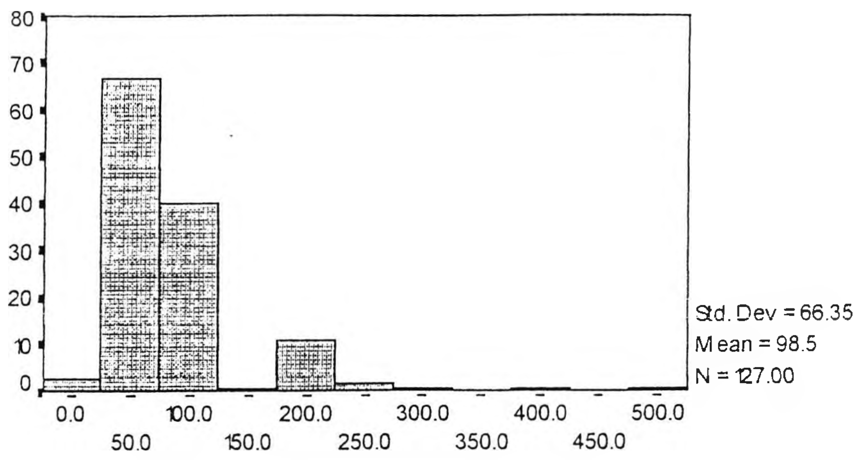


Fig. 4. Frequency distribution of times taken in seconds to answer Question 3.

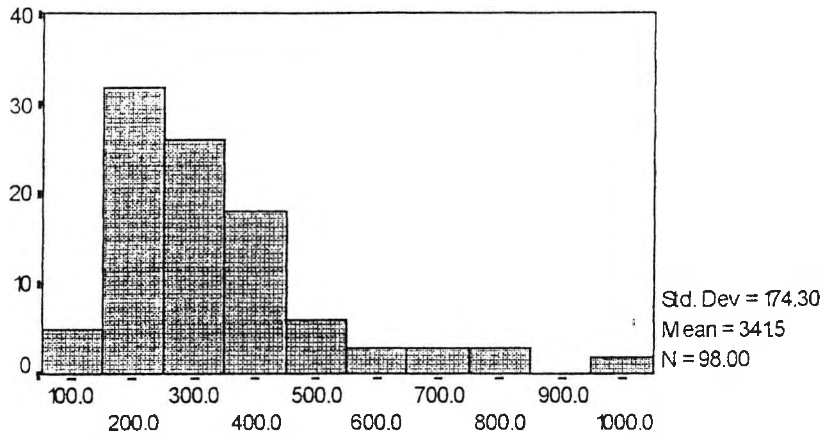


Fig. 5. Frequency distribution of time taken in seconds to answer all questions.

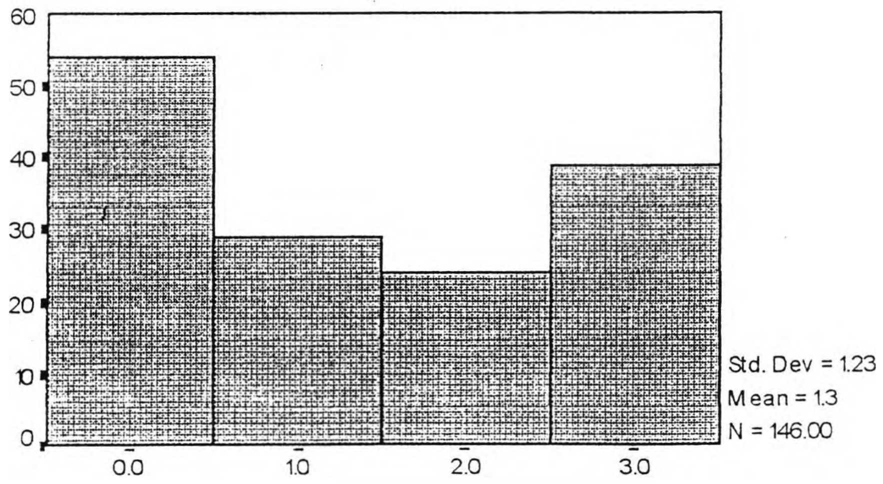


Fig. 6. Frequency diagram of total score.

3.2. Analysis of scores data

Clearly, the variable most likely to give a good measure of comprehension is the total score. Fig. 6 shows the distribution of total score, which is bimodial.

The modes, 0 (the 'failures') and 3 (the 'successes'), are analysed in detail later. With total score as response variable an initial Analysis of Variance (ANOVA) was performed with two factors: (i) NLVL, level of student class, and (ii) SPEC, the specification, with the total time taken was used as a covariate. The ANOVA table is given in Table 2.

Table 2
Analysis of variance of total scores [a]

Source of variation	Sum of squares	DF	Mean Square	*F*	Sig. of *F*
Main effects	25.016	9	2.780	2.235	0.030
NLVL	10.355	5	2.071	1.665	0.155
SPEC	14.661	4	3.665	2.947	0.026
Covariates	17.283	1	17.283	13.895	0.000
TOTQT	17.283	1	17.283	13.895	0.000
2-Way interactions	18.020	18	1.001	0.805	0.688
NLVL, SPEC	18.020	18	1.001	0.805	0.688
Explained	60.320	28	2.154	1.732	0.035
Residual	83.337	67	1.244		
Total	143.656	95	1.512		

147 cases were processed.
51 cases (34.7 pct) were missing.
[a] Anova of TOTS by NLVL and SPEC, with TOTQT. HIERARCHICAL sums of squares; Covariates entered AFTER main effects.

Table 3
Mean total scores, (number of cases), by specification

SPEC	1	2	3	4	5
	1.71	0.84	1.06	1.74	1.32
	(21)	(19)	(18)	(19)	(19)

It may be seen that the form of specification, SPEC, is significant with a *p*-value of 0.026, but that the effect of class level, NLVL, and the interaction, NLVL*SPEC, have not been found to be significant. Accordingly, NLVL has been omitted from the next ANOVA. The mean scores, for the different specifications are given in Table 3.

We see that the ranking of the specifications, by decreasing mean TOTS, is

4(E)	Meaningful names, comments but no structure
1(D)	Meaningful names, comments and structure
5(B)	Meaningful names, no comments but structure
3(A)	No meaningful names, no comments but structure
2(C)	No meaningful names, but comments and structure

which may be compared with the conjectured 1(D) 4(E) (5(B) 2(C)) 3(A). It is seen that 1(D) and 4(E), distinguished by the presence or absence of structure, are not significantly different in terms of mean response, with means 1.71 and 1.74, respectively. Also, the mean scores for 3(A) and 2(C), (1.06 and 0.84), without meaningful names are substantially less than that for 5(B), with meaningful names, (1.32).

The experiment had been designed with a factorial 2 × 2 + (1) 'treatment-structure', and the ANOVA for this design is given below:

Tests of Significance for TOTS using Cov Adj SEQUENTIAL Sums of Squares

Source of Variation	SS	DF	MS	*F*	Sig. of *F*
WITHIN + RESIDUAL	119.70	91	1.32		
REGRESSION	12.90	1	12.90	9.81	0.002
CMT	2.53	1	2.53	1.92	0.169
MNFUL	5.80	1	5.80	4.41	0.039
STRCT	0.24	1	0.24	0.19	0.667
CMT * MNFUL	0.76	1	0.76	0.58	0.449
(Model)	25.74	5	5.15	3.91	0.003
(Total)	145.44	96	1.52		
R-Squared =	0.177				
Adjusted R-Squared =	0.132				

```
Regression analysis for WITHIN + RESIDUAL error term--Individual Univariate 0.9500 con-
fidence intervals. Dependent variable TOTS
```

COVARIATE	B	Beta	Std. Err.	t-Value	Sig. of t
TOTQT	−0.00214	−0.30393	0.001	−3.132	0.002
COVARIATE	Lower -95%	CL- Upper			
TOTQT	−0.003	−0.001			

The only significant factor is whether MNFUL, i.e. the indicator of whether meaningful names are given to variables. Commenting, CMT, and Structure, STRCT, with a *p*-values of 0.169 and 0.667, are not significant, and there is not a significant interaction between naming and commenting. We note also that the covariate, TOTQT, the total time to answer the questions has been found to be very significant with an estimated regression coefficient for TOTQT of −0.00214 (se = 0.001), establishing an inverse relationship between total score and total time to answer the questions. Unsurprisingly, the better students take less time.

3.3. Success and failure analysis

3.3.1. Individual question analysis

Multiple Logistic Regression Analysis was done on the responses to each of the three questions. In all three analysis the use of meaningful variable names was the only significant factor, with a significant negative coefficient for time taken for both Questions 2 and 3. The results are therefore confirmatory of the results obtained above.

3.3.2. Factors affecting success, (score = 3/3), and failure, (score = 0/3)

Multiple Logistic Regression Analysis was done separately on the outcomes: ‘Success’, (score = 3/3), and ‘Failure’, (score = 0/3). The results of the analysis of ‘Success’ are shown below:

```
Logistic analysis of the probability of getting 3/3
   Number of selected cases = 147.
   Number rejected because of missing data = 50.
   Number of cases included in the analysis = 97.

Dependent variable: ITHREE

Initial -2 Log likelihood = 110.71022.

Fitted model
   -2 Log likelihood = 79.978
   Goodness of fit = 78.204.

                                                Chi-Square           DF    Sig.
   Model Chi-Square                             30.733               9     0.0003

   Classification table for ITHREE

                        Predicted
                        0          1                Percent correct
                        0          1
   Observed
   0               0    67         5                93.06%
   1               1    13         12               48.00%
                                   Overall          81.44%
```

Variables in the equation

Variable	B	Std. Err.	Wald	DF	Sig.
C1	−3.3108	1.4739	5.0462	1	0.0247
C2	−2.2444	1.3642	2.7070	1	0.0999
C6	−3.7509	1.2891	8.4669	1	0.0036
C4	−1.5814	1.2642	1.5649	1	0.2109
C5	−0.4114	1.3227	0.0967	1	0.7558
CMT	1.7208	0.7630	5.0872	1	*0.0241*
MNFUL	0.6734	0.7109	0.8973	1	*0.3435*
STRCT	−0.6645	0.7724	0.7402	1	*0.3896*
TOTQT	−0.0111	0.0036	9.5154	1	*0.0020*
Constant	3.7577	2.0728	3.2864	1	0.0699

The fitted model is, prob(Success) $= 1/1 + \exp(-\eta)$, where $\eta = \sum_{i=0}^{9} B_i V_i$. B_i is the estimated coefficient of V_i, (B_0 = constant, and $V_0 = 1$). This model has an 81% correct classification rate on the data.

We note that the meaningfulness of variable names, MNFUL, is *NOT* a significant beneficial factor, but commenting, CMT, *IS*, with a p-value of 0.0241. More able subjects are able to make use of the information in comments, and hence obtain high scores, whilst for such subjects the use of meaningful names is of minor relative importance, on such small specifications.

We also note that the total time to answer all questions, TOTQT, is a significant inverse indicator of good performance, and that there is a more noticeable variation between the groups when considering this response variable.

In contrast, the logistic analysis of the probability of obtaining a 0/3 scores unsurprisingly shows that none of the experimental factors are significant, but that there is a significant positive dependency of the probability of a 0/3 score on the total time to answer the questions, ($p = 0.034$).

We conclude that none of the factors can really be used to help the weakest group who will take a long time over the test and still score zero. For the better group who answer all questions correctly they seem to do this quickly and get help only from the added comments.

3.4. *Analysis of perceived-comprehensibility rankings*

In the second part of the experiment, students ranked all five specifications, from least comprehensible (−2) to most comprehensible (2). Analysis is primarily by graphical means. The data may be found in Appendix C.

Examination of the data/graph for all the subjects, Fig. 7, confirms that Spec.1, (with comments and meaningful names), is found most comprehensible by most candidates, and is found to be least comprehensible by the least candidates, as conjectured. The converse result is found for Spec. 3, (no comments and names without meaning), also as conjectured. The ratings of Spec.'s 2 and 5 are arbitrarily dispersed, with neither getting high or low proportions of the extreme ratings. Spec. 4 seems to be perceived similarly to 2 and 5 on the whole. Hence it would seem that the monolithic nature of the code in Spec. 4 has detracts from the perceived clarity of Spec.1, in spite of the fact that Structure has not been shown to be a significant factor in the analysis of scores.

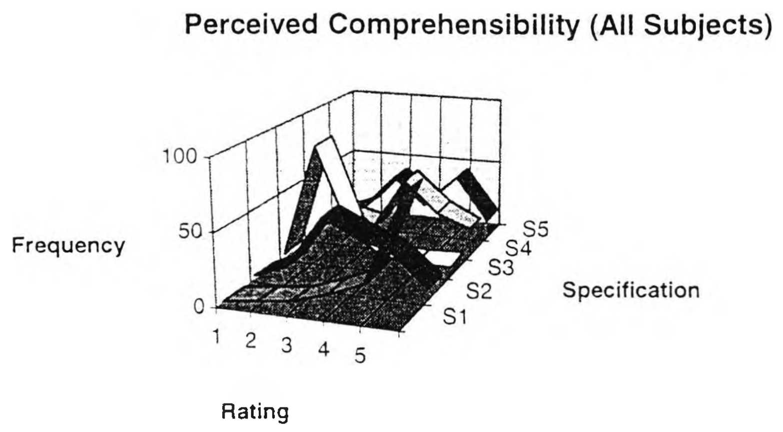


Fig. 7. Frequency plot of perceived comprehensibility, by specification.

Kendal's W measure of concordance in rankings (Kendal, 1948), of the specifications has the value of 0.147, with a p-value of 0.193 for the null hypothesis that $W = 0.0$. The clear trends for specifications 1 and 3 have been obscured by the lack of concurrence concerning the other specifications, when all subjects are considered together.

We may ask if the perceptions of the comprehensibility of the alternative specifications depends upon the ability of the subjects, best measured by their total score on the questions, TOTS. For those with TOTS = 3, (full marks), the conclusions do not change, and Kendal's W measure of concordance is higher at 0.6047 with a p-value of 0.11, slightly closer to significance as we would expect from the graphical display, but still reduced by the effects of Specs. 2, 5 and 4. For those with less than full score, the lack of clarity of Spec. 3 is unanimous. Also, whilst relatively few of these weaker subjects rates Spec. 1 at the low end of the scale, rather surprisingly, few of them see Spec. 1 as being superior to Spec.'s 2, 5 and 4. Perhaps the lack of perception of the advantages of comments and meaningful names (and structure), as present in Spec.1 indicates that the weaker subjects do not have the intellectual skills to make use of these meaningful cues. Reasons for this will undoubtedly vary, from lack of basic intellectual ability to insufficient training.

4. Conclusions

This paper has reported on the results of a study to evaluate the impact of the style factors, of naming, commenting and structure on the comprehensibility of a small specification. The specification environment chosen has been Z, though it should be emphasised that no conclusions are possible in this study about the comprehensibility of Z compared to other specification notations. However we would expect that our conclusions about the influence of these factors on comprehensibility might be applicable to other specification environments.

We have found that those obtaining higher scores take less time than those who obtain low scores. Total time taken is therefore a correlational surrogate of ability, and has been used as an adjusting covariate in the analysis of how the style factors affect the comprehension scores. Meaningful naming is the only style variable found to be significant in ANOVA, but commenting is found to be the only style factor predictive of the best score 3/3 with all three questions correct. For the small specification of this study the contrast between monolithic and structured schemas does not significantly influence the scores obtained. However, the monolithic schema does seem to reduce the *perceived* comprehensibility. These conclusions are specific to the specification used, and to its Z environment. However, since all of the style factors are significant in one guise or another, the study confirms the conventional wisdom of good programming practice, that style is very important to comprehensibility.

There is an extensive literature on metrics concerned with the production and maintenance of software e.g. (Inglis, 1985; Fenton, 1994). It would be desirable to have available indices of comprehensibility which are able to indicate the extent to which good style has been followed at the specification stage. High comprehensibility scores might be expected to incur increased costs in time spent and training given at the initial part of the lifecycle but would also be expected to provide benefits at de-bugging, maintenance and modification stages.

A comprehensibility index or metric would have to involve the factors of naming, commenting and structure, as well as others, but it is not clear, a priori, what the relative weights should be for these factors in the overall index. We suggest that such weights are best determined as a result of measurement of performance evaluations on the target group of users. The final resulting framework would be a quantified set of measures that could be used at the software specification to aid in the optimisation of the software quality.

While performance based determination of a comprehensibility index is recommended, other factors other than objective performance should also be taken into account. A feeling of ease in practitioners is clearly important, since it will be likely to enable sustained activity on the part of the practitioner. As a specification is also part of the overall chain of design and development of the software, the ease with which this early stage is accomplished and understood will have repercussions for the later stages. Determination of the way that style influences the perceived comprehensibility and maintainability is therefore important, and might be obtained at the same time that performance measures are determined. The combined measurement of timings, performance scores, and attitude measurement as used in this study might therefore be taken as a paradigm by data may be collected and on which the development of a comprehensibility index may be based. The relative importance of performance-based, and attitude-based measures of comprehensibility may well be based on prior experience, but could be established using a number of extended case-studies.

It is clear that the best way forward is the use of well designed statistical experiments to ensure results are obtained as efficiently as possible with the rigour to give supporting evidence for any conclusions drawn. An increased use of experimental design in the conduct of empirical research in the area of formal specification will put some objective evidence into the area to challenge and support those arguments which at present may rely too much on subjective opinions. By the same token, use of the techniques of modern statistical analysis are required to look at how the evidence collected can be interpreted to give valid conclusions.

As the development of the work described here further pilot studies need to take place on other aspects of comprehension. In particular a study is needed of a specification large enough to investigate further the effect of structure as a factor. Larger studies across the student populations of several institutions would ensure that the variation due to factors such as background, learning experience and environment could be taken into account.

However the development of metrics which could apply to formal methods needs to be based on research which is commercially based and on a realistically large scale. To date few such studies of the benefits of using formal methods within an industrial setting have been conducted and most have been inconclusive.

Those involved with software engineering stress the importance of the delivery of software which meets its specification, is easy to maintain and can be modified, all within budget constraints. Development of a set of metrics to be used in relation to formal methods, which would be able to demand or produce a high level of quality over a range of attributes including comprehensibility could be one way forward.

Acknowledgements

The authors would like to thank Professor Norman Fenton and the anonymous referees for their encouragement and suggestions. All of the statistical analyses reported in this paper were conducted with SPSS for Windows. Refer to (Norusis, 1993) for documentation.

Appendix A

The cleaned data set consisted of:

NLVL	Course level (C1–C6)
Student id number	From 1–147
SPEC	One of the five specifications 1–5
CMT(Comment)	These three
MNFUL(Meaningful_names)	Were dummy variables
STRCT(Structured)	Indicating the type of specification given
TREAD	Time to read to intro (in seconds)
TOTQ1,TOTQ2,TOTQ1	Times to answer Questions 1–3 respectively
TOTQT	Total time to answer the questions rankings of the five specifications from least comprehensible to most; coded as −2 −1 0 1 2
S1,S2,S3	Scores on the three questions; either 0 or 1
TOTS	Total score out of 3

Summary of variables

Variable	Mean	Std. dev.	Minimum	Maximum	*N*
Scores for questions 1--3, and total-score					
S1	0.46	0.50	0.00	1.00	146
S2	0.55	0.50	0.00	1.00	146
S3	0.32	0.47	0.00	1.00	146
TOTS	1.33	1.23	0	3	146
Indicator variables for subjects with zero and three total-score					
IZERO	0.37	0.48	0	1	146
ITHREE	0.27	0.44	0	1	146
Times to read the introduction, and the three questions					
TREAD	168.24	129.29	30.00	780.00	95
TQ1	149.23	110.65	27.00	660.00	111
TQ2	105.79	73.21	25.00	480.00	139
TQ3	98.47	66.35	17.00	480.00	127
TOTQT	341.52	174.30	120.00	960.00	98

Appendix B

Details of the Factor Analysis of the timing data. The matrix of factor loadings are:

Factor 1

TQ1	0.67773
TQ2	0.45367
TQ3	0.80060
TREAD	0.36831

Note that all the loadings are positive, with the times for Questions 1 and 3 having the highest loadings on the Common factor. The loadings would have to be divided by the response standard deviations to obtain the relevant correlations.

Final statistics

Variable	Communality	Factor	Eigenvalue	Pct of var	Cum Pct
TQ1	0.45932	1	1.44174	36.0	36.0
TQ2	0.20581				
TQ3	0.64096				
TREAD	0.13565				

Appendix C. Perceived comprehensibility rating data

Frequency tables for the comprehensibility of specifications, vertically ordered by specs. 1–5; and horizontally by comprehensibility coded as:

Least comprehensible...				most
-2	-1	0	1	2
	2	3	4	5

The first three tables are for each of the total scores 3,2,1. The final table is for all subjects.

Tots = 3 (N = 39)				
1	1	1	7	29
2	17	14	3	2
30	5	1	2	1
4	2	18	14	2
2	14	5	13	5
Tots = 2 (N = 24)				
1	0	3	4	3
1	9	9	5	9
19	2	0	1	0
1	5	8	5	8
2	8	4	9	4
Tots = 1 (N = 28)				
0	1	3	6	3
6	8	5	6	5
12	10	4	1	4
7	3	11	2	11
3	6	5	13	5

Tots=0 (N=47)				
2	5	8	8	8
3	15	10	15	10
25	9	5	3	5
9	5	16	10	16
8	13	8	11	8
All Subjects)N=138)				
4	7	15	25	87
12	49	38	29	8
86	26	10	7	8
21	15	53	31	21
15	41	22	46	14

Appendix D. Specification 1

The new users name is taken in and an identity number is assigned from the pool of unused numbers. The unused number set is amended and the new pair of user and their number are added to the existing users.

Add

$\Delta System$
$name? : Person$
$n? : \mathbb{N}$
$message! : Response$

$n? \in Unused\_Ids$
$name? \notin \mathrm{dom}\ Users$
$Unused\_Ids' = Unused\_Ids \setminus \{ n? \}$
$Users' = Users \cup \{ name? \mapsto n? \}$
$message! = OK$

Here error messages are generated by the failure of either of the two preconditions in the Add schema.

AddFail

$\Xi System$
$name? : Person$
$n? : \mathbb{N}$
$e\_message! : Response$

$(name? \in \mathrm{dom}\ Users \wedge e\_message! = name\_in\_use) \vee$
$(Unused\_Ids = \varnothing \wedge e\_message! = no\_id\_available)$

Finally the behaviours are combined

$AddUser = Add \vee AddFail$

D.1. Questions

1. What conditions give rise to error messages?
2. The size of which set would give you the number of current users on the network?
3. Which set or sets give you information about the total number of users the network will support?

References

Bowen, J.P., Hinchey, M.G., 1995. Ten commandments of formal methods. IEEE Computer 28 (4), 56–63.
Cohen, B., 1989. A rejustification of formal notations. Software Engineering Journal 4 (1), 36–38.
Craigen, D., Gerhart, S., Ralston, T., 1995. In: Hinchey, M., Bowen, J. (Eds.), Applications of Formal Methods. Prentice-Hall, Englewoodcliffs, NJ.
Fenton, N.E., 1994. Software measurement: A necessary scientific basis. IEEE Trans. on Software Engineering 20 (3), 199–206.
Fenton, N., Pfleeger, S.L., Glass, R.L., 1994. Science and substance: A challenge to software engineers. IEEE Software 11 (4), 86–95.
Finney, K., 1996. Mathematical notation in formal specification: Too difficult for the masses? IEEE Trans. on Software Engineering 22 (2), 158–159.
Finney, K., Fedorec, A., 1996. In: Dean, N., Hinchey, M. (Eds.), An Empirical Study of Specification Readability Teaching and Learning Formal Methods. Academic Press, New York.
Hall, J.A., 1990. Seven myths of formal methods. IEEE Software 7 (5), 11–19.
Inglis, J., 1985. Standard software quality metrics. AT&T Tech. J. 64 (1), 113–118.
Kendal, M.G.: Rank Correlation Methods. Charles Griffin, London.
Kitchenham, B., Pickhard, L., Pfleeger, S.L., 1995. Case studies for method and tool evaluation. IEEE Software 12 (4), 52–62.
Norusis, M., 1993. SPSS, Advanced Statistics, Release 5.0, SPSS.
Saiedian, H., 1996. An invitation to Formal Methods. Computer 29 (4), 16–30.
Spivey, J.M., 1992. The Z Notation: A Reference Manual, 2nd ed., Prentice-Hall, Englewoodcliffs, NJ.
Tenny, T., 1988. Program readability: Procedures versus comments. IEEE Trans. on Software Engineering 14 (9), 1271–1279.
Woodcock, J.C.P., 1989. Structuring specifications in Z. Software Engineering Journal 4 (1), 51–66.

Kate Finney is Senior Lecturer in Mathematics at the University of Greenwich. Her research interests are the development of metrics applicable to formal specification and the issues surrounding the accessibility of mathematical notation.

Keith Rennolls is Professor of Applied Statistics and Head of Statistics at the University. His applications interests are wide, but those relevant to this paper include Psychometrics, Learning Theory, and Measurement Theory.

Alex Fedorec is Senior Lecturer in Systems and Software Engineering at the University of Greenwich. He started as a DOS-VSE PL/1 programmer in the late 1970s and has worked on applied research in parallel and image processing.