

# An Analysis of Two Formal Methods: VDM and Z

13 August 1997

Authored by:  
Thomas McGibbon  
ITT Industries - Systems Division



DoD Data & Analysis Center for Software

# An Analysis of Two Formal Methods: VDM and Z

Contract Number F30602-89-C-0082  
(Data & Analysis Center for Software)

**Prepared for:**  
**Air Force Research Laboratory -**  
**Information Directorate (AFRL/IF)**  
**525 Brooks Road**  
**Rome, NY 13441-4505**

**Prepared by:**  
**Thomas McGibbon**  
**DoD Data & Analysis Center for Software (DACS)**  
**ITT Industries - Systems Division**  
**Griffiss Business & Technology Park**  
**775 Daedalian Drive**  
**Rome, NY 13441-4909**



**DoD Data & Analysis Center for Software (DACS)**  
**P.O. Box 1400**  
**Rome, NY 13442-1400**  
**(315) 334-4905, (315) 334-4964 - Fax**  
**[cust-lain@dacs.dtic.mil](mailto:cust-lain@dacs.dtic.mil)**  
**<http://www.dacs.dtic.mil>**

The Data & Analysis Center for Software (DACS) is a Department of Defense (DoD) Information Analysis Center (IAC), administratively managed by the Defense Technical Information Center (DTIC) under the DoD IAC Program. The DACS is technically managed by Air Force Research Laboratory Information Directorate (AFRL/IF) Rome Research Site. ITT Industries - Systems Division manages and operates the DACS, serving as a source for current, readily available data and information concerning software engineering and software technology.

# REPORT DOCUMENTATION PAGE

*Form Approved*  
*OMB No. 0704-0188*

Public reporting burden for this collection is estimated to average 1 hour per response including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project, (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY ( <i>Leave Blank</i> )	2. REPORT DATE 20 August 1997	3. REPORT TYPE AND DATES COVERED N/A	
4. TITLE AND SUBTITLE An Analysis of Two Formal Methods: VDM and Z		5. FUNDING NUMBERS F30602-89-C-0082	
6. AUTHORS Thomas McGibbon - DACS Director			
7. PERFORMING ORGANIZATIONS NAME(S) AND ADDRESS(ES) ITT Industries, Systems Division, 775 Daedalian Drive Rome, NY 13441-4909		8. PERFORMING ORGANIZATION REPORT NUMBER DACs-CRTA-97-1	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Technical Information Center (DTIC)/ AI 8725 John J. Kingman Rd., STE 0944, Ft. Belvoir, VA 22060 and Air Force Research Lab/IFTD 525 Brooks Rd., Rome, NY 13440		10. SPONSORING/MONITORING AGENCY REPORT NUMBER  N/A	
11. SUPPLEMENTARY NOTES Available from: DoD Data & Analysis Center for Software (DACs) 775 Daedalian Drive, Rome, NY 13441-4909			
12a. DISTRIBUTION/ AVAILABILITY STATEMENT Approved for public release, distribution unlimited		12b. DISTRIBUTION CODE UL	
13. ABSTRACT ( <i>Maximum 200 words</i> )  This paper compares and contrasts the strengths and weaknesses of the Vienna Development Method (VDM) and Z in the software design life cycle phase, and compares and contrasts VDM and Z to other formal models. Tool support, lessons learned, and technical and achieved business benefits are emphasized. Based on available data, this paper analyzes the return-on-investment (ROI) from use of these methods, and this ROI data is compared to ROI data from cleanroom software engineering and other process improvement methods.			
14. SUBJECT TERMS Formal Methods, VDM, Z, Zed, Metrics Software Measurement		15. NUMBER OF PAGES 21	
		16. PRICE CODE N/A	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

# An Analysis of Two Formal Methods: VDM & Z

## Table of Contents

Abstract & Ordering Information	1
1. Introduction	2
2. Traditional Uses of Formal Methods	2
3. Use of Formal Methods in Design	3
4. Comparison of Method Characteristics	5
5. Comparison of Tool Support	7
6. Comparison of Field Experience	8
a. Table 3 Conclusions	13
7. Detailed Financial Analysis	14
References	16
About the Author	17
<b>Appendix</b>	
Annotated Bibliography	A1

### Tables

<u>Table 1</u> : Comparison of VDM and Z Characteristics	6
<u>Table 2</u> : Tool Support Comparison	8
<u>Table 3</u> : Practical Experience with VDM and Z	10-12
<u>Table 4</u> : Financial Comparison of Formal Methods	14

### Figures

<u>Figure 1</u> : Defect Rates for CICS Project	13
---	----

# An Analysis of Two Formal Methods: VDM & Z

## Abstract and Ordering Information

### Abstract

This paper compares and contrasts the strengths and weaknesses of the Vienna Development Method (VDM) and Z in the software design life cycle phase, and compares and contrasts VDM and Z to other formal models. Tool support, lessons learned, and technical and achieved business benefits are emphasized. Based on available data, this paper analyzes the return-on-investment (ROI) from use of these methods, and this ROI data is compared to ROI data from cleanroom software engineering and other process improvement methods.

### Ordering Information:

A bound version of this report, is available for \$30 from the DACS Product Orderform at: <http://www.dacs.dtic.mil/forms/orderform.shtml> or you may order it by contacting:

DACS Customer Liaison  
775 Dadaelian Drive  
Griffiss Business Park  
Rome, NY 13441-4909

(315) 334-4905; Fax: (315) 334-4964;

(800) 214-7921- Toll Free;

[cust-liasn@dacs.dtic.mil](mailto:cust-liasn@dacs.dtic.mil)

---

### Acknowledgements:

The author would like to gratefully acknowledge comments on an earlier draft by Mr. Robert Vienneau and the help of Mr. Lon R. Dean in producing this report.

## Introduction

The purpose of this paper is to compare and contrast two of the most frequently used formal methods in design: the Vienna Development Method (VDM) and Z. Hayes [14] has provided a fairly thorough comparison of the appearances, data types, states, initialization, operations, syntax, preconditions, and exception handling of the two methods. These comparisons are summarized. Tool support for each method is compared and contrasted, and the results of applying these two methods in practice is presented. Formal methods are generally believed to reduce development costs and reduce defects in operational software. Practical experience with these methods is examined to see if the facts support this hypothesis.

Formal methods, in the sense of software development, are mathematically based techniques used to precisely describe a system. They can be applied throughout the development of a system and involve the use of refinement techniques and proofs of correctness at each stage to insure that the current specification completely and correctly refines the previous specification. Formal methods can also be used to ensure unsafe or insecure states cannot arise in any system satisfying a formal specification.

Formal methods grew out of program proving techniques, early examples of which are Edsger Dijkstra's predicate transformers and Harlan Mills' function approach. The syntax of predicates and function specifications drew on a mixture of mathematical logic and set theory, but this mathematical background was initially not fully formalized. Gradually formal methods have come to emphasize formal specifications. The development of formal methods has been accompanied by the development of logic customized for system development. For example, temporal logic includes operators to indicate that a proposition is always true, is true until some other proposition becomes true, or will be eventually true, in addition to the usual operators in the propositional calculus of negation, disjunction, conjunction, implication, and equivalence. Larch, Communicating Sequential Processes (CSP), Petri nets, and State Charts are all examples of formal methods.

VDM and Z seem to be the two most popular and most frequently used formal methods for specifications. They are the most frequently referenced methods in the literature, have the most world wide web pages addressing their use, and are among the few methods for which ISO standards exist. The focus of this paper is on comparing and contrasting these two methods.

How, where and why formal methods have been applied is examined in the remainder of this paper. VDM and Z are compared and contrasted based on their syntax and semantics and based on real field use of the two methods. The last section examines the return on investment associated with the use of these methods. Comparisons are shown in tabular fashion with necessary supporting text.

## Traditional Uses of Formal Methods

Formal methods are traditionally used when a system has correctness as a concern, such as in safety-critical and security-critical systems, systems in which the cost of system failure is catastrophic, and systems where standards organizations mandate their use ([8], [9], [11], [12], [15], [26]). As stated by Gerhart, Craigen and Ralsten [8], formal methods are primarily used for:

- Quality assurance for systems that require a high degree of confidence, auditable information, and targets of low or zero error rates
- Developing a better understanding of an application domain and communicating that understanding

- Providing evidence of best practice
- Systems that require structure recovery or functional enhancement.

Formal methods are most frequently used to capture and define system and functional requirements. Formal methods have been used for safety and security-critical purposes in:

- Certifying the Darlington Nuclear Generating Station plant shutdown system [9]
- Designing the software to reduce train separation in the Paris Metro [9]
- Developing a collision avoidance system for United States airspace [9]
- Achieving clearance to carry sensitive information through an internet gateway [9]
- Assuring safety in the development of programmable logic controllers [11]
- Developing a water level monitoring system [27]
- Developing an air traffic control system [12].

Formal methods have been used as an aid in the specification of an oscilloscope [3], in the development of IBM's Customer Information Control System transaction processing system [16], and in development of IBM's Cobol Structuring Facility [20]. Formal methods have also been used for design verification [17] of a RISC processor [25] and in standards development [1].

Experience with formal methods in the United States includes Cleanroom Software Engineering; research at the Naval Research Laboratory, such as the Software Cost Reduction project; various NASA projects; and academic research. Most of the formal methods experience documented in the open literature, however, has largely taken place in European countries, especially the United Kingdom. For that matter, there are no technical journals within the United States devoted to Formal Methods. The main English-language journals are in the United Kingdom.

## Use of Formal Methods in Design

Decomposition and refinement are two important elements of the design of software systems. Decomposition involves the process of dividing a system into smaller elements or modules. Refinement involves specification at different levels of abstraction, and showing that a specification at a lower level satisfies a requirement at a higher level.

Formal methods, such as Z and VDM, are often used primarily for requirements specification, not design. Traditional formal specification languages, such as Z, VDM, Larch, and Lamport's transition axiom method are methods, however, that are well suited to the software design process [27]. Traditional formal specification languages can be, and have been, used to specify module interfaces. Implementers of modules can implement a module without knowledge of the specifics of called modules. In addition, implementers of a called module can implement its specification without knowledge of the calling module, as long as the interface remains the same. Traditional formal methods are used to prove satisfaction of requirements arising in the refinement of a design. Formal methods are also used for procedural specifications of system functionality

However formal methods cannot be used for all aspects of system design (e.g. user interface design) and most projects that employ formal methods utilize a combination of traditional and formal methods to complete the design. Luqi and Goguen [21] state that formal methods do not yet handle large and complex system development. Practitioners of formal methods consider formal methods appropriate for

proving that programs satisfy certain mathematical properties of a system. The gaps between formal specifications and code are still great. Luqi and Goguen conclude that formal methods should play a part in reliability. A British Air Traffic Control system named the Central Control Function (CCF) Display Information System (CDIS), a security-critical system, and a development of Programmable Logic Chips (PLCs) illustrate the impacts of formal methods on design.

As described in Hall [12], Fenton and Pfleeger [4], and Pfleeger and Hatton [23], the design of the CDIS, a system with safety-critical requirements, incorporated the use of traditional and formal methods. They concluded that formal design yielded highly reliable code. The design of this system was expressed in four parts with a design overview document to connect the parts:

- The application code was designed using VDM specifications, created as refinements of the core requirements specification.
- The design specification of this system's local area network used a mixture of VDM and the Calculus of Communicating Systems (CCS).
- Concurrency requirements were derived by using Finite State Machines.
- The user interface was described in pseudo-code.

Larsen, Fitzgerald, and Brookes [19] assessed the impact of introducing a modest amount of formal specification into an existing development process for a security-critical system. The study involved the parallel development by two separate engineering teams of a system component called the trusted component. One team developed it utilizing traditional methods. The other team augmented their traditional methods with formal specifications wherever they felt it appropriate. They used VDM during the design phase to specify type definitions and procedural functionality. Larsen observed during design reviews that formal software design was specified at a higher level of abstraction than would normally be expected. This means that more design decisions would have to be made by the implementor - relying on the programmer to make design decisions is a controversial issue.

Halang and Kramer [11] utilized formal methods for the design of programmable logic chips (PLCs). PLCs are replacing hard-wired switching networks in a range of applications. They used Obj and high-level Petri-net models to specify requirements and designs of the system. Formal requirements were converted to function or black boxes (blocks) in the design, and the blocks were interconnected to form a complex program. Petri net models of the function block diagrams, which include algebraic interface specifications, were used to determine the static and dynamic properties of both distributed and concurrent programs. The designs were verified against the critical requirements using proof techniques utilizing the Obj3 system. They observed that formal requirements and design specifications make it possible to reuse function blocks and the proofs to verify them. Formal specifications and designs were shown to demonstrate the consistency between the specifications and the code for all possible data inputs. They also observed that certification authorities can use formal verification techniques to show a program's dependability and to prove the correctness of the entire program with respect to its requirements and design specifications

## Comparison of Method Characteristics

Formal methods can be compared on any number of characteristics. Table 1 provides a comparison of Z and VDM based on the most important characteristics of the specification languages ([6], [14], [27], [28]):

- All formal methods are based in mathematics. Some methods are based on set theory and first order predicate calculus. Others are based on temporal logic, which is an extension of propositional logic to formalize how the truth values of some propositions alter with the time at which they are evaluated. Both VDM and Z are based on set theory and first order predicate calculus.
- Formal methods utilize either a property-oriented or model-oriented approach and have different levels of rigor. Model-oriented formal methods specify system behavior by the construction of a mathematical model with an underlying state (data) and a collection of operations on that state. Property-oriented formal methods define system behavior indirectly by stating a set of properties, usually in the form of axioms, that the system must satisfy. VDM and Z are both of the model-oriented type.
- Both VDM and Z are only used to specify the functional aspects of systems. Neither support all aspects of design.
- VDM and Z differ in their appearance on the page as well. VDM specifications are heavily loaded with keywords (e.g. **ext**, **rd**, **wr**, **dom**, **post**). The boxes and schemas of Z distinguish its appearance from VDM's.
- Z provides a schema calculus for combining specifications, for example, to handle error handling or status information. VDM provides no similar mechanism.
- The syntax of Z and VDM are different in defining before and after states of variables.
- Inputs and outputs are also distinguished differently in the syntax of Z and VDM.
- VDM and Z have different methods for describing whether variables can be changed or are read-only.
- VDM explicitly handles exception handling. Z does not.

In summary, although different in syntax and structure, Z and VDM do not differ radically from one another. They are similar in their foundations and goals, and both allow the specifier to state requirements precisely and refine these specifications into designs correctly

**Table 1: Comparison of VDM and Z Characteristics**

<b>Characteristic</b>	<b>VDM</b>	<b>Z</b>
Mathematical Basis	<ul style="list-style-type: none"> <li>• Set Theory</li> <li>• First Order Predicate Calculus</li> </ul>	<ul style="list-style-type: none"> <li>• Set Theory</li> <li>• First Order Predicate Calculus</li> </ul>
Model or Property Oriented	Model-Oriented	Model-Oriented
Specified Properties	Behavior (functionality) and design of sequential programs	Behavior (functionality) and design of sequential programs
Design Aspects Not Supported	<ul style="list-style-type: none"> <li>• System Timing</li> <li>• Concurrency</li> <li>• User Interface</li> </ul>	<ul style="list-style-type: none"> <li>• System Timing</li> <li>• Concurrency</li> <li>• User Interface</li> </ul>
Page Appearance Differences	Heavily Keyword oriented (e.g. pre-, post-, invariants)	Boxes or Schemas
Structuring Mechanisms	None	Schema Calculus which allows various schemas to be combined to form new schemas
Specification of State Changes	<ul style="list-style-type: none"> <li>• Before: hooked variables</li> <li>• After: unhooked variables</li> </ul>	<ul style="list-style-type: none"> <li>• Before: undecorated variables</li> <li>• After: primed variables</li> </ul>
Identification of Inputs and Outputs	No explicit way of specifying	<ul style="list-style-type: none"> <li>• Inputs: variable names ending in “?”</li> <li>• Outputs: variable names ending in “!”</li> </ul>
Distinguishing Constants and Variables	Keyword <b>wr</b> for variables that can change. Keyword <b>rd</b> for variables that are read only	<b>D</b> for variables that can change.  <b>X</b> for variables that are read only.
Exception Handling Support	Has notation to mark and define error conditions	Not explicitly supported in notation. Defined by specifying an operation using structuring mechanisms.

## Comparison of Tool Support

Tool support provided for VDM and Z are shown in Table 2. Using formal methods and specification languages in practice requires that any candidate method have tool support in one or more of the following categories:

- **Editing Facilities.** Tools in this class are used to input and edit formal specifications.
- **Document Printing.** If computers are used to capture formal specifications, methods need to be provided to print the specifications in a “pretty” format. L<sup>A</sup>T<sub>E</sub>X, troff, and postscript appear to be the most popular formats supported.
- **Visual Specification.** Some tools are available that allow specifiers to specify requirements using pictures or graphics. These tools then automatically produce and maintain the specifications in a target formal specification language.
- **Syntax Checking.** Syntax checking tools check specifications for grammatical correctness. Most syntax checkers allow specifications to be imported from source files in ASCII or mathematical syntax.
- **Semantic Analysis.** Semantic analyzers carry out checks on the well-formedness of the specifications, including checking declaration scope, use of state variables, use of hooked and bang values, and use of record types.
- **Proof Support.** Proof checkers and theorem provers assist users in deriving and managing formal proofs.
- **Code Generation.** Tools are beginning to appear which generate computer language source code representations of the specifications.

The fact that similar tool support exists for both Z and VDM may indicate that both methods are equally popular in the formal methods community.

**Table 2: Tool Support Comparison**

<b>Class of Tool</b>	<b>VDM Tools Available</b>	<b>Z Tools Available</b>
Editing Facilities	<ul style="list-style-type: none"> <li>• VDM through Pictures (VtP) by IDE</li> </ul>	<ul style="list-style-type: none"> <li>• IBM - Z Tool</li> <li>• Computer Aided Design in Z (CADiZ)</li> </ul>
Document Printing	<ul style="list-style-type: none"> <li>• SpecBox (supports L<sup>A</sup>T<sub>E</sub>X Output)</li> </ul>	<ul style="list-style-type: none"> <li>• Computer Aided Design in Z (CADiZ) - supports troff output in UNIX</li> </ul>
Visual Specification	<ul style="list-style-type: none"> <li>• VDM through Picture (VtP) by IDE</li> </ul>	<ul style="list-style-type: none"> <li>• None Found</li> </ul>
Syntax Checking	<ul style="list-style-type: none"> <li>• SpecBox</li> <li>• Delft VDM SL</li> </ul>	<ul style="list-style-type: none"> <li>• Computer Aided Design in Z (CADiZ)</li> <li>• ZTC, a free tool for PC and UNIX</li> <li>• Fuzz, a commercial product for DOS and UNIX</li> </ul>
Semantic Analysis	<ul style="list-style-type: none"> <li>• SpecBox</li> <li>• Delft VDM SL</li> </ul>	<ul style="list-style-type: none"> <li>• Computer Aided Design in Z (CADiZ)</li> </ul>
Proof Support	<ul style="list-style-type: none"> <li>• mural - validate a formal spec against an informal description proof assistant</li> </ul>	<ul style="list-style-type: none"> <li>• Zola from Imperial Software Technology, Inc.</li> </ul>
Code Generation	<ul style="list-style-type: none"> <li>• VDM Domain Compiler</li> </ul>	<ul style="list-style-type: none"> <li>• None Found.</li> </ul>

## Comparison from Field Experience

The formal methods community faces a challenge to demonstrate and document positive benefits and positive return on investment from the application of formal methods. This challenge must be met for formal methods to receive acceptance and use as sound engineering methods within the United States. No single formal method has been applied throughout a total system development to date.

Two systems, however, that were specified and designed using Z and VDM have quantitative results documented in the technical literature:

- The Customer Information Control System (CICS), a transaction processing system developed using Z by IBM in the United Kingdom.
- The Central Control Function (CCF) Display Information System (CDIS), developed by Praxis using VDM.

Table 3 compares these two projects and their reported results. These two systems are quite different applications. One is a general purpose On-Line Transaction Processing (OLTP) system, and the other is a custom, real-time system with severe constraints.

The principal goal of this comparison is to identify and compare lessons-learned from applying each method and to examine the impact of each method on rework and development costs. Significant cost savings in a development project can be achieved through an increase in productivity, a reduction in the number of defects induced into a system, or the detection and resolution of defects closer to the point of insertion of the defects. Finding and resolving a defect after the product is released increases the repair from 100 to 250 times the cost of finding and resolving a defect during the design or coding phase [22].

Certainly no firm conclusions can be drawn from just one application of each method. However, some interesting comparisons between the methods can be seen by examining Table 3 and by comparing this data with industry standards and experience with Cleanroom developments:

- The size of the formal design specification in pages per Thousand Source Lines Of Code (KSLOC) was approximately 2.75 times greater in the Z specification than in the VDM specification. This difference could have been caused by a number of factors, such as, the differences in programming languages used with each method (C vs. PLAS) and the experience of the VDM team and their customer with the formal method used.
- The defect rate of 11 defects/KSLOC as reported by the VDM team compares unfavorably to an industry average of 7 defects/KSLOC [22]. I found this statistic quite surprising and would have expected this statistic to have been closer to reports from Cleanroom experience [22], where defect rates less than 1 defect/KSLOC have been reported. The approximate 40% reduction in defect rates as reported from the Z application compares more favorably to experiences of Cleanroom. Cleanroom reports, however, greater than 85% reduction in defects as compared to traditional methods.
- The productivity of the VDM team (13 SLOC/Day) represents an 85% improvement over industry standards of 7 SLOC/Day [22]. By comparison, the 9% improvement as reported by the Z project represents a fairly modest increase in productivity as compared to industry standards.
- The 0.75 defects/KSLOC found after product release from the VDM project is an improvement over the industry average of approximately 1.0 defects/KSLOC [22]. Although the Z project stated no explicit defect rate, if IBM was experiencing industry average defect rates after product release prior to using formal methods, a 2.5 times improvement would result in approximately 0.30 defects/KSLOC after product release. These numbers compare unfavorably to a post-release defect rate of less than 0.05 defects/KSLOC [22] from Cleanroom experience.
- Both projects showed similar positive findings in terms of the severity of reported problems from field use. This shows that few, if any, design problems existed in the system at the point of release of the products. These findings are similar to observations from Cleanroom developments.
- The observed deficiencies from the CDIS project shows that VDM cannot be used to explicitly state all requirements of a system. Although no deficiencies were identified in the CICS project for Z developed projects, the author of the CDIS paper had considered utilizing Z for the CDIS project, but eliminated it from consideration because Z's error-handling conventions were clumsy.
- The conclusions stated about formal methods from both projects were of special interest. The CDIS system was a project developed with a specific customer involved in the whole process; whereas the CICS project was developed for a large class of users. The conclusions for the CDIS project came from the customers viewpoint. Conclusions for the CICS project came from the development project and management team. The advantages stated for CDIS were expected and confirm the objectives of formal methods: precise and comprehensive system specifications.

**Table 3: Practical Experience with Z and VDM in Specification and Design**

<b>Comparison Factor</b>	<b>VDM Application</b>	<b>Z Application</b>
Project Name	Central Control Function (CCF) Display Information System (CDIS)	Customer Information Control System (CICS)
Information Source	Hall [12]	Houston and King [16]
Application Type	Air Traffic Control Information System (Real Time System). CDIS displays information about incoming and outgoing flights, weather conditions, and equipment status at airports.	On-Line Transaction Processing System. Generic facilities can be tailored to business requirements by an Application Programming Interface (API) used to invoke CICS services.
Additional Constraints	<ul style="list-style-type: none"> <li>• Performance: information must be displayed in 1-2 seconds of receipt.</li> <li>• Availability: 99.97%</li> <li>• No Single Point of Failure</li> </ul>	None reported
Company Using Formal Method	Praxis	IBM
Location of Use	United Kingdom	United Kingdom
Customer Base or System	London Area and Terminal Control Centre for flights at Heathrow and Gatwick airports	General Business Community
Date of General Release of Completed System	Autumn, 1993	June, 1990
Development Type	New Development	Enhancement to Existing, 22 year old, System.
Rationale for Use of Formal Methods	Safety, Life Critical System.	<ul style="list-style-type: none"> <li>• Clarify internal interfaces</li> <li>• Provide basis for future</li> </ul>
Rationale for Choice of Method	VDM was familiar to requirements team and had been used on other projects for customer.	Unknown. Author says Z was selected after “much research”.

**Table 3: Practical Experience with Z and VDM in Specification and Design (continued)**

<b>Comparison Factor</b>	<b>VDM Application</b>	<b>Z Application</b>
Language Used for Functional Specifications	<ul style="list-style-type: none"> <li>• VDM for operational and data specifications</li> <li>• Entity Relationship Diagrams for World Model</li> <li>• Data Flow Diagrams for Processing Requirements</li> </ul>	Z
Language Used for System Specifications	<ul style="list-style-type: none"> <li>• Functional Design: VVSL</li> <li>• Process Design: Finite-State Machines to diagram and VVSL to characterize complex states.</li> <li>• User Interface: IBM Presentation Manager</li> <li>• LAN Design: Calculus of Communicating Systems</li> </ul>	Z
Language Used for Designs	<ul style="list-style-type: none"> <li>• Functional Design: VVSL</li> <li>• Process Design: Finite-State Machines to diagram and VVSL to characterize complex states.</li> <li>• User Interface: IBM Presentation Manager</li> <li>• LAN Design: Calculus of Communicating Systems</li> </ul>	Z
Were Proofs Used?	Yes, on LAN Design because it was such a critical element.	No
Language Used for Source Code	C	PLAS
Total KSLOC from Formal Methods	197 KSLOC	~48 KSLOC
Percent of New and Modified Source Code Resulting from Formal Methods Design	100%	18% (48 KSLOC/268 KSLOC)
Length of Formal Design Specifications	3,000 pages = 15 pages/KSLOC	2,000 pages = 42 pages/KSLOC

**Table 3: Practical Experience with Z and VDM in Specification and Design (continued)**

<b>Comparison Factor</b>	<b>VDM Application</b>	<b>Z Application</b>
Defect Rates with Formal Methods	11 Defects/KSLOC at System & Integration Test	Estimated 40% Reduction <sup>1</sup>
Productivity	13 SLOC/Person-Day	9% improvement (attributed to less rework during development)
Defects Reported by Customers on Formally Developed Code	0.75 Defects/KSLOC	2.5 times reduction after 8 months of operation
Severity of Customer-Reported Problems	Few faults were specification or requirements based, and they were less costly to repair.	Much less than for other projects
Observed Deficiencies of Selected Formal Method	<p><u>Requirements:</u></p> <ul style="list-style-type: none"> <li>• Cannot distinguish between essential and merely desirable functions</li> <li>• Cannot specify global properties</li> <li>• Cannot specify usability, performance, reliability and safety requirements.</li> </ul> <p><u>System Spec</u></p> <ul style="list-style-type: none"> <li>• Cannot specify user interface.</li> <li>• No concurrency specification</li> </ul>	None reported
Conclusions	<p><u>Advantages:</u></p> <ul style="list-style-type: none"> <li>• Specification was comprehensive</li> <li>• Specification was precise</li> <li>• System tests derived from specs, so customer could see level of testedness.</li> </ul> <p><u>Disadvantages:</u></p> <ul style="list-style-type: none"> <li>• Difficult to get overview from formal methods.</li> <li>• Difficult to interpret spec</li> </ul>	<ul style="list-style-type: none"> <li>• It is possible to introduce Z into development process without formal refinement.</li> <li>• Overall quality of the product improved, as measured by defects/KSLOC.</li> <li>• Faults were found earlier in the development cycle.</li> </ul>

<sup>1</sup> The defect rate improvement was not explicitly stated in the text of the source document. It was computed based on the graph shown in Figure 1, which also appeared (without the grid) in the text of the source document. No specific values were given for the Y-axis. However a linear grid was overlaid on this graph, values were interpolated from the graph, and a total percent improvement was calculated.

### Table 3 Conclusions

The conclusions I draw from Table 3 (with only one data point per formal method) is that VDM specifications may be easier and more productive to use (based on productivity figures), but the higher than industry average in defects that are introduced using the VDM method compares very unfavorably to Z defect rates. The high cost of rework [22] observed in the US software industry could, overall, make VDM more costly to use than Z.

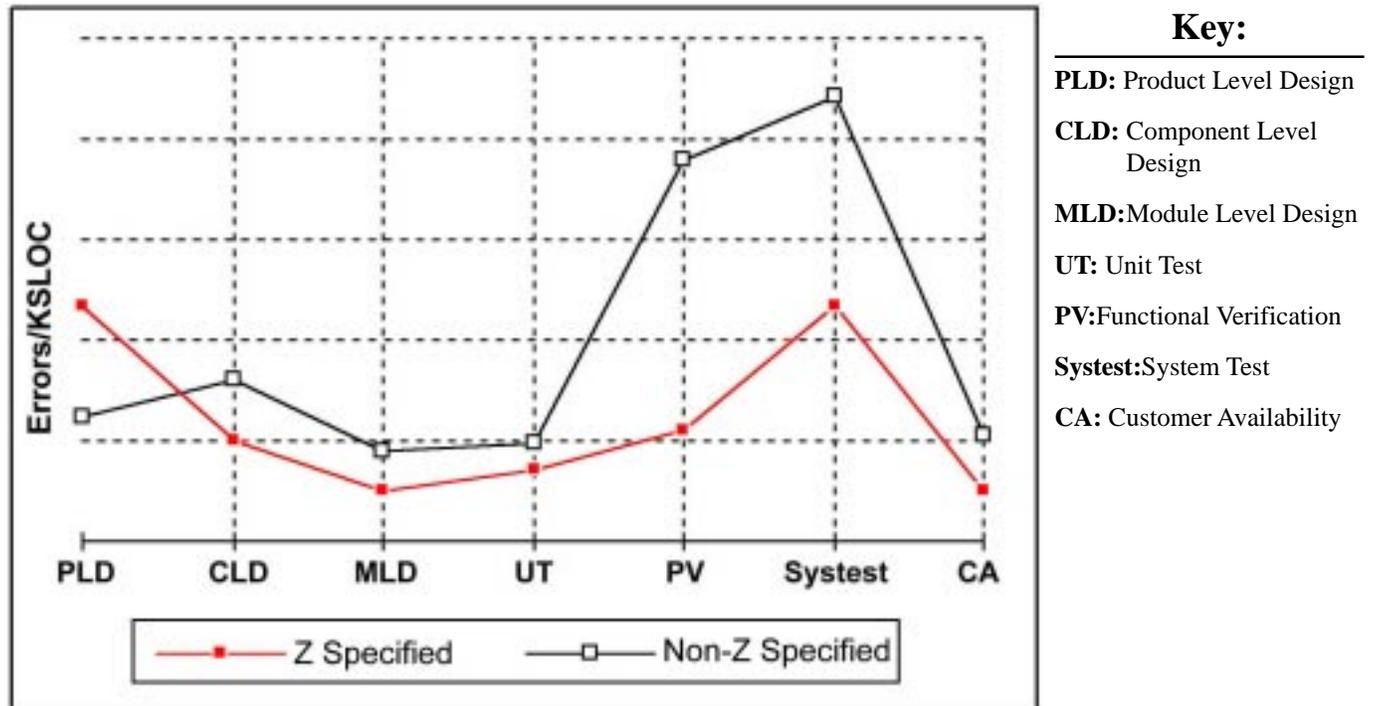


Figure 1: Defect Rates for CICS Project

### Detailed Financial Analysis

A previous DACS report [22] presented a detailed cost model. This model was used to analyze development costs and cost savings from reduced rework resulting from software process improvements, such as an increased SEI Capability Maturity Model (CMM) ranking, software inspections, software reuse, and Cleanroom Software Engineering. The model was implemented as a Microsoft Excel spreadsheet.

That spreadsheet model has been extended based on the data from Table 3 (which contains only one data point for each formal method). This model can be used to compare and contrast Z and VDM with each other, as well as with traditional software development. Results are shown in Table 4. The first column labels the parameters and some outputs of the model. The remaining three columns represent the three methods to be compared and contrasted: VDM, Z, and traditional software development. Each row in Table 4 is discussed following the table.

**Table 4: Financial Comparison of Formal Methods**

	<b>Formal Methods VDM</b>	<b>Formal Methods Z</b>	<b>Traditional Methods</b>
Estimated SLOC	32,000 SLOC	32,000 SLOC	32,000 SLOC
Productivity Improvement	86%	9%	0%
Estimated Effort	211.9 Person Months	361.6 Person Months	368.0 Person Months
Equivalent Cost	\$1,255,508	\$2,142,480	\$2,180,400
Average Defects/KSLOC	11	4.2	7
Expected Number of Defects	352	134.4	224
<b>% Defects Introduced by Phase</b>			
Design	35 %	35 %	35 %
Coding	65 %	65 %	65 %
<b>% Defects Detected by Phase</b>			
Design	47 %	48 %	40 %
Coding	41 %	42 %	35 %
Test	86 %	87 %	73 %
Defects Left for Customer	24.29	8.43	33.81
Post Release Defects/KSLOC	0.76	0.26	1.06
Maintenance Costs	\$242,923	\$84,313	\$338,083
Total Rework Costs	\$338,814	\$177,771	\$417,719
Reduction from Traditional Methods	18.89 %	57.44 %	N.A.
Total Life Cycle Costs	\$1,498,431	\$2,226,793	\$2,518,483
Reduction from Traditional Methods	40.50 %	11.58 %	N.A.
Schedule Length	19.10 Months	23.00 Months	23.18 Months
Reduction from Traditional Methods	17.60 %	0.78 %	N.A.

- The spreadsheet was designed to allow anyone to start from a problem of their own choosing. It thus begins with an estimate of the size of the program (in SLOC) to be estimated for the various methods. The Intermediate COCOMO 1.1 model was used to estimate costs and schedules. SLOC is the main cost driver in intermediate COCOMO 1.1. The example shown in Table 4 is a medium-sized semidetached system with very high reliability requirements, product complexity, and execution time constraints. These seem to be typical characteristics of systems to which formal methods are applied [15].

- The COCOMO estimates were adjusted to reflect the observed productivity gains from formal methods. As shown in Table 3, the Z project had a 9% improvement, and the VDM project produced 13 SLOC per person day. If one accepts that the industry average productivity is 7 SLOC per person day, VDM represents almost a doubling of productivity when compared to traditional methods.
- Estimated effort is a COCOMO output. Effort and schedule estimates include a planning and requirements phase. Both VDM and Z compare favorably to traditional software development methods, with VDM being much better than traditional methods.
- Equivalent cost is estimated based on the assumption of a cost of \$5,925 per person month. Comparisons based on equivalent cost are unchanged from comparisons based on effort.
- The average number of defects per KSLOC is an observed parameter. Traditional developments typically observe 7 defects per KSLOC [22]. As shown in Table 3, 11 defects per KSLOC were observed during the VDM project, and a 40% reduction was observed during the Z project. If this 40% reduction were from the “industry average,” the Z project would have experienced 4.2 defects per KSLOC. The VDM values compare unfavorably to the other methods. The Z method experiences the best of the three.
- The defects expected is merely the product of the average number of defects per KSLOC and the size in KSLOC. This is the number of defects introduced throughout the life cycle.
- Rework is reduced by introducing less defects and by detecting and removing defects as close to the point of defect insertion as possible. However, the pattern of defect insertion and removal by life cycle phase has not been reported for VDM and Z. Thus, the percent defects introduced by phase and percent defects detected by phase are taken from data on traditional methods. The percent defects detected by phase were uniformly increased for VDM and Z to obtain a total percentage of defects removed that matches reported data within rounding errors.
- Applying the defect introduction and removal percentage to the expected number of defects, one obtains the defects left for the customer. The VDM project observed 0.75 defects per KSLOC in defect reports by customers. The Z project observed a 2.5 times reduction (where the industry average is approximately 1 defect per KSLOC). The Z post release defect rate is better than the VDM project. Both VDM and Z are better than traditional methods.
- Total rework costs are computed and compared to traditional methods. It is assumed that rework hours consist of 2.5 hours per defect during design and coding, 25 hours per defect during test, and 250 hours per defect during maintenance. Rework costs are assumed to be \$39 per hour. As can be seen in Table 4, VDM rework costs are reduced only somewhat from traditional methods, while Z compares very favorably to traditional methods.
- The next block compares the total costs (development costs and rework from maintenance) associated with software development. The “bottom line” shows that VDM is less costly than Z, but both VDM and Z cost less than traditional methods.
- The final block compares estimated schedule length from Cocomo. Z has a schedule length similar to traditional methods. VDM has a shorter schedule than Z.

## References

1. D. Blyth, C. Boldyreff, C. Ruggles, and N. Tetteh-Lartey, "The Case for Formal Methods in Standards," *IEEE Software*, pp. 65-67.
2. J. P. Bowen and M.G. Hinchey, "Seven More Myths of Formal Methods," *IEEE Software*, July 1995, pp. 34-41.
3. N. Delisle and D. Gartan, "A Formal Specification of an Oscilloscope," *IEEE Software*, September 1990, pp. 29-36.
4. N. Fenton and S. L. Pfleeger, "Can Formal Methods Deliver?," *Computer*, February 1997, pp. 34.
5. C. Fidge, P.Kearney, and M. Utting, "A Formal Method for Building Concurrent Real-Time Software," *IEEE Software*, March/April 1997, pp. 99-106.
6. B. Fields, "A Guide to Reading VDM Specifications," Technical Report UMCS-92-12-4, Department of Computer Science: University of Manchester, <http://www.cs.man.uk/>, 1992.
7. S. L. Gerhart, "Applications of Formal Methods: Developing Virtuoso Software," *IEEE Software*, September 1990, pp. 6-10.
8. S. L. Gerhart, D. Craigen, and T. Ralston, "Experience with Formal Methods in Critical Systems," *IEEE Software*, January 1994, pp. 21-28.
9. S. L. Gerhart, D. Craigen, and T. Ralston, "Regulatory Case Studies," *IEEE Software*, January 1994, pp. 30-39.
10. W.W. Gibbs, "Software's Chronic Crisis," *Scientific American*, September 1994, pp. 86-95.
11. W. A. Halang and B. J. Kramer, "Safety Assurance in Process Control," *IEEE Software*, January 1994, pp. 61-67.
12. A. Hall, "Using Formal Methods to Develop an ATC Information System," *IEEE Software*, March 1996, pp. 66-76.
13. J. A. Hall, "Seven Myths of Formal Methods," *IEEE Software*, September 1990, pp. 11-19.
14. I. J. Hayes, C. B. Jones and J. E. Nicholls, "Understanding the Differences between VDM and Z," Technical Report UMCS-93-8-1, Department of Computer Science: University of Manchester, <http://www.cs.man.uk/>, 11 August 1993.
15. M. G. Hinchey and J. P. Bowen (editors), *Applications of Formal Methods*, Prentice Hall, 1995.
16. I. Houston and S. King, "CICS Project Report: Experiences and Results from the use of Z", *Proceedings of VDM '91*, Volume 551, Springer Verlag, Berlin, 1991, pp. 588-596.
17. R. A. Kemmerer, "Integrating Formal Methods into the Development Process," *IEEE Software*, September 1990, pp. 37-50.
18. J. Knight and B. Littlewood, "Critical Task of Writing Dependable Software," *IEEE Software*, January 1994, pp. 16-20.
19. P. G. Larsen, J. Fitzgerald, and T. Brookes, "Applying Formal Specification in Industry," *IEEE Software*, May 1996, pp. 48-56.

20. R. Linger and H. Mills, "A Case Study in Cleanroom Software Engineering: The IBM COBOL Structuring Facility," *Proc. Compsac*, IEEE CS Press, Los Alamitos, Calif., 1988, pp. 10-17.
  21. Luqi and J. A. Goguen, "Formal Methods: Promises and Problems," *IEEE Software*, January/February 1997, pp.73-85.
  22. T. L. McGibbon, "A Business Case for Software Process Improvement," DACS State of the Art Report, September, 1996.
  23. S. L. Pfleeger and L. Hatton, "Investigating the Influence of Formal Methods," *Computer*, February 1997, pp. 33-43.
  24. J. M. Spivey, "Specifying a Real-Time Kernel," *IEEE Software*, September 1990, pp. 21-28.
  25. M. Srivas and M. Bickford, "Formal Verification of a Pipelined Microprocessor", *IEEE Software*, September 1990, pp. 52-64.
  26. L. G. Williams, "Assessment of Safety-Critical Systems," *IEEE Software*, January 1994, pp. 51-60.
  27. J. M. Wing, "A Specifier's Introduction to Formal Methods," *Computer*, September 1990, pp. 8-24.
  28. J.C.P. Woodcock, "Structuring Specifications in Z", *Software Engineering Journal*, January 1989, pp.51-66.
- 

## About the Author

**Tom McGibbon** is director of the Data and Analysis Center for Software (DACs), the DoD's Software Information Clearinghouse. I ask that all readers of this paper visit the DACs outstanding web site at <http://www.dacs.dtic.mil/>.

Tom is a 1973 graduate of Clarkson University, with a BS Degree in Mathematics and Computer Science. He is an overworked Masters graduate student in the Software Engineering program at Southern Methodist University.

### Contact Information:

Thomas McGibbon  
DoD Data & Analysis Center for Software (DACs)  
ITT Industries - Systems Division  
775 Daedalian Drive  
Rome, NY 13441-4909  
  
(315) 334-4905, (315) 334-4964 - Fax  
[tmcgibbo@dacs.dtic.mil](mailto:tmcgibbo@dacs.dtic.mil)

## Annotated Bibliography

J. P. Bowen and M.G. Hinchey, "Seven More Myths of Formal Methods," *IEEE Software*, July 1995, pp. 34-41.

This paper expands on J.A. Halls seven myths of FMs and examines seven more myths:

- Formal Methods Delay the Development Process.
- Formal Methods lack Tools.
- Formal Methods Replace Traditional Engineering Design Methods.
- Formal Methods only Apply to Software.
- Formal Methods are Unnecessary.
- Formal Methods are not Supported.
- Formal Methods People Always Use Formal Methods.

N. Fenton and S. L. Pfleeger, "Can Formal Methods Deliver?", *Computer*, February 1997, pp. 34.

This brief article examines the inhibitors of the application of formal methods and document cases in which formal methods were used throughout the development cycle. This article contradicts other claims that formal methods reduce number of defects.

B. Fields, "A Guide to Reading VDM Specifications," Technical Report UMCS-92-12-4, Department of Computer Science: University of Manchester , <http://www.cs.man.uk/> , 1992.

This paper provides a summary of the syntax of the Specification Language for the Vienna Development Method (VDM). All other VDM references that were found were too voluminous to be useful for this paper. It is not a tutorial or a formal description of VDM. It briefly examines expressions, data types, how definitions are made, and short examples of the use of VDM.

C. Fidge, P.Kearney, and M. Utting, "A Formal Method for Building Concurrent Real-Time Software," *IEEE Software*, March/April 1997, pp. 99-106.

This article discusses the use of Z in developing real-time requirements and the application of Z to concurrency issues.

S. L. Gerhart, D. Craigen, and T. Ralston, "Experience with Formal Methods in Critical Systems," *IEEE Software*, January 1994, pp. 21-28.

This article provides an analysis of the practical application of formal methods in five commercial and four regulatory applications. Regulatory cases exhibit safety or security critical attributes. The authors claim that there has been no widespread or repeat penetration of formal method use.

W.W. Gibbs, "Software's Chronic Crisis," *Scientific American*, September 1994, pp. 86-95.

This paper exposes a larger scientific community (the audience of a *Scientific American* journal article) than the software community to the inherent problems in the software industry and the benefits resulting from use of formal methods. The author examines the critical problems facing the software industry. Formal methods are suggested as one approach to overcoming these problems.

W. A. Halang and B. J. Kramer, "Safety Assurance in Process Control," *IEEE Software*, January 1994, pp. 61-67.

The authors examined the use of formal methods in Programmable Logic Controllers. Programmable Logic Controllers (PLCs) are replacing hard wired switching networks. PLCs can process more information faster than the switching networks. In PLCs, exhaustive testing is impossible and errors cannot be detected solely by peer reviews. They used Obj and Obj3 to automate part of specification testing and formal verification. Obj is an algebraic language that lets one specify requirements and designs independently of any data representations and implementation.

A. Hall, "Using Formal Methods to Develop an ATC Information System," *IEEE Software*, March 1996, pp. 66-76.

This was the best article found on the application of formal methods. Formal methods were used in the specification, design and verification of the Central Control Function (CCF) Display Information System (CDIS). Because of size and complexity of the CDIS, the developers used several formal methods to develop the sequential and concurrent aspects of the CDIS. The author describes the use of formal methods in each phase of development. VDM, and its variants, were the primary formal specification language used. Productivity and error rate data is shown.

I. J. Hayes, C. B. Jones and J. E. Nicholls, "Understanding the Differences between VDM and Z," Technical Report UMCS-93-8-1, Department of Computer Science: University of Manchester , <http://www.cs.man.uk/> , 11 August 1993.

This paper examines interesting differences and similarities between Z and VDM. Ideas are presented in the form of a imaginary dialog between several people. This paper compares appearance on the page, data types, states, initialization, operations, syntax, preconditions, and exception handling.

I. Houston and S. King, "CICS Project Report: Experiences and Results from the use of Z", *Proceedings of VDM '91*, Volume 551, Springer Verlag, Berlin, 1991, pp. 588-596.

This paper provides the details of actual experience with Z. Customer Information Control System (CICS) is a transaction processing system developed by IBM in the UK. CICS consists of 800,000 Lines of Code (LOC), of which 50,000 LOC were new or modified code. Thirty seven thousand new or modified LOC were completely specified using Z, and 11,000 LOC were partially specified using Z. The authors claim that Z reduced development costs and error rates.

J. Knight and B. Littlewood, "Critical Task of Writing Dependable Software," *IEEE Software*, January 1994, pp. 16-20.

This is a good introductory article showing why and how formal methods are applied to safety critical software. In safety critical systems, the consequences of failure are extremely high, usually a threat to human life. Safety-critical systems are expected to never fail. When a computer system can cause a catastrophic failure, developers and regulators will apply very high dependability criteria to both the hardware and the software. Formal methods have been the method of choice for these needs.

P. G. Larsen, J. Fitzgerald, T. Brookes, "Applying Formal Specification in Industry," *IEEE Software*, May 1996, pp. 48-56.

In this paper, the authors describe the introduction of formal methods into the specification and modeling activities of the development of a security-critical system. They examine formal methods' effectiveness by comparing the results of two groups: one that applied formal methods and one that did not. Interesting comparisons of formal methods to traditional methods are made.

Luqi and J. A. Goguen, "Formal Methods: Promises and Problems," *IEEE Software*, January/February 1997, pp.73-85.

The authors examine the state of formal methods, and their ability to support small, medium and large projects. The evolutionary nature of software is considered. The authors distinguish between small, large, and huge grain methods. Classic formal methods fall into the small grain category. They have a mathematical basis at the level of individual statements and small programs, but rapidly hit a complexity barrier when programs get large, because they do not provide structuring or encapsulation. Nine specific problems with formal methods are discussed.

S. L. Pfleeger and L. Hatton, "Investigating the Influence of Formal Methods," *Computer*, February 1997, pp. 33-43.

The authors attempted to carefully gather and analyze data and results from a product developed with formal methods. Results obtained were inconclusive, but the authors discovered how to gather data better. They conclude that using formal specifications can lead to code that is relatively simple and easy to test. They believe that if you couple formal methods with thorough testing, you can produce highly reliable code.

L. G. Williams, "Assessment of Safety-Critical Systems," *IEEE Software*, January 1994, pp. 51-60.

In this article, the author compares application of the Software Cost Reduction (SCR) and VDM formal methods. His comparison illuminates features that could aid the Verification and Valiation (V&V) of specifications for safety-critical systems. The author believes that V&V of specifications against functional and safety requirements remains one of the most difficult and challenging aspects of critical-system development.

J. M. Wing, "Formal Methods," *Software Engineering Encyclopedia*, J. Marciniak - Editor, 1994.

This paper provides a good discussion of various ways of classifying Formal Methods (FMs). It provides a comprehensive introduction to the topics and issues that surround FMs. This paper examines formal specification languages in terms of syntactic domains, semantic domains, and a satisfies relationship. It examines behavioral specifications, structural specifications, and properties of specifications. It compares the strengths and weaknesses of various classes of FMs. It looks at the users and uses of FMs, as well as examining how various FMs can be used during each phase of the product lifecycle. It provides a partial taxonomy of types of FMs. It briefly compares Z, VDM and Larch. Finally, this paper examines the limits of FMs.