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

Norman Fenton
Centre for Software Reliability
City University
Northampton Square
London EC1V OHB

1 June 1996

Abstract

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 publicised claims are: less than half the normal number of customer-reported errors; and a 9% saving 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 we feel that their widespread commercial use is ultimately dependent on 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.

1. Introduction

In the last ten 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. We feel that, 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 summarised 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.
- 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 recognised 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 saving 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 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 (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 the 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 analyse 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. We believe that 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. We believe that 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 1994] 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 world-wide. 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

    (2000 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 re-specified 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.

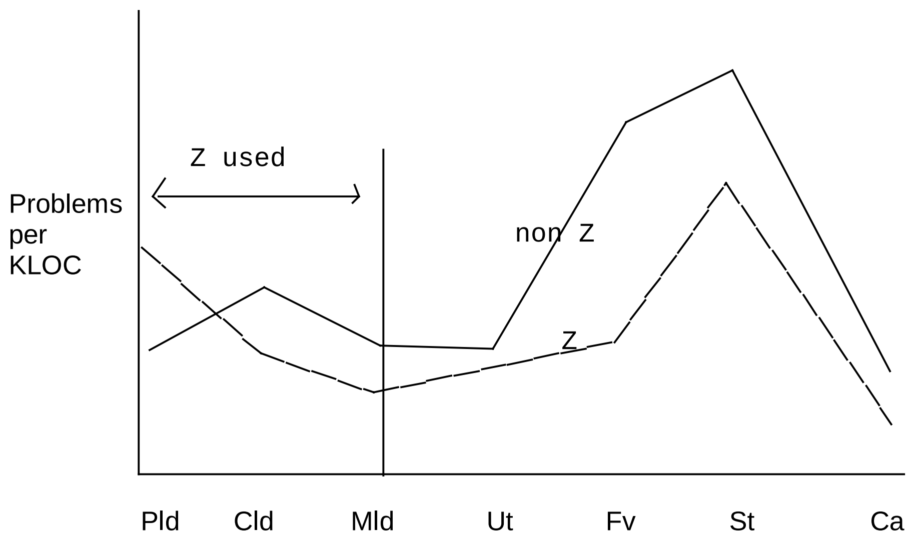
3. The quantitative claims made

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 development life-cycle. [Phillips 1989] draws the graph up to the system test phase, but [Houston and King 1991] adds 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 our own findings when we analysed 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.



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

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 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 pertion, 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 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 1 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 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 8 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 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.

4. Comparing the case study with best practice

IBM do 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, Pickard and Pfleeger in the study of experimental validation of software tools and techniques. In [Kitchenham et al 1995] they 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 or not 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

But analysis of these issues is vague. 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 but the intentions were clearly stated. They intended:

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

To measure in the specification and design phase

- Time spent on producing documentation
- Size of documentation
- Resources spent on inspections
- Number of problems found & their severity

To measure at the coding stage

- Time spent writing & testing code
- Number of lines of code produced
- Problems at each stage (unit test, function test etc)
- Problems after shipping
- Categorisation of problems by type (specification,design...)
- Categorisation 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 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

but no results are given 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 8 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 which worked on Z were (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 practise no rigor was applied to the interface between the different levels of abstraction and in particular the links between Z and Dikjstra's language of guarded commands do not appear to have been formalised.

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 problems 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 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

Minimise external factors

Minimise confounding factors

Plan and collection 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.

6. Acknowledgements

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.

7. References

Bowen JP and Stavridou V, Safety critical systems, formal methods and standards, Software Eng J, 8(4), 189-209, 1993.

Bowen JP and Hinchley MG, Seven more myths of formal methods, IEEE Software, 12(3), 34-41, July, 1995.

Collins. B.P, Nicholls. JE, Sorensen.IH, Introducing Formal Methods: the CICS Experience with Z, *Mathematical Structures for Software Engineering*, Clarendon Press Oxford 1991 pp153 -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* Aug 1975, Vol 18 No8 pp453-457

Fenton NE and Hill G, Systems Construction and Analysis: A Mathematical and Logical Approach, McGraw Hill, 1992.

Fenton NE and Mole D, A note on the use of Z for flowgraph decomposition, J Information & Software Tech,Vol 30 (7), 432-437, 1988.

Fenton NE, Pfleeger SL, 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 and 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, 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

Kitchenham. B, Pickhard. L, Pfleeger. S. L, Case studies for Method and Tool Evaluation *IEEE Software* July 1995 pp 52-62

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 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.

Phillips. M,CICS/ESA 3.1 Experiences, *Proceedings of the 4th Annual Z User Meeting*. Springer Verlag Workshops in Computing, pp 179-185. 1989