Department of Commerce
National Institute of
Standards and Technology
GCR 626

AN INTERNATIONAL SURVEY OF INDUSTRIAL APPLICATIONS OF FORMAL METHODS

VOLUME 2
CASE STUDIES

by
Dan Craigen
ORA Canada

Susan Gerhart
Applied Formal Methods

Ted Ralston
Ralston Research Associates

March 1993

Blank Page

AN INTERNATIONAL SURVEY OF INDUSTRIAL APPLICATIONS OF FORMAL METHODS

VOLUME 2
CASE STUDIES

ABSTRACT

Formal methods are mathematically-based techniques, often supported by reasoning tools, that can offer a rigorous and effective way to model, design and analyze computer systems. The purpose of this study is to evaluate international industrial experience in using formal methods. The cases selected are, we believe, representative of industrial-grade projects and span a variety of application domains. The study had three main objectives:

- To better inform deliberations within industry and government on standards and regulations;
- To provide an authoritative record on the practical experience of formal methods to date; and
- To suggest areas where future research and technology development are needed.

This is the second volume of a two volume final report on an international survey of industrial applications of formal methods. In this volume, we provide the details of the twelve case studies. For each of the case studies, we present a case description, summarize the information obtained (from interviews and the literature), provide an evaluation of the case, highlight R & D issues pertaining to formal methods and provide some conclusions.

AN INTERNATIONAL SURVEY OF INDUSTRIAL APPLICATIONS OF FORMAL METHODS

VOLUME 2
CASE STUDIES

INTRODUCTION

The purpose of this study is to evaluate international industrial experience in using formal methods. The cases selected are, we believe, representative of industrial-grade projects and span a variety of application domains. The intent of the study is threefold:

- To better inform deliberations within industry and government on standards and regulations;
- To provide an authoritative record on the practical experience of formal methods to date; and
- To suggest areas where further research and technology development are needed.

This study was undertaken by three experts in formal methods and software engineering: Dan Craigen of ORA Canada, Susan Gerhart of Applied Formal Methods, and Ted Ralston of Ralston Research Associates. Robin Bloomfield of Adelard was involved with the Darlington Nuclear Generating Station Shutdown System case.

Support for this study was provided by organizations in Canada and the United States. The Atomic Energy Control Board of Canada (AECB) provided support for Dan Craigen and for the technical editing provided by Karen Summerskill. The U.S. Naval Research Laboratories (NRL), Washington, D.C., provided support for all three authors. The U.S. National Institute of Standards and Technology (NIST) provided support for Ted Ralston.

This final report consists of two volumes. The first volume describes the reason for the study, the cases that were studied, our approach to performing the study, and our analysis and conclusions resulting from the cases investigated.

This second volume of the final report provides the details on the case studies. For each of the case studies we present a case description, summarize the information obtained (from interviews and the literature), provide an evaluation of the case, highlight R & D issues pertaining to formal methods, and provide some conclusions. Earlier drafts of the case studies were reviewed by the relevant participants.

AN INTERNATIONAL SURVEY OF INDUSTRIAL APPLICATIONS OF FORMAL METHODS

VOLUME 2
CASE STUDIES

TABLE OF CONTENTS

ABSTRACT i

INTRODUCTION ii

TABLE OF CONTENTS iii

1 STRUCTURED SYSTEMS ANALYSIS AND DESIGN METHOD TOOLSET 1
1.1 Case Description 1
1.1.1 Product and Process Profile 2
1.1.2 References 2
1.2 Interview Summary 3
1.2.1 Organizational Context 3
1.2.2 Project Context and History 3
1.2.3 Formal Methods Factors 4
1.2.4 Formal Methods and Tool Usage 5
1.2.5 Results 5
1.3 Evaluation 7
1.3.1 Background 7
1.3.2 Product Features 7
1.3.3 Process Features 8
1.3.4 Observations 9
1.3.5 Formal Methods R & D Summary 10
1.4 Conclusions 11

2 IBM'S CUSTOMER INFORMATION CONTROL SYSTEM 12
2.1 Case Description 12
2.1.1 Product and Process Profile 12
2.1.2 References 13
2.2 Questionnaire 1 14
2.3 Interview Summary 16
2.3.1 Organizational Context 16
2.3.2 Project Context and History 16
2.3.3 Application Goals 17
2.3.4 Formal Methods Factors 18

2.3.5 Formal Methods and Tool Usage 19
2.3.6 Results 20
2.4 Evaluation 22
2.4.1 Background 22
2.4.2 Product Features 22
2.4.3 Process Features 23
2.4.4 Observations 25
2.4.5 Formal Methods R & D Summary 26
2.5 Conclusions 27

3 CLEANROOM SOFTWARE METHODOLOGY 29
3.1 Case Description 29
3.1.1 Product and Process Profile 30
3.1.2 References 30
3.2 Interview Summary: IBM 31
3.2.1 Organization and Project Context 31
3.2.2 Formal Methods Factors 33
3.2.3 Formal Methods and Tool Usage 33
3.2.4 Results 35
3.3 Interview Summary: NASA Goddard Center 36
3.3.1 Organization and Project Context 36
3.3.2 Application 37
3.3.3 Formal Methods and Tool Usage 37
3.3.4 Results 39
3.4 Evaluation 40
3.4.1 Background 40
3.4.2 Product features 41
3.4.3 Process features 41
3.4.4 Observations 42
3.4.5 Formal Methods R & D Summary 43
3.5 Conclusions 45

4 DARLINGTON: TRIP COMPUTER SOFTWARE 46
4.1 Case Description 46
4.1.1 Product and Process Profile 46
4.1.2 References 47
4.2 Interview Summary 48
4.2.1 Organizational Context 48
4.2.2 Project Context and History 49
4.2.3 Application Goals 54
4.2.4 Formal Methods Factors 55
4.2.5 Formal Methods and Tools Usage 55
4.2.6 Result 57
4.2.7 Interview with a regulator: Richard Taylor 58

4.2.8 Interview with David Parnas 60
4.2.9 Interview with a consultant: Nancy Leveson 61
4.2.10 Project Milestones (OH Perspective) 62
4.2.11 Project Milestones (AECB Perspective) 63
4.3 Evaluation 65
4.3.1 Background 65
4.3.2 Product Features 65
4.3.3 Process Features 66
4.3.4 Observations 67
4.3.5 Formal Methods R & D Summary 68
4.4 Conclusions 69

5 LaCoS ESPRIT PROJECT 70
5.1 Case Description 70
5.1.1 References 71
5.2 Questionnaire 1 72
5.3 Interview Summary 75
5.3.1 Lloyd's Register 75
5.3.2 Interview with Matra Transport 80
5.4 Evaluation 83
5.4.1 Background 83
5.4.2 Product Features 84
5.4.3 Process Features 84
5.4.4 Observations 85
5.4.5 Formal Methods R & D Summary 86
5.5 Conclusions 88

6 MULTINET GATEWAY 89
6.1 Case Description 89
6.1.1 Product and Process Profile 90
6.1.2 References 91
6.2 Interview Summary 92
6.2.1 Organizational Context 92
6.2.2 Project Context and History 92
6.2.3 Application Goals 93
6.2.4 Formal Methods Factors 93
6.2.5 Formal Methods and Tools Usage 94
6.2.6 Results 95
6.2.7 Interview with an assessor: Jim Williams 95
6.3 Evaluation 97
6.3.1 Background 97
6.3.2 Product Features 98
6.3.3 Process Features 98
6.3.4 Observations 99

6.3.5 Formal Methods R & D Summary 100
6.4 Conclusions 101

7 SACEM - A RAILWAY SIGNALLING SYSTEM 102
7.1 Case Description 102
7.1.1 References 102
7.2 Questionnaire 1 103
7.3 KVS 105
7.4 CTDC Calcutta 106
7.5 Interview Summary 107
7.5.1 Organization 107
7.5.2 Project Context 107
7.5.3 Application 108
7.5.4 Formal Methods Usage 111
7.5.5 Results 113
7.6 Evaluation 115
7.6.1 Background 115
7.6.2 Product features 115
7.6.3 Process features 116
7.6.4 Observations 117
7.6.5 Formal Methods R & D Summary 117
7.7 Conclusions 119

8 NIST TOKEN-BASED ACCESS CONTROL SYSTEM (TBACS) 120
8.1 Case Description 120
8.1.1 Formal Methods Background 120
8.1.2 References 120
8.2 Questionnaire 1 121
8.3 Interview Summary 123
8.3.1 Organizational Context 123
8.3.2 Project Context and History 123
8.3.3 Application Goals 124
8.3.4 Formal Methods and Tools Usage 125
8.3.5 Results 128
8.4 Evaluation 130
8.4.1 Background 130
8.4.2 Product features 130
8.4.3 Process features 131
8.4.4 Observations 132
8.4.5 Formal Methods R & D Summary 133
8.5 Conclusions 135

9 TEKTRONIX: USE OF Z METHOD ON OSCILLOSCOPES 136
9.1 Case Description 136

9.1.1 References 136
9.2 Questionnaire 1 (Delisle) 137
9.3 Interview Summary 139
9.3.1 Organizational Context 139
9.3.2 Project Context and History 139
9.3.3 Applicatio n Goals 140
9.3.4 Formal Methods Factors 141
9.3.5 Formal Methods and Tools Usage 141
9.3.6 Results 143
9.4 Evaluation 144
9.4.1 Background 144
9.4.2 Product Features 145
9.4.3 Process Features 145
9.4.4 Observations 146
9.4.5 Formal Methods R & D Summary 148
9.5 Conclusions 149

10 TRAFFIC ALERT AND COLLISION AVOIDANCE SYSTEM (TCAS) 150
10.1.1 Product and Process Profile 151
10.1.2 References 151
10.2 Interview Summary 152
10.2.1 Organizati onal Context 152
10.2.2 Project Context and History 153
10.2.3 Applicatio n Goals 154
10.2.4 Formal Methods Factors 155
10.2.5 Formal Methods and Tools Usage 155
10.2.6 Results 156
10.2.7 Interview with Larry Nivert 156
10.2.8 Interview with David Lubkowski 157
10.2.9 Interview with a pilot 158
10.3 Evaluation 159

10.3.1 Background 159
10.3.2 Product Features 159
10.3.3 Process Features 160
10.3.4 Observations 161
10.3.5 Formal Methods R & D Summary 161
10.4 Conclusions 162

11 INMOS TRANSPUTER 163
11.1 Case Description 163
11.1.1 Product and Process Profile 163
11.1.2 Issues of Interest 164
11.1.3 References 164
11.2 Interview Summary 165
11.2.1 Organizational Context 165
11.2.2 Project Context and History 165
11.2.3 Application Goals 166
11.2.4 Formal Methods Factors 166
11.2.5 Formal Methods and Tools Usage 166
11.2.6 T800 Floating Point Unit 167
11.2.7 T800 Scheduler 167
11.2.8 T9000 Virtual Channel Processor 168
11.2.9 Tools 169
11.2.10 Results 169
11.3 Evaluation 170
11.3.1 Background 170
11.3.2 Product Features 171
11.3.3 Process Features 171
11.3.4 Formal Methods R & D Analysis 173
11.4 Conclusions 175

12 HEWLETT-PACKARD MEDICAL INSTRUMENTS 176
12.1 Case Description 176
12.1.1 Product and Process Profile 176
12.1.2 References 177
12.2 Interview Summary 178
12.2.1 Organizational Context 178
12.2.2 Project Context and History 179
12.2.3 Application Goals 179
12.2.4 Formal Methods Factors 180
12.2.5 Formal Methods and Tools Usage 180
12.2.6 Results 181
12.2.7 Defibrillator 182
12.3 Evaluation 183
12.3.1 Background 183

12.3.2 Product Features 183
12.3.3 Process Features 184
12.3.4 Observations 186
12.3.5 Formal Methods R & D Summary 186
12.4 Conclusions 187

Appendix A Acknowledgements 188

1 STRUCTURED SYSTEMS ANALYSIS AND DESIGN METHOD TOOLSET (SSADM)

Praxis Systems Ltd.

1.1 Case Description

Praxis Systems Ltd. is a software engineering company located in Bath, England. It was founded in 1983 and conducts business using both conventional software development techniques as well as formal methods. Praxis supplies high-integrity software systems and was the first independent software house to be registered under British Standard 5750 (in 1986), which is the UK equivalent of the ISO 9000 quality assurance standard.

Praxis has developed considerable strength in the area of formal methods for systems development which are now used extensively in the company. In the last three years, over a dozen significant projects have been carried out involving the use of formal methods including Vienna Development Methodology (VDM), Z, Communicating Sequential Processes (CSP), Temporal Logic, and an internally-developed Process Modelling Language. These projects have involved the production of both government-sponsored and commercial systems. Of these projects, we considered four to fit our definition of a real industrial application:

1. An operating system interface specified in VDM and CSP.

   Under contract to a commercial client, Praxis formally defined the interfaces between various components of a distributed factory control system using CSP. A particular component for which Praxis had design authority was subsequently specified using VDM and the relationship between the VDM and CSP specifications established.

2. A security policy model and extensions to a secure operating system specified in Z

   On a Government security project, Praxis has produced a VDM-based functional specification of secure enhancements to a secure operating system (B1/B2 level) and then developed what Praxis calls a "demonstrably conformant" C code to implement these enhancements.

3. Components of the upgrade to London's Air Traffic Control system specified in VDM

   As part of its advanced program to expand and develop the air traffic control system over the south-east of England, the Civil Aviation Authority is building a major new operations centre, the Central Control Function (CCF) facility, at the London Air Traffic Control Centre. Praxis has been selected to conduct a design study for the CCF Display and Information System (CDIS) which forms part of this program.

CDIS forms a vital component of the data entry and display equipment used by air traffic controllers. Praxis used VDM to do an abstract specification of the whole system and for some parts of the design.

4. The use of Z to specify SSADM infrastructure and toolset

   Under contract to a commercial client, Praxis has developed a Computer-Assisted Systems Engineering toolset to support the use of the CCTA standard development method SSADM. In this project, Z was used to develop a formal specification of the toolset infrastructure. In the early stages of requirements analysis, Z was used for concept exploration. Later, the project team developed guidelines for the use of Z with Object-Oriented Design methods.

We selected the SSADM project to be the case study for this study for several reasons. First, it was completed and data had been recorded for it. Second, our first choice was the CDIS London ATC project; however, it turned out to be premature as the project is just finishing and Praxis itself has not had time to analyze the results. Lastly, the SSADM project appeared to be a good example of how Praxis combines formal and informal methods in their normal system development process.

1.1.1 Product and Process Profile

Size: Total SSADM system size is 45,000 lines of non-empty, non-comment code, 37,000 lines of which were specified in 350 pages of Z schemas (550 schemas of which 280 defined top-level operations.

Stages:

July 1987	Project start
February 1988	Formal Specification completed
June 1989	Implementation delivered
4 phases	0. short requirements analysis
	1. infrastructure specified
	2. infrastructure software built
	3. tools built (mostly by client)

1.1.2 References

1. Hall, J. Anthony, "Seven Myths of Formal Methods", IEEE Software, September, 1990, pp. 11-21.

2. Rawlings, Rosamund, "Some Numbers from an Object-Oriented Development", in *Object-oriented Software Engineering: The Next Step*, Ed. Bruce Anderson, British Computer Society, London, 1990.

3. Brownbridge, David, "Using Z to Develop a CASE Toolset", in *Proceedings of the 1989 Z User Meeting*, Workshops in Computing Series, Springer-Verlag, December, 1989.

4. Hall, J. Anthony, Plenary 3 Keynote Address, *13th International Conference on Software Engineering*, Austin, Texas, May 13-17, 1991.

1.2 Interview Summary

Date of Interview:	May 15, 1992
Organization:	Praxis Systems plc, Bath, England
Respondents:	Anthony Hall, Rosamund Rawlings, Dave Ellis, Gavin Finnie, and David Brownbridge
Interviewers:	Dan Craigen and Ted Ralston (author)

1.2.1 Organizational Context

Praxis is a software engineering company providing services for consultancy to full turnkey implementations. At the time of the project, ELLA (a CAD behavioral description language) was still part of Praxis (the ELLA business was recently sold), and there were about 120 individuals in the company. Clients have usually been large organizations where quality was more important than cost (to a point). Praxis was incorporated in 1984 and its main message to its customers is that Praxis will use the most up-to-date methods and will follow a rigorous quality system (e.g., ISO 9000/BS 5750). Praxis was the first independent software house in the UK to be registered under these requirements.

1.2.2 Project Context and History

The SSADM project started in mid-1987 (not including proposal time) and completed in mid-1989. By February 1988, the specification was completed; in July, September and December 1988, and March 1989 incremental deliverables of the class hierarchy were made.

The product was to be in two parts, both in support of SSADM. The first was a framework---or, in Praxis' terminology, an "infrastructure"---for the entire method, to manage all the documents required by the method, to check the rules of the method, and to provide a project management framework. The second part was a set of automated tools for this infrastructure. Hence, the product was substantially more complex than a data flow tool.

At the initial stages, Praxis was concerned about the feasibility of the product, as the requirements were ambitious and required integrating components based on a variety of techniques (some AI, some object-oriented, some data flow) and involving a number of features such as version control, configuration management and the like. The main concerns were how to understand the required system and how to provide a toolset that was

feasible and within budget. There had been historical concerns with previous attempts to develop Integrated Programming Support Environments in Europe; many were far too ambitious.

There was a very short requirement phase (essentially, the client had already decided on the requirements) and a longer specification phase where formal methods were used to describe the infrastructure, but not the tool kit. Note that the basic structure of the tool was as both an infrastructure and a tool kit. The project was conducted in three phases over a two-year period. The first phase carried out the system specification and architecture, and chose an implementation language (Objective C) and libraries. The second phase entailed the design and implementation of the infrastructure, while the design and implementation of the tools was carried out in the final phase.

1.2.3 Formal Methods Factors

Praxis used Z to specify the infrastructure used by project teams using the SSADM method, a more or less standard method in aerospace and defense contracting work in Europe. The toolset runs in a multi-user environment, with various project-management and configuration-management utilities distributed among a number of networked workstations (Suns and PERQs, an ICL workstation). The environment provides a set of basic classes (e.g., diagram, table, and matrix) from which tools for structured analysis can be developed by specialization.

The process used involved a series of steps, beginning with a high-level specification, and moving down through successively lower levels toward implementation. First, the requirement (in English language) was formalized, and a high-level specification of the entire system was produced in Z (consisting of 350 pages and about 550 schemas defining about 280 operations). From this high-level specification, some lower level parts of the system were written in informal design documents, while others were written in Z. Also, only some of the lower level modules were specified in Z; others were written informally. Coding was carried out from both the informal designs and the lower level Z specifications.

The end product seemed to be less troublesome than normal. However, it is hard to determine whether this is due to the use of formal methods, the use of object-oriented programming, or the use of talented individuals. It is claimed in [3] that test suites were independently constructed by using the formal Z schemas as a reference, not lower-level code documentation (e.g., tests on an object class can be derived from the Z schema for that object, rather than the class description of the Objective C object class). There were few faults found at system test and during client usage. Having got the details right at the start helped and there were few integration problems.

1.2.4 Formal Methods and Tool Usage

The only tools used in the SSADM project were a prototype parser and type checker obtained from an Alvey R & D project called FORSITE. Neither were industrially robust, and the type checker was described as "eccentric" (i.e., it would get things wrong). This tool was used because a type checker was needed and the FORSITE type checker was the only one readily available at the time. It was barely adequate.

1.2.5 Results

Praxis provided the following data on this project during the interview. It pertains solely to the productivity issues.

The SSADM system was developed under BS 5750 product quality requirements which require full specification and design documentation. The effort figures include requirements capture and specification, initial investigations and architecture design, system specification in Z, implementation in Objective C, and system testing. They do not include final product testing or user documentation. The effort expended was recorded on timesheets during the project.

The scale of the project is indicated in the following table, which shows the number of classes and the size in thousands of lines of non-empty, non-comment code (KLOC).

Subsystem	Classes	KLOC*
1. Management Facilities	153	25
2. Tool-writers Class Library	99	12
3. Tools (5 only)	81	8
Total	333	45

* KLOC are non-empty non-comment lines

The effort expended on parts 1 and 2 combined totalled 2235 work-days; part 3 took 483 work-days. This gives a figure of 16.5 lines of code per work-day or 8 work-days per class.

The division of effort between process phases was broken down as follows:

Requirements	7%
Specification	29%
Development (incl. unit testing)	50%
Testing and Delivery	14%

The Z specification of parts 1 and 2 comprises 350 pages including the English annotations, broken down into 550 schemas of which 280 are top-level operations.

The object-oriented paradigm of "model-view-controller" was followed. The figures in the table below show the amount of code and number of classes in each to the libraries, while the "tool" number is the average of the five tools developed by Praxis (the full system was planned to have approximately 60 tools).

	Model classes	KLOC	View-Controller classes	KLOC
Stepstone libraries	26	5	63	16
Toolwriters library	31	2	68	10
Average tool	6	1	10	0.6

Praxis used Boehm's COCOMO productivity measurement model to measure the productivity performance in terms of the size to predict how many lines of code per day should be produced (Boehm, B.W., Software Engineering Economics, Prentice Hall, 1981). They considered the SSADM system to be a "semi-detached" system (in COCOMO terms) because it has distributed and multi-user aspects, and produced a COCOMO predicted benchmark of 11 lines of code per day. However, because the SSADM system used an object-oriented implementation, which differs markedly from a conventional implementation as assumed in COCOMO, it did not seem appropriate to use the COCOMO benchmark. In [2] Rawlings refers to a factor of 3.7 times as many lines for a C program compared to an Objective C program implementing the same small problem (citing Brad Cox example in Object Oriented Programming: An Evolutionary Approach).

Therefore, the project used another statistical measurement technique called "function point analysis" to assess the expected size and productivity (Albrecht, A.J. and J.E. Gaffney, "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation", IEE Transactions on Software Engineering, Vol. SE-9, No.6, November 1983, pp. 639-648). Rawlings in [2] indicates the following assumptions were made: each top-level operation in the Z specification is equivalent to an external input (the simplest function type); the logical internal files are ignored; the operations are of average complexity; and a neutral general application adjustment. Based on these assumptions, a function point count of 1120 is expected, which according to Albrecht and Gaffney would lead to an estimate of 48,000 work-hours (6400 work-days).

1.3 Evaluation

1.3.1 Background

Product:
A Computer Aided Systems Engineering toolset for SSADM

Industrial Process:
Contract development per requirements/design previously developed by client

Scale:Complete, integrated toolset (some 60 tools); 37,000 lines of Objective C, and 350 pages of Z (schemas plus English annotations)

Formal Methods Process:
Short requirements dialogue, followed by formal specification of the infrastructure and tools in Z, then implementation in Objective C

Major Results:
1. Toolset delivered on schedule and to requirements;
2. Productivity enhanced in part due to use of formal methods and to use of object-oriented programming

Interview Profile:
1/2 day in combined interviews with members of Praxis management and the development team

1.3.2 Product Features

Client satisfaction	n/a	(1)
Cost	n/a	(2)
Impact of product	n/a	(3)
Quality	+	(4)
Time to market	n/a	(5)

(1) Owing to factors outside the project, which had nothing to do with the formal methods, the product was not used. Therefore, it is not possible to assess client satisfaction.

(2) Praxis claimed the development cost was within budget, although delivery schedules during the two years the project slipped slightly. However, no data on product costs were collected which makes it impossible to compare the impact of the development on cost.

(3) Since the product has not been used, it is not possible to judge its impact.

(4) Few errors were discovered in reviews and testing, although it proved difficult to adduce this exclusively to formal methods since object-oriented programming and good people could have played an equally positive role. Whatever the cause, the product was acceptable in that it met the stated quality requirements.

(5) Time to market was not an objective of this product and hence it is not applicable.

1.3.3 Process Features

General Process Effects

Cost	n/a	(6)
Impact of Process	+	(7)
Pedagogical	+	(8)
Tools	-	(9)

Specific Process Effects

Design	+	(10)
Reusable components	n/a	(11)
Maintainability	n/a	(12)
Requirements capture	n/a	(13)
V&V	+	(14)

(6) Cost data was not broken down and, therefore, it was not possible to assess the cost effects of the formal methods on process. In our view, the use of the formal method contributed to achieving control of the system development, and that this is an important positive result of the process, but it is not possible to relate this to an impact on the cost of the process.

(7) The impact of the use of formal specification improved communication among the team members (some of whom were not familiar with the SSADM or Z methods) and between Praxis and the client, although it was also noted that early in the project the Z notation was an impediment to communication with the client.

(8) This was the first time Praxis had used formal methods on a complete development (earlier efforts had been formal specifications or studies). As such, a number of useful lessons regarding the use of a formal method on a development were learned. One was the experience in using a formal method in conjunction with another approach (object-oriented programming). A second lesson was the relatively easier (and unexpected) job of implementing the code from the formal specification. A third was the relatively difficult task of finding adequate metrics that accommodated

formal methods and object-oriented approaches.

(9) At the time, Z tools were very primitive. The tools used on the project were essentially untried, not robust, and inadequate to the needs. No attempt to improve them was made during the project.

(10) While an initial requirements document had been produced informally by the client, the design of the infrastructure and tools was not feasible from that document. The formal specification effort sharpened the design by helping the Praxis team understand the requirements and control the design based on that understanding.

(11) Reuse of either newly created components, or components already existing, was not an objective of the project and not undertaken.

(12) The product was not put into use by the client. Hence, no data is available on maintenance or maintainability.

(13) Insufficient information,

(14) Praxis told us that testing was assisted positively by having the formal specification but we did not explore this in great detail. The claim is that the formal specification was used as a reference in constructing test suites.

1.3.4 Observations

With respect to the claims for better productivity, we have collected the following data from Praxis in support of these claims.

COCOMO predicts approximately 11 lines of code per day for this class of system; Praxis figures show 17 lines of code per day were produced. The next figure shows a comparison of effort (in percent) expended per phase on the CASE project and a typical project using a conventional method.

Phase	Z Method %	Conventional %
Requirements	6	6
Spec and Arch	27	0
Design	0	10
Development	53	62
Integration and Test	14	22

It is worth noting that the conventional methods are slightly more productive for the middle three phases than the formal method, although this conclusion masks somewhat the effect that the effort expended in the Specification and Architecture phase had on the final phase.

The smaller amount of effort expended using Z on the more expensive phase of "Integration and Testing" reflects the quality of the specification effort earlier in the process. On balance, however, the data collected thus far on this project does not make a clear case one way or the other for Praxis' claim of increased productivity. If anything, the overall productivity in terms of level of effort seems to be practically the same.

Praxis believes the data are interesting in their own right. In our view, the COCOMO data is relatively meaningless, largely because it requires comparative data from other projects with which to compare results and that this data needs to be collected over a period of time on a number of projects in order to have a baseline. It is perhaps valid to note that the use of COCOMO shows that a formal method like Z can be matched to conventional productivity metrics if desired, although as Rawlings notes in [2] care on making assumptions of equivalence between new and different techniques (in this case object-oriented methods) and conventional ones must be taken. Praxis acknowledged that the only reason the COCOMO data was collected was to satisfy the client's management system which required an accounting of how many lines of code per day was produced.

Given the difficulty with the COCOMO data, the use of another technique that is more fairly matched to the techniques seemed sensible. Again, comparative data is necessary to develop a full picture of productivity. Absent this, about all one can do is note the data that Praxis developed using the function point analysis method.

1.3.5 Formal Methods R & D Summary

1.3.5.1 Methods

Specification

The Z method was used to develop a formal abstract specification of the system based on requirements supplied by the client. The features of Z most exploited in this project were precise expression of functions and operations in modelling the requirements in order to improve design, and the specification also served as a more precise communication medium than an English text.

Design and Implementation

The formal Z specification provided a clearer path to implementing the code, and implementation proceeded in a less troublesome manner than expected.

Validation and Verification

V & V per se was not performed. However, system testing was carried out, and the Z specification provided assistance in carrying out this testing. More work on using formal specifications to generate test cases seems warranted by this experience.

1.3.5.2 Tools

Language Processors

Primitive, prototype parser and type checker were the only tools used and the type checker was barely adequate. Problems with the tools were a less than desirable screen-based editor (based on the QED screen-based editor) which was changed to an "ascii" editor which did not require using QED for more flexibility, a lack of a tool that would expand and merge Z schemas, and other editing, browsing and cross-referencing features. It was noted that a way to partially type check an early, incomplete specification would have been desirable.

Automated Reasoning

No proving or proof checking was carried out or contemplated in this project. This was simply an effort in which formalizing the specification was seen to be sufficient. It was noted that the use of an object-oriented language for implementation narrowed the gap between specification and implementation, and hence, refinement in the sense of theorem proving (either by hand or mechanically) was not seen as necessary, and an informal relationship between specification and code was considered sufficient.

Other Tools

No other tools were considered necessary. However, based on this experience Praxis personnel have identified the need for additional tools (see above).

1.4 Conclusions

The SSADM project is one of the few projects we examined that combined a formal method with another informal approach (object-oriented). It illustrates that this can be done without excessive increases in development costs or delays in schedules, at least to the extent where a formal method is used for an abstract specification that helps clarify and understand difficult requirements.

It also is one of the few projects that attempted to keep measurement data, and illustrates the difficulties in trying to apply metrics intended for conventional software development techniques to new and substantially different methods. It points to the need for more research on this topic, as clients and companies are likely to continue to require metrics for project management.

The lack of adequate tools for the formal method did not hinder the project significantly, although it is clear some basic capabilities for editing and typechecking were needed and desired. These basic needs have been met subsequent to the completion of this project, since there are few reasonably robust typecheckers and editors currently on the market.

2 IBM'S CUSTOMER INFORMATION CONTROL SYSTEM

2.1 Case Description

IBM's Customer Information Control System (CICS) is a large (slightly more than 500,000 lines of mixed language code), on-line transaction processing program which runs on some 30,000 installations around the world (in banks, insurance companies, manufacturing firms, airlines, and many others). It generated about $2.3 billion in revenue for IBM in 1989. It was first developed in 1968 and is written in System 370 Assembler and PLAS (a high-level internal IBM programming language).

IBM Hursley Labs Ltd is a subsidiary of IBM UK Ltd., an IBM holding company for all IBM UK operations. It is a product development laboratory and is run as a profit/loss centre.

In 1981, IBM Hursley began to investigate new ways to improve future releases of CICS. IBM-Hursley has responsibility for all aspects of CICS, including the development and upgrade of future releases. As part of a general effort within IBM to introduce software engineering techniques into their software development process, in an attempt to deliver defect-free software, IBM Hursley began a contract with the Programming Research Group (PRG) at nearby Oxford University. PRG is the group which developed the Z method [1]. Beginning with the Software Engineering Workshops (an internal IBM training series for software developers), the PRG team provided education in formal, mathematically based methods to the IBM software developers, both through lectures and small projects using the Z method to specify small modules. This initial effort at education lasted through 1984-85. Z was chosen in 1985 to be used to specify several modules of the next commercial release of CICS.

The issues of interest to this study in the Hursley formal methods experience are threefold. First, we are interested in IBM's experiences in integrating the formal method into the general software development process at Hursley (including education and training experience). Second, we want to attempt to understand more completely the cost-benefit data which has been anecdotally reported over the years that this project has been running. Lastly, we are interested in the IBM experience in using a formal method for re-engineering parts of an old and evolving system.

2.1.1 Product and Process Profile

Size: CICS is several hundred thousand lines of code. CICS/ESA 3 Release 1 involved some 268,000 of new and modified lines of code, of which 37,000 were completely specified using Z, and about 11,000 were partially specified using Z.

Stages: 1979-81 Explorations on abstraction and encapsulation
1981-3 Restructure CICS decision; further explorations on Z and Dijkstra's Guarded Command Language
1983 Decision to use Z taken
1984-5 Z use on CICS/OS/VS in earnest; pilot studies
1989 CICS/ESA Version 3 Release 1 shipped
1990 Release 1.1 shipped
1991 Release 2 shipped

2.1.2 References

1. Spivey, J.M., The Z Notation: A Reference Manual, Prentice-Hall, 1989.

2. Collins, B.P., Nicholls, J.E. and Sorensen, I.H., "Introducing Formal Methods: the CICS Experience with Z", IBM Technical Report TR 12.260, December 1987.

3. Nix, C.J. and Collins, B.P., "The Use of Software Engineering, including the Z Notation, in the Development of CICS", Quality Assurance, Vol.14, No.3, pp. 103-110, September, 1988.

4. Phillips, Mark, "CICS/ESA 3.1 Experiences", Z User Group Meeting Minutes, Springer Verlag, September 1990.

5. Hayes, Ian, "Applying Formal Specification to Software Development in Industry", IEEE Transactions on Software Engineering, Vol. SE-11, No.2, February 1985.

6. Normington, Glyn, "Clean/Z", internal IBM paper, 3 April, 1992.

7. Houston, Iain, and King, Steve, "CICS Project Report: Experiences and Results from the Use of Z", VDM '91: Formal Software Development Methods, vol. 551, pp. 588-96, Lecture Notes in Computer Science, Springer-Verlag, December 1991.

8. Coleman, M.J. and J. Allan, Automated Quality Tracking, IEE Software Quality Assurance, May 1989, pp. 149-154.

9. Interviews with personnel at IBM Hursley and Programming Research Group, Oxford University.

2.2 Questionnaire 1

International Study on Industrial Experience with Formal Methods

Project: CICS
Respondent: John Wordsworth, IBM Hursley Lab
Date: April, 1992
Subject: Responses to Questionnaire 1

1. The organization develops the CICS (Customer Information Control System) products for sale under license to IBM customers worldwide. The goals of the organization are to be (remain) the market leader for transaction processing systems. There is a continual search for reduced cost and improved quality in our transaction processing products.

   CICS is a transaction processing system that allows customers to write applications to run in a CICS-provided environment. That environment supports file and data base access, pseudo-conversational transaction processing, terminal management, function shipping, transaction routing, recovery and integrity of data over connected systems, etc.

2. The project began in 1981, and its aim was to apply mathematics to the development of CICS/OS/VS so as to improve quality and reduce development cost. Certain parts of the product were in need of re-development, and these were decomposed into a number of domains (abstract data objects) to provide services (dispatching, program loading, monitoring, statistics gathering, status saving and restoring, etc.) to other parts of the product. It was assisted by a research contract with Oxford University Computing Laboratory Programming Research Group that has been renewed many times. At the end of 1983, a decision to use Z for specifying the domains was taken, and general Z education was undertaken in 1984. Pilot studies were made to develop processes, procedures and tools to support the restructure of the part of CICS being redeveloped. The new material became part of a successor product to CICS/OS/VS called CICS/ESA Version 3. Release 1 was shipped in 1989, Release 1.1 in 1990 and Release 2 in 1991. Some field experience of the use of the product has therefore been gained, as well as experience of developing a new function on a well-specified base.

3. The product was developed because a policy of continual development and improvement had rendered certain parts of CICS difficult to understand and difficult to change. Impending requirements for support of improved hardware (multi-processing, etc.) meant that control had to be taken of the difficult parts.

4. The development of the domains as abstract data objects would have been impossible without formal methods. Z was chosen for recording specifications.

Dijkstra's guarded command language was chosen for recording algorithms. These were chosen because they were powerful and expressive ways of recording, and because Oxford is only 60 miles from Hursley. The initial impulse to use Oxford as consultant arose from a talk given by Tony Hoare and heard by the CICS product manager at an Association for Computing Machinery (ACM) chapter meeting.

5. Z was used for writing specifications of the domains, and the guarded command language for recording the algorithms to implement the operations. Tools for editing, printing and cross-referencing the mathematical notation were developed. They have since been enhanced to include syntax, scope and type checking.

6. Advantages include quality improvement in delivered code, confidence of the developers in the work they were doing, and improvement in development productivity in later releases. However, some people didn't like it. Error rates in inspections suggest that many errors are found early with formal methods, and that the later stages of coding proceed faster than expected.

7. There are several technical reports, technical examples, conferences papers, etc., that report and explain this work and the techniques used in it. Other uses of Z in the CICS/ESA product area have been reported in various technical papers.

2.3 Interview Summary

Date of interview:	May 11, 1992
Organization:	IBM Hursley Laboratory, UK; CICS/ESA, Product Division
Respondents:	Peter Collins, Peter Lupton, Paul Munday, Glyn Normington, and (tools demo only) Jonathon Hoare
Interviewers:	Dan Craigen and Ted Ralston (author)

2.3.1 Organizational Context

IBM Hursley Labs Ltd is a subsidiary of IBM UK Ltd., an IBM holding company for all IBM UK operations. Hursley is a product development lab and is run as a profit/loss centre. There is a general Director to whom the various product managers report.

Within the CICS Product Division there are several groups: a group concerned with the overall product development (ESA), and several groups responsible for the various hardware platform implementations of CICS. ESA is the largest of the groups and consists of some 400 people.

The organization develops the CICS (Customer Information Control System) products for sale under license to IBM customers worldwide. The goals of the organization are to be (remain) the market leader for transaction processing systems. There is a continual search for reduced cost and improved quality in their transaction processing products.

CICS is a transaction processing system that allows customers to write applications to run in a CICS-provided environment. That environment supports file and data base access, pseudo-conversational transaction processing, terminal management, function shipping, transaction routing, recovery and integrity of data over connected systems, etc.

2.3.2 Project Context and History

History: Business development responsibility for CICS was moved to Hursley in 1974-75 from IBM Palo Alto at a time when another project was seen by IBM to be the wave of the future in transaction processing. CICS was then seen to be declining in importance.

In 1981, personnel at Hursley began to investigate new ways to improve future releases of CICS. IBM Hursley has responsibility for all aspects of CICS, including the development and upgrade of future releases. Some 400 of the 1500 software engineers on Hursley's staff are employed on CICS.

As part of a general effort within IBM at that time to introduce software engineering techniques into the IBM software development process in an attempt to deliver defect-free software, Hursley began a contract with the Programming Research Group at Oxford University, the group which developed the Z method. Beginning with the Software

Engineering Workshops (an internal IBM training series for software developers), the Programming Research Group provided education in formal, mathematically based methods to the IBM software developers, both through lectures and small projects using the Z method to specify small modules. This initial effort at education lasted through 1984-85. Z was chosen in 1985 to be used to specify several modules of the next commercial releases of CICS (CICS/ESA Version 3, Release 1, shipped in 1989, followed by Release 1.1 in 1990, and Release 2 in 1991)

Versions: Release 3.1 consisted of 268,000 lines of new and modified code, of which Z was used to specify some 37,000 lines fully, and another 11,000 lines partially. Release 3.2 consists of approximately 50,000 lines of code specified in Z.

Follow-on Projects: In addition to the continued use of Z on additional CICS modules, such as the Application Programmers Interface, Hursley has begun a small project to link Z with IBM's Cleanroom method (called Clean/Z). The project combines elements of the Cleanroom approach, using Z in place of box-structures for the formal specification, and elements of Knuth's "literate programming" style for design documentation.

2.3.3 Application Goals

CICS is an important revenue generator for IBM, producing over $2 billion in revenue annually. Ten years ago, CICS was not such an important product as today, although it was not considered trivial either. Time to market was a sort of goal but not primary, mainly because IBM had an installed base and CICS was a product that customers had come to expect at a regular, not time critical, schedule of releases. Cost reduction and improving quality were major drivers at the beginning. Midway through the project (c. 1985) productivity became an important goal in terms of future functionality. Reliability was and is an important goal since it is too costly to do a lot of field repair.

Improving the quality of the application was an important product goal. To achieve it, the development team indicated they needed intellectual control over the code, which they believed a formal specification effort would give them. The marketing and sales force was a key group to be convinced that improving quality and adding more functionality to CICS would make CICS a more saleable product. Marketing and sales were generally of the view that CICS was more or less fine the way it was and that it only needed some restructuring, rather than improving quality and adding new features. However, this resistance was overcome and clients were impressed by IBM's drive for improved quality. The recent release of CICS 3.1 has been very successful, having been installed in roughly half the CICS sites worldwide.

One of the main points in convincing the marketing staff was that the effort to improve quality would lead to a reduction in costs, and this became one of the major drivers behind the formal specification effort. Problem reduction is the aim of productivity, where productivity is measured over the lifecycle of the product.

A common process model is followed at Hursley (and IBM worldwide) for software product development. It is called Product Process Architecture (PPA) and consists of a number of "stages": requirements, specification, high level design, low level design, code, testing, and delivery. The PPA terms IBM uses for these stages are: Requirements; Product Level Design (PLD); Component Level Design (CLD); Module Level Design (MLD); Code; Unit Test; Functional Verification Test; Product Verification Test; System Verification Test; Package and Release; Early Support Program; and General Availability. Of note, the PLD stage was introduced as a change to the PPA in order to accommodate the production of a formal specification. PPA was not in place at the start of the CICS/ESA use of Z in 1982 but went into effect shortly thereafter. At the start, Hursley followed the Design Reviews process, then current in IBM.

2.3.4 Formal Methods Factors

The use of Z on CICS resulted from several influences but was mainly bottom up, i.e., it came from the development team. The CICS/ESA team, which in 1982 was relatively small (about 10 people), had been having trouble and foresaw continuing problems, with increasing functionality and quality largely because they did not know what CICS really was. The specification at that time was written in pseudo-code, was unmanageably large, and was poorly documented. The CICS code itself was a hodge-podge of modules with patches in various languages that dated back to the late 1960s). At the time (c. 1982), the team was exploring the use of abstract data types, encapsulation, and the like, because the pseudo-code they had was imprecise and ambiguous, and they were unable to document the state of the system or the pre-conditions.

At the start of the project, design reviews were basically conventional inspections, which were common in the early 80s, but these were soon replaced with the PPA process. Reviews became both informal and formal parts of the process. The team would discuss parts of the spec both before formal design reviews and during the formal reviews. In the view of the developers, Z helped this communication, particularly in comparison to previous experiences. Z helped in two ways: the underlying maths notation was much better than the pseudo-code they had had before; and Z influenced the clarity of the English annotations. The pseudo-code ignored details; it just gave an approximation (WHILE/UNTIL loop example was given and the fact that pseudo-code did not document state or pre-conditions) and defects could be detected only while doing an inspection, once the code had been written. As a result, Program Level Designs are now used for all other modules, even those not using Z, and the Z Program Level Designs help uncover problems earlier.

The decision to use formal methods modified the Product Process Architecture by adding the PLD level and a new formal document. This formal document included both Z and English (as is usual in Z specifications). Z was used at Program Level Design, Component Level Design and Module Level Design stages. The Guarded Command Language was

used at Component Level Design and Module Level Design.

The pressure to change the development process resulted from grass roots displeasure with the loss of intellectual control. This displeasure percolated up to CICS management; a serendipitous talk by Hoare at Oxford resulted in contact between Hursley and the Programming Research Group. The first contracts were between Hursley Technical Planning Group and Oxford. These then migrated to the CICS development team and Oxford.

As they started considering the use of formal methods, there was a choice between Z and CDL. Oxford's work on developing Z specifications for a part of the Application Programming Interface helped to convince Hursley to use Z. Composability (using the schema calculus) was viewed as positive and certainly helped in choosing Z over VDM.

2.3.5 Formal Methods and Tool Usage

CICS/ESA 3.1 had 268,000 lines of new or rewritten code. Of this, about 50,000 lines could be traced to Z specifications. Typically, a CICS release is around 100,000 lines of code. Hence, this release was much larger and was the first time that there had been a significant rewrite. They rewrote low level domains (e.g., storage manager, dispatcher; i.e., the basic services). It was not at the level of resource management (e.g., database or file). This is a bottom-up process and it seems that some of the current work is now at the resource management level. In their rewrite, they re-engineered existing code and documentation and used the ensuing Z specifications to implement new code.

The interface between the new and old code has been a source of a number of problems. There have also been problems with interfacing with the underlying operating systems.

Maturity: As of the start of the project, the formal methods process was not mature; now, they feel it is. It is still immature with respect to concurrency, and there is no formal approach to concurrency with which Hursley is happy. There are no real time obligations in CICS, except for delaying execution. The process was certainly mature beforehand. While the insertion of formal methods into an already mature process led to some increased immaturity, the CICS developers felt the risks were reduced.

Risks: There exists the usual fall-back position: go back to the conventional approach. Hence, the risk of using formal methods was considered by Hursley to be minimal.

Scalability: There were no previous example of systems of the same size being handled with formal methods. However, scalability issues were of importance in the choice of Z (with the schema calculus).

Readability/Reviewability: In the view of the developers, a good Z specification plus explanatory English text is the best specification one could have. A bad Z specification

could be worse than English. The group felt that Z was superior to VDM, SEDL, etc., with respect to readability.

The tools evolved as follows and were produced by the CICS development team.

Pencil and paper and eraser
|
Editing and printing of Z documents
|
Cross-reference tool
|
PS/2 based tool

The first three levels did not have automatic semantic analysis.

The tools were very cheap to develop, easy to use, and robust. The tools were simple in their functionality and implementation. None of the tools were formally specified.

The tools were independent of the LCS (Library Control System). LCS performed configuration management, etc., for the code. IBM is considering adding text to LCS and thereby including the Z specifications in LCS. LCS was built to handle unstructured code.

2.3.6 Results

The major claim by IBM on the efficacy of using Z to specify major parts of CICS is found in the paper by Mark Phillips, and repeated by other IBM personnel.

> "The initial conclusions from using the Z notation in the development of CICS/ESA 3.1 were that Z specifications are appropriate both in new components and where large changes involve rewrites. Measurements indicate quality improvement of CICS/ESA 3.1 throughout the development process. There is also an improvement in development costs, and the number of errors detected was reduced. The initial development benefit has been assessed to be a 9% improvement in the total development cost of the release, as opposed to developing the 37,000 lines without Z specifications. The assessment of cost benefit was based on the reduction of programmer days fixing problems (italics in original)".

Further, during the interview Hursley personnel indicated that the number of customer reported faults had been reduced by 60% in the Z specified code compared to the non-Z specified code. They attributed this improvement to the more detailed reviews required by the formal specification effort which uncovered design errors earlier.

In addition, several quality assurance and process claims have also been made, again principally by IBM personnel [2]. The quality claims centre around defect detection,

elimination, root cause removal, and quality prediction through creating defect removal models [8]. Attempts to use these latter defect removal models were not successful, because of difficulties in calibrating them for formal methods. There are also some qualitative claims made about improvement of the development process from the point of view of the managers and developers impressions.

Figure 1 lists some of the key metrics used by IBM Hursley in measuring the efficacy of Z in developing the latest CICS release. These metrics were used as a standard part of the IBM development process, although not all the product and process metrics used by IBM are listed here.

- General
  - Developer or tester person hours
  - Thousands of lines of new or changed source
  - Thousands of lines of shipped source
  - Line item descriptive name or identifier

- Inspections
  - Preparation time for each inspector
  - Inspection time for each inspection
  - Total rework time(resolve and repair)
  - Duration of inspection meeting
  - Defects recorded, coded by severity and cause
  - Notation and languages of the design material

- Testing
  - Test cases prepared, attempted success/fail
  - Defects found, by severity and functional area
  - When fixed, cause and earliest detection stage
  - Customer implementation
  - Number of licenses
  - Number of apparent defects
  - Defects in CICS code or documentation
  - Hours spent by IBM in maintenance and defect removal/repair

Figure 1 Metrics

2.4 Evaluation

2.4.1 Background

Product:

IBM's Customer Information Control System, a large transaction processing system installed at 30,000 locations worldwide.

Industrial Process:

Re-engineering of a number of modules of the CICS software, and addition of new features.

Scale:

CICS is approximately 500,000 lines of source code; Z was used to completely specify 37,000 lines of code and to partially specify a further 11,000 lines of code.

Formal Methods Process:

A formal specification using the Z method introduced into an established software engineering process.

Major Results:

9% reduction in total development cost; 60% reduction in errors after coding; successful integration of formal method into an established software engineering process.

Interview Profile:

1 day of interviews with CICS Product Division personnel, plus tool demo.

2.4.2 Product Features

Client satisfaction	n/a	(1)
Cost	n/a	(2)
Impact of product	+	(3)
Quality	+	(4)
Time to market	n/a	(5)

(1) Not enough evidence was collected to make a definitive judgment. The client in this case is the CICS end-user. The only direct evidence collected regarding client satisfaction was the 60% reduction in number of customer reported errors that require correction, which by itself would indicate a positive impact. Indirect evidence also indicates CICS users are satisfied with the new release, as they have installed it and continue to use it. This should be weighed against the observation that the customers are somewhat "captive" since CICS is an application product matched to IBM hardware (and consequently might have little choice), although

CICS is portable to other platforms.

(2) The 9% reduction in total development cost coupled with the reduction in customer reported faults supports the contention that the early defect discovery and correction helped achieve a reduction in costs. However, without further data (such as previous development costs) with which to compare the 9% reduction claim, or evidence showing the connection between the fewer customer reported faults and the 9% reduction, it is not possible to assess the impact the use of Z had on the cost of the product.

(3) CICS is considered by Hursley to be an important product and a major revenue source for IBM. The fact that IBM still offers, updates, and supports CICS after so many years of service would also seem to indicate IBM's product decision-makers believe this as well. However, no independent evidence was collected to support this view.

(4) Quality improvements were observed with respect to defect reduction and to the process used by IBM to develop CICS. While other factors may have also contributed to this quality improvement, the use of the formal specification definitely contributed to solving several quality problems, such as bringing the documentation and code under control and improving design reviews. Reliability, as a constituent element of quality was also improved, as indicated by the 60% reduction of customer reported faults.

(5) While even Hursley personnel admitted they are not renowned for speedy delivery, it should be noted the majority of the products they produce, including CICS, are not time-to-market sensitive. CICS is an old, established application package for an installed base and is on an evolutionary path, with regular releases every couple of years. Consequently, time to market was not an applicable measure in this context.

2.4.3 Process Features

General Process Effects

Cost	0	(6)
Impact of process	+	(7)
Pedagogical	+	(8)
Tools	+	(9)

(6) While no figures were collected, significant "start-up" costs were incurred at the beginning in education and training, including the contract with Oxford and the development of an internal course for the Software Engineering Workshops. In addition, the time the software developers spent away from their main duties while they learned the new method must be considered. On the other hand, the positive

improvement of the process and the product must be weighed, as should the fact that introduction of any new method or technique incurs start-up costs. On balance, it is our view that the impact on the process in terms of cost was neutral.

(7) The PPA process was reinforced by the addition of the formal specification (PLD) stage by means of requiring earlier in-depth reviews.

(8) A number of important lessons were learned over the years this project has been running.

(9) Starting with "pencil and paper" and developing tools as needed meant the tools were a good fit to the process. The fact that the tools are easy to use and robust has also helped the effort in using Z within the CICS/ESA organization.

Specific Process Features

Design	+	(10)
Reusable Components	n/a	(11)
Maintainability	+	(12)
Requirements Capture	+	(13)
V & V	+	(14)

(10) The formal method had a positive impact on the design in the sense that the formal specification process concentrated attention on both high-level and low-level decisions, earlier than at the implementation stage. Also, using abstract data types to represent the CICS basic services, such as the storage manager and the dispatcher, was a first for CICS and was facilitated by using Z. The pseudo-code they were using previously did not support Abstract Data Types (ADTs). Through abstraction and documenting the state of the system, Z also made reviewing easier than with the pseudo-code.

(11) While some reuse of existing components of CICS was involved, in the sense that functionality in old modules was re-engineered, we did not address this topic fully enough to make a judgment.

(12) CICS/ESA Version 3 is in use in over 2000 sites worldwide, and its reliability has been the subject of favourable comment by the users. This is also supported by the 60 percent reduction in customer reported faults, from which one can infer that maintenance of CICS/ESA Version 3 has improved.

(13) Given that this project involved a re-engineering of an existing product, as well as the addition of new functionality, the formal specification effort had a positive impact on gaining intellectual control and understanding of the requirements either as they were expressed in the original documentation or, in the case of new features,

de novo. The formal method was indirectly useful in negotiating with the CICS marketing personnel by giving confidence to the developers they could add new features and maintain high quality and not run up the development costs.

(14) Unit, function, system and product testing were performed on CICS/ESA 3.1 and 3.2. in accordance with the PPA process. The use of formal methods has no real impact on system testing. On function testing, Z was of some value. Most of this testing is black box testing, but there is some white box testing as well. The main influence of Z occurred at the unit testing level. Hursley built a new testing environment which allowed for the testing of new domains (specified in Z) and they could plug in prototype domains. There is no statistical testing in the current CICS process and if the CICS process is related to Cleanroom they still will not drop function and unit testing. We judged this impact to be positive because the unit testing allowed more rapid domain testing through prototyping.

2.4.4 Observations

As a case study of industrial experience, the CICS case is one of the best known, longest running projects in which a formal method has been used on a real industrial software product. A reasonable amount of information is available through private interviews and a few published papers and technical reports, enough to conclude that the effort expended by IBM's Hursley Laboratory:

- involved a real product with demonstrable commercial pay-off and importance to IBM,
- has been focused on both re-specifying old code and specifying new code,
- produced satisfactory results that permitted the release of the latest CICS product (CICS/ESA 3.1), and
- Z is continuing to be used in the development of subsequent releases of CICS (e.g., Release 3.2).

At the same time, the published facts do not as yet provide enough data to conclude definitively that the productivity and quality assurance claims are justified or meaningful. Rather, they suggest a modest productivity improvement and an ability to produce fewer errors, detect errors earlier in the process, and find deeper errors that could go undiscovered until after delivery to the customer. It is difficult to assess the significance of the productivity improvement claim since other productivity data from earlier CICS developments (against which IBM claims to have measured this latest release) have not been provided for comparison. Similarly, some data is provided on previous error detection experiences with earlier releases. However, the most one can do is to infer that IBM had this interest in using a formal specification method because of a real problem.

Given the claims regarding reduction in total cost of development and reduction of customer reported faults, it still remains necessary to obtain more quantitative comparative

data or authoritative corroboration from IBM before a definitive judgment can be made. In part this is because of IBM's understandable hesitancy to disclose what could be confidential information, and in part because the people involved in the CICS project are programmers and software developers who do not as a rule publish. It is also difficult in advance to assess the effects of a radical change in technology, and to develop metrics adequate to measuring them.

It appears that the use of Z or other formal methods has not spread to other parts of IBM. The Clean/Z project mentioned above is a very small effort at this point, and it is too early to tell whether it will in fact turn into anything meaningful. There are questions as to how much of the Cleanroom method is in fact used in Clean/Z, since statistical testing is not used, and the Knuth "literate programming" style is not a part of Cleanroom. Simply substituting Z schemas for box structures does not mean that the full Cleanroom method has been adopted.

2.4.5 Formal Methods R & D Summary

2.4.5.1 Methods

Specification

The Z method was used exclusively to specify the "basic services" features of CICS and the Application Programmer Interface, rather than the resource management services. CICS has a few modules that have timing and locking constraints (mainly execution delays, but Z was not used on this part of CICS, in part because the developers knew Z could not handle these properties. Also, Z schemas are basically atomic and not modular (e.g., no global definitions or scoping ability) which is seen as a potential problem as systems get larger.

Design and Implementation

Refinement was not attempted, in large part because the CICS developers believe that proof would be too time consuming. The resulting errors have primarily been coding problems, not design problems. One emerging problem, which is not necessarily specific to Z or indeed formal methods, is keeping the specification current with the code. This has been observed in other cases in this study.

Validation and Verification

No clear issues emerged from this case as the Z specification seemed to have very little impact on the Testing or Verification parts of the Product Process Architecture. With respect to Unit Testing, it was noted that symbolic execution could help convince others that functions modeled in Z worked but that the cost of achieving symbolic execution might be too high.

2.4.5.2 Tools

Language Processors

An automated semantic analysis capability was noted as desirable (once a Z semantics is finalized).

Other Tools

An interface to a configuration manager, or a Z specific configuration manager, would be useful.

2.4.5.3 Recommendations

Specification

Although CICS has minimal timing constraints, Hursley personnel indicated a belief that Z was not able to handle concurrency or other timing considerations adequately. As observed in some of the other cases using Z, this deficiency should be addressed.

Design and Implementation

Hursley did not use refinement to implement CICS. They noted that, at the time, refinement by using proofs would have taken too much time. In their view, refinement could be useful and more research is needed, especially to understand the path to lower levels, which is one of the goals of the Clean/Z project.

Tools

Hursley is happy with the status of their Z tool and does not see any reason for more capability at this time. They noted that development of a pre-condition calculator would be useful.

2.5 Conclusions

The CICS project has used a formal method successfully to improve a well-established "bread and butter" product for IBM. Primarily, this means the formal method has been used with success to gain intellectual control of the system. "Gaining intellectual control" means that aspects such as unmanageable documentation, a lack of understanding of what all the code did, and other such issues that plague old, "dusty deck" products, have been brought under the control of the team responsible for it, through the effort to re-engineer important existing modules as well as adding new functionality. The use of an abstract specification helped achieve this control by helping to clearly identify difficult aspects of the system and to facilitate better reviews. This is important in terms of both the improvement of the product and the marketing of CICS.

The CICS project is one of the few formal methods efforts we examined that has extended over a relatively long period (a decade). As such, it has passed through several important

stages -- feasibility study, education/training, experimentation, real development, extension to further releases -- all of which have contributed to acceptance and routinization of the method at least within the CICS/ESA group.

This latter observation is also important: except for a small effort on Clean/Z, use of the Z method has not spread outside of the CICS/ESA group to other parts of the organization or company. This failure to spread is clearly a negative aspect of this experience.

The CICS project has also taken an opportunistic approach to developing tools. This means the tools were not developed until a need for them arose (not beforehand as in some other projects). As such, the tools do what is expected of them, neither more nor less, and all things being equal, they are not an issue in the use of the formal method (i.e., concerns for their reliability or performance do not get in the way).

Given the reputation IBM has for measuring many aspects of the development process, the absence of published comparable data by which to compare the formal methods results on CICS with previous CICS development efforts using other methods is regrettable. However, it is also unfair to expect commercial companies to disclose proprietary data. Perhaps the absence also illustrates the difficulty that exists in finding adequate and acceptable metrics for new methods generally not just formal methods.

3 CLEANROOM SOFTWARE METHODOLOGY

3.1 Case Description

Cleanroom Methodology

This case includes interviews with two separate organizations, both using the same basic set of principles that embody the Cleanroom Methodology:

- A pipeline of increments
- Statistical quality control through usage testing and reliability models
- Mathematical verification in place of unit testing

Formal methods play an important role in the Cleanroom methodology, but the methodology has several parts that must work in concert to achieve the goals of high quality and productivity. Mathematics-based development and verification (with no unit debugging) are expected to yield such high quality software that a statistical testing process can be applied. Formal team reviews for correctness and simplicity are applied at each step, including stabilization of specifications before design. Appropriate mathematical structures for different parts of the system may include box structures, grammars, and logic. Natural language is used to explain the formal specifications. The basic formalism often used is function-theoretic, with considerable effort expended on intermediate functions that constitute the design decisions. Data structure discipline requires the use of only objects, such as stacks and queues, that simplify reasoning. Testing is based on usage profiles that are built up as part of the requirements process. For more complete explanations of the formal methods aspects of Cleanroom, see the references below.

IBM Cleanroom Software Technology Center and the COBOL/SF product

The product subject of this case is the IBM COBOL Structuring Facility (COBOL/SF). The case also investigates the more general experience with Cleanroom and its transfer within IBM. Briefly describing the product (from the respondent questionnaire #1):

COBOL/SF automatically transforms unstructured COBOL programs into structured form. The product is comparable in function and complexity to a modern high-level language compiler. It applies proprietary graph-theoretic algorithms to convert programs to an internal form suitable for application of the Structure Theorem constructive proof. The resulting mechanically-structured program is then transformed by rewrite rules into a structured equivalent that more resembles code a programmer would produce.

3.1.1 Product and Process Profile

Size: 80KLOC,
28KLOC reused from earlier versions, 18KLOC changed, and 34KLOC new.
Effort: Seventy person-months of effort spread over seven months in 1987, building the product in five increments.
Errors: 3.4/KLOC
Productivity: 740 lines/person-month
Stages:

Prototype:	Proof of concept.
Version 1:	First product, structured VS COBOL II only.
Version 1A:	First product upgrade, added new features and improved structuring.
Version 2:	Second product, added OS/VS COBOL structuring and many new features in a major re-architecting.

NASA Goddard and a Satellite Control System

NASA Goddard Space Flight Center (GSFC) operates a Software Engineering Laboratory (SEL), in cooperation with the University of Maryland and Computer Sciences Corporation. The SEL experiments with technology and methodology of interest for advancing the state of practice in development of GSFC software systems. A series of experiments is in progress for Cleanroom. SEL versions of Cleanroom are tailored for the GSFC environment and may involve different combinations and degree of application. Measurement of experiments is planned and helps guide the experimentation over years and projects. The case experience reported here is offered for comparison with the IBM Cleanroom approach, which is the major subject for the case study.

3.1.2 References

1. Richard C. Linger and Harlan D. Mills. "A Case Study in Cleanroom Software Engineering: the IBM COBOL Structuring Facility", CompSac, IEEE Computer Society, 1988.

2. Michael Dyer. "The Cleanroom Approach to Quality Software Development", Wiley Series in Software Engineering Practice, 1992.

3. R. Linger, H. Mills, and B. Witt. "Software Engineering Laboratory (SEL): Cleanroom Process Model", NASA Goddard Space Flight Center, SEL-91-004.

3.2 Interview Summary: IBM

IBM Cleanroom and the COBOL/SF Product

Date of Interview:	April 2, 1992, 2.5 hours
Organization:	IBM, Gaithersburg, Maryland
Respondents:	Richard C. Linger and Phil Hausler
Interviewers:	Dan Craigen and Susan Gerhart (author)

3.2.1 Organization and Project Context

Mission: Linger and Hausler manage the Cleanroom Software Technology Center (STC). The center is composed of 15 people and will be expanding in the near future. It was founded in 1990 as a result of earlier Cleanroom experiences, especially the COBOL/SF program. The center is chartered to transition Cleanroom technology to other divisions in IBM through consultation, education and tools. A general approach has been to assign an experienced Cleanroom practitioner to a development team and for that individual to consult with the project. They also use video and networks to keep in touch, and to help members from the first adopting teams as they work on successor projects.

Linger claimed that they now have a number of results where the software developed using Cleanroom improved the results (error rates) of the COBOL/SF project. Linger went on to say that he sees an easy order of magnitude reduction in error rates and feels they have a chance of making two orders of magnitude. (IBM has corporate goals to achieve 10X improvement this year and 100X by 1995.)

Other Cleanroom projects include teams making modifications to large systems, e.g., adding functionality, quality re-engineering, and reverse engineering problem modules. Phil Hausler, after working on COBOL/SF, moved to an AI project, with a large organization of 50, that developed a system of 107 KLOC.

The approach used to transition Cleanroom is as follows:

1. The labs identify a number of projects where Cleanroom could be used.
2. The relevant individuals are trained in Cleanroom and expert consultation is provided.
3. There is an attempt to have the individuals at the labs increase the use of Cleanroom. Hence, there is an explicit attempt to have the labs become self-sufficient in the technology.

Educational Backgrounds: There is a wide variation in the educational backgrounds of those being exposed to Cleanroom, from retrainees to Ph.Ds. In general, however, they are well educated with bachelors or masters degrees in computing science and mathematics and with industrial experiences of 5 to 10 years.

Hundreds of individuals within IBM have been trained in Cleanroom and there are a number of projects within IBM where Cleanroom is being, or to be, used. The goal is to add 5% to what the attendees already know. A curriculum of four courses is now taught, one of which uses the older Linger, Witt, Mills book.

Management Structure: The management structure is small teams: a first-line manager plus a half-dozen developers and testers. Any kind of management structure is viewed as acceptable for Cleanroom, but stress must be placed on teamwork. Several examples of how teams work include:

- People often change roles from testers to developers and vice versa.
- Typically, there is one architect on the team.
- If a change comes in, it goes through the affected team leader, but they also maintain a database with formal change tracking.

Incremental planning is a substantial part of the management, and must be agreed to by both teams (developers and testers) up front to certify against each increment.

There are two critical success factors: management commitment and support, and team motivation.

Process Maturity: With respect to the Software Engineering Institute Maturity Model, Linger felt that those projects using Cleanroom were certainly at Level 5. Feedback is provided from early experience into courses within projects. As well, there is feedback for process improvement as builds and reliability measures are used. The wider IBM process is typically waterfall, with lots of unit testing. It was noted that Cleanroom products usually result in a smaller amount of code than normal: i.e., higher functionality per line of code. There is no correlation between error rate and team size.

How Cleanroom came about: In the early 1970s, the Federal Systems Division needed someone to teach courses on structured programming and verification. Through the development of the course, Linger became more involved with Mills. Their collaboration led to the book [4] on correctness verification, and software analyses and design.

In the early 1980s, Cleanroom started coming together: unit testing and debugging were stopped, with verification taking their place. Instead, testing was to be used to certify quality, not to put quality in after the fact. About that time, the box structuring ideas also came along. They could not figure out how to scale up the axiomatic approach but functions do scale up. Mills had the theoretical basis but had to build up the practice.

The COBOL/SF Application

Objectives: The goal was to develop a COBOL structuring program that would be a companion product to the VS COBOL II compiler. The structuring program was regarded as a critical product. Resources were made available under severe time constraints.

Technical Challenges: This project broke new ground in many ways: it was the first structuring engine in IBM, it was an experiment with Cleanroom, and it was a prototype effort to convince IBM management.

The complexity of the COBOL semantics almost brought the project to a halt. COBOL was formalized using attribute grammars and denotational semantics. IBM Federal Systems Division had a number of proprietary graph theoretic algorithms for structuring flowchartable programs. Structuring COBOL was a formidable task. It was tough to get COBOL programs into flowchartable form, requiring significant graph theory (a Ph.D. mathematician was on the team).

An ANSI standard was used in the restructuring to help define OS/VS COBOL and VS COBOL II. IBM required the capability to restructure 100% (without that there would be no confidence in the product). The COBOL formal description was primarily expressed in attribute grammars. There were three subsystems, each with its own spec, which were 2-3 inches thick.

3.2.2 Formal Methods Factors

The criteria for choosing a formal method included referential transparency, completeness and closure. It is crucial to have the right theoretical underpinnings. No extensive search was made nor were other candidates evaluated: they accepted Mills' approach.

3.2.3 Formal Methods and Tool Usage

Specification techniques included grammars, transformations, conditional rules, and state machines.

Box structures at the highest level were uninteresting but the internal design used lots of state machines and common services. One change that the Cleanroom researchers would do now is to use full box structured notation. Box structures were still under development at the time of the project.

They used Program Design Language in the second major phase. The design is expressed as functions in Program Design Language, including conditional rules for intended functions.

Verification review used mental proofs based on the correctness theorem. No written proof of correctness was produced. Most Verification Conditions were checked in seconds, but some take hours. They found lots of reasoning patterns, e.g., queue data types. Hausler had doubts about ever finishing but claimed that people get very facile doing these Verification Conditions mentally.

Systematic data structures with regular properties had an important role in carving up the state space. The rule "functions first, performance second" was usually satisfactory.

Linger sums up with ``The heart and soul of Cleanroom are the verification reviews."

Testing: The test team has to determine the usage probability distribution, stratify the input for the incremental builds, and run the test cases. No knowledge of the internals is used in developing the test

cases. Usage testing is considered 20 times more effective than coverage testing. With usage testing, you get better and better testing results. Newer Markov models give a high number of cases for a few states. On one project, they captured baseline usage for a new device controller.

No coverage monitoring is used, although some alternate distributions are introduced for critical failures. Some self checking is applied but there remains the usual oracle problem: What is the right answer? Linger observed that if the testing has more than four errors per KLOC then one should take the code off the machine.

This project used formal methods in all aspects of the project. The heuristics used to decompose were smallest interfaces, with referential transparency being maintained. They were always guided by completeness and provability goals.

At the time, Cleanroom was a new technique. Nobody had built a structuring engine, hence a prototype was developed first. Only one other such product was on the market; afterwards, there were three or four competitors. There were other Cleanroom projects of significant scale, but this project was the first of this scale (80 KLOC).

Cleanroom generally has a relentless push for code reduction and simplification. The users try to reduce the size of the state space and look at getting the functionality correct; they review performance issues later.

In this case, the tools were all PC-based: a good text editor, certification modellers, PDL-to-PL/I translator, and test case generator (see the appendices of Dyer's book [2]). Today, they would use a smart editor. All tools were very simple and developed by the group. Other groups are customizing their own tools. There were no interfaces to other developmental tools.

Tool development was an auxiliary activity, not a goal in itself. For some humour, consider how well the one noted tool---the wastebasket---meets the objectives: cost-effective, robust, easy to learn and to use. Cleanroom is not a tool-based methodology.

They would like something to keep track of prime programs (e.g., a loop body), and navigation through and coverage of sets of correctness questions to facilitate the additional review of ideas and designs before verification. They used function-theoretic verification, which scales up, versus axiomatic (Hoare-style) which no one knew how to scale up at that time. Linger noted that they had spent a lot of time working on proof checking research. (None of this work is available outside of IBM.) Regarding automation, they were unsure how much change would have occurred if they had had theorem proving, because the other role of verification is communication within a team.

The team owns the product, errors, etc., that helps track change. Individuals are assigned tasks in the usual manner, but the team is responsible for review and ownership.

The kinds of errors found were not deep design, just typos, etc. A typical build will not run the first time, but when fixed it will run forever. Much less time (a few days) is spent on critical testing, so builds go faster. They did once pull an increment off the machine, took it back and re-designed.

3.2.4 Results

Without Cleanroom, they felt they would never have completed the COBOL/SF product. They faced hopeless complexity (COBOL semantics, lots of rules) but did produce a very reliable product in the

field. Linger claimed that COBOL/SF was recognized as an incredibly reliable product that resulted in the Cleanroom Software Technology Center.

Cleanroom has been used to find errors in operational code and has saved substantial resources.

It takes less than one person-year per year to maintain the system. True, there are fewer restructuring installations than compiler installations, but the restructuring runs for long amounts of time.

The team felt many effects of the Cleanroom experience. They were invigorated and geared toward 0-defect. That became part of their work ethic.

If they could do anything differently, they would use the luxury of up-front education, but there was nothing else substantive to change. They would use better software tools, but no substantial changes in the methodology.

Cleanroom continues to change. In research, they are working on an extension of box structures for particular types of systems; real-time extensions, dealing with concurrency (now being used in complex buses, etc. but the results can't be released); improved Markov statistical process; trade-offs in quality certification. In practice, they are learning that each team tailors for themselves. Cleanroom makes available a set of technologies and practices that can be customized and adapted to an organization's environment.

3.3 Interview Summary: NASA Goddard Center

NASA Goddard Cleanroom Experiments

Date of Interview:	April 2, 1992, 2.5 hours
Organization:	NASA Goddard Center, Greenbelt, Maryland
Respondents:	Scott Green (NASA) and Victor Basili (U. Maryland)
Interviewers:	Dan Craigen and Susan Gerhart (author)

3.3.1 Organization and Project Context

Software Engineering Laboratory: NASA Goddard Space Flight Center (GSFC) in Greenbelt, Maryland, builds satellite attitude determination and ground support systems. For 15 years, NASA GSFC has worked with the University of Maryland and a contractor as part of a Software Engineering Laboratory (SEL). The SEL experiments with new technologies and new processes, publishes reports, and conducts an annual Software Engineering Workshop.

Case studies in the SEL are conducted on-line (as the project is producing its product) along several themes: Cleanroom, Ada, and CASE.

The general model for the experiments is described in many papers by Professor Victor Basili on the concept of an "Experience Factory" with the goal of projecting and improving software development from an "experience base", e.g., costs/defects. For example, in attempting to answer questions such as `Should we cost a Cleanroom project in the same way as other projects?', they use these projects to evaluate technologies that they are considering bringing in-house.

Process: The normal NASA process follows the waterfall approach, using its own handbook. Projects tend to short-duration (one year) to satisfy specific satellite missions. A large base of software has been developed that drives future applications. This "experience base" also influences the SEL.

This was one of the standard satellite systems being built at GSFC. However, the goals of the project did differ from the general goals by being a "designated SEL" project, i.e., a defined process was adopted and the process modeled and measured.

The project had a strict time schedule, working on one subsystem concurrent with other subsystem developments. Economics were not a factor since the project was "routine" and predictable. Reliability was important but was handled as usual at GSFC. Quality ditto.

Timeline: Consider the project to be the Coarse/Fine Attitude Determination System (CFADS) application as described below. The "extended project" is CFADS plus the SEL experiment structure. We will blend these together since both are the subject. The timeline for experiments is:

Experiment 1 (e1):	Sept. 1988--1990	ACME a.k.a. CFADS
Experiment 2 (e2):	1990--May 1992	ISTP
	1990--Dec. 1991	SAMEXTP
Experiment 3 (e3):	92-??	TOMS

Experiments started with courses from Basili (University of Maryland) and Terry Baker (IBM) and Michael Dyer (IBM) plus Cleanroom originator Harlan Mills' "Theoretical Cleanroom" talks and experiences of previous project teams.

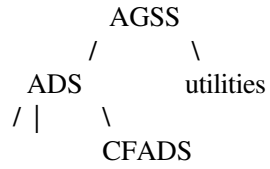
Management: Scott Green is the manager of the project for e1, and the process modeller for e2 and e3. The process modeller defines the goal of the experiment, data to collect, and follows through with the data collection and analysis.

The project manager determines the size of build, one of the trickier parts of the management of the process. Members of both teams conduct the requirements analysis.

3.3.2 Application

The following is the context of CFADS in the Attitude Ground Support System:

AGSS: Attitude Ground Support System
ADS: Attitude Determination System



CFADS was a real project with a tight time schedule. A secondary goal was a SEL experiment to characterize the methodology. A novelty was the merger of two sub-systems previously doing "coarse" and "fine" analysis separately.

3.3.3 Formal Methods and Tool Usage

Criteria: Cleanroom was chosen as a state-of-the-art technique to import into NASA. The rationale for how Cleanroom was used (emphasizing reading, not verification) went back to earlier experiments at the University of Maryland comparing testing and reading. They noted that reading was more effective, to the extent that reading can be viewed as a verification technique in itself. It is naturally linked to formal methods, by requiring a structured way of reviewing the code, namely stepwise abstraction.

Experimental Process:

0. A course on Cleanroom, including how to read code. These were lectures, not labs.
1. An independent organization sent the specifications to SEL.
2. For the first month the developers and testers worked together to understand the specification. Requirements analysis.
3. Development of the design then started. Developers used state machines in CFADS. The testing teams worked out the statistical patterns of the inputs and developed the test suite.
4. Peer review of the design and software. They had relatively small incremental builds.

The formal methods used were state machines and box structures. Such approaches allow for good understanding of the system and result in clean interfaces. There was limited teaching of box structures and state machines and SEL modified the Cleanroom methodology to their needs. Essentially, they took what they thought would work within the organization. Box structures are being used in the second experiment.

The requirements document for CFADS looks like a data flow diagram. Usually, SEL abstracts from the data flow diagram using Cleanroom. However, Green and Basili have found that the resulting implementation, using Cleanroom, is much like what they have had in the past. This group is very knowledgeable of their domain of application.

No special tools were used nor are tool developments planned.

Experience: The CFADS experiment resulted in several observations.

The application was mainly formulae, requiring less logic and more sequential code. This may have affected the reading process. The equations came from the specification.

Regarding box structures, if one is unfamiliar with the system, then box structures are good for understanding. In the NASA applications, the requirements are usually in the form of data flow, where the box structures can be used to analyze the flows. But the system was designed like all previous ones, in terms of architecture so value of state machines was not maximized. Another view is that they used state machines as models.

CFADS incorporated numerical methods techniques such as least squares. The specifications were also very detailed and included algorithms. The specifications are generally not very stable. However, the better understanding of the specifications that resulted from their use of Cleanroom made it easier to modify the software according to changes in the specification.

3.3.4 Results

Experimental Results: From the SEL perspective, the results of using Cleanroom have been favourable. SEL feels that Cleanroom is effective for <50KLOC (and up to four person-years of effort).

Experiment 1 on 40KLOC yielded good results versus the SEL baseline: half the number of changes, three-quarters the number of errors, and nearly twice the productivity. The team was able to handle unstable specifications and successfully mastered the review process, although there were weaknesses in the technical aspects of Cleanroom involving abstractions and operational testing profiles. The second experiment (on two project of 20 KLOC and 150 KLOC) brought in a process handbook and focused more training on the use of abstractions. However, while the trends are favourable, the second experiment produced fewer changes and errors while maintaining the SEL baseline productivity.

Improvement Trends: Two issues cloud the second experiment: (1) larger size and (2) external contracting. The outside/inside team worked together (physically), as normally occurs in NASA. External contracting requires additional tailoring and further investigation. It is hard to control all factors in experiments such as this. Therefore, trends are more important than actual numbers. The results are not conclusive but their data shows their Cleanroom approach to be effective for small projects (four person-years), though less assured up to 150 KLOC.

The SEL approach will be spread to four other NASA sites through NASA cooperative arrangement, but this SEL approach is more general than Cleanroom.

Lessons learned have been codified in a NASA guidebook [3], including such general advice:

- Do Cleanroom to a level of rigor that is appropriate.
- All methods must be tailored, which the NASA guidebook assists.

3.4 Evaluation

Since COBOL/SF is the primary application, the evaluation is performed with respect to that product and the IBM experience. The NASA Goddard experience is provided as background, but since the Cleanroom methodology was tailored extensively we have not incorporated it into the evaluation, except as notes of other points of view. Also, our NASA Goddard interview was brief, less focused on formal methods, and did not gather sufficient information to complete an evaluation. However, the SEL experiment papers and the NASA Goddard Cleanroom Guidebook are recommended for more insight into the general methodology.

3.4.1 Background

Product:

A commercial language processor, the COBOL Restructuring Facility.

Industrial Process:

A new methodology, Cleanroom, was used in the product development. Various standard industrial practices are greatly modified, e.g., no unit debugging in favour of verification by inspection. Incremental builds provide both system planning and reliability estimation.

Scale: This product came in at 80KLOC although it was projected to be much larger (400KLOC). 70 person-months of effort were involved.

Formal Methods Process:

Cleanroom involves mathematics-based specification and verification using some appropriate formalism (here function-theoretic), and statistical testing for quality certification.

Major results:

COBOL/SF was viewed as highly superior with respect to both reliability and productivity, leading to efforts to institutionalize the Cleanroom process through an IBM Software Technology Center.

Interview profile:

Approximately one-half day with Richard Linger, STC head, and Phil Hausler, project member and Cleanroom process manager.

3.4.2 Product features

Client satisfaction	n/a	(1)
Cost	n/a	(2)
Impact of product	n/a	(3)
Quality	+	(4)
Time to market	n/a	(5)

(1) IBM clients seem to be satisfied by the COBOL/SF process. An IBM Application Brief describes the experience of Pratt & Whitney. However, we have insufficient information with which to draw a conclusion.

(2) We have no data regarding product cost.

(3) The COBOL/SF product is on the market and used, but we have no data on impact of the product on IBM revenues.

(4) Error rates were claimed to be below industrial average and errors were minimal to fix. Formal methods, as both formalism and discipline, were credited.

(5) While time to market was very important, we have no data to determine how important. The use of formal methods seemed to provide a steady process with ability to overcome what might have been significant technical obstacles (COBOL semantics, algorithms).

3.4.3 Process features

General process effects

Cost	+	(6)
Impact of process	+	(7)
Pedagogical	+	(8)
Tools	0	(9)

Specific process effects

Design	+	(10)
Reusable components	n/a	(11)
Maintainability	+	(12)
Requirements capture	n/a	(13)
V&V	+	(14)

(6) A relatively small team developed this product, with credit given to the formal methods influence (which may have motivated and attracted personnel).

(7) The Cleanroom methodology gained recognition within IBM.

(8) The Cleanroom methodology has become part of a broader IBM education effort. Individuals on the project learned both the new methodology and the technical aspects of this language processor. Feedback is built into the Cleanroom project monitoring system.

(9) Tools did not play a major role in the COBOL/SF project. Cleanroom emphasizes mental power.

(10) The function-theoretic approach was applied with success to this particular problem.

(11) Part of previous versions were reused, but we did not discuss the role of formal methods in that.

(12) This product is claimed to require a small amount of maintenance, less than one person per year.

(13) By inference, Cleanroom is claimed to support requirements capture by forcing stabilization and following an incremental build. However, we did not discuss that aspect in this application.

(14) Good error results---i.e., low defect rates per KLOC and severity---are claimed.

3.4.4 Observations

The general outline of the product and the process are clearly presented in the CompSac paper [1]. Such data is essential for empirical case studies to begin building cost models for formal method processes.

The interviews exposed several aspects of the application. While the product technical base is a variant of compiler technology and draws upon the core of computer science, restructuring to the degree required by COBOL/SF was regarded to be, and was, a technical challenge. Computer science expertise, including mathematics, was required for the team. Cleanroom may have worked well, primarily because of the nature of the problem. But embedding the computer science expertise in a process with extensive review and statistical testing distinguished the COBOL/SF project as an industrial process.

Cleanroom is a difficult process to understand because it mixes together so many aspects. Despite our readings and interviews, we still do not have a strong feeling for the formal methods used. This point is addressed in the next section.

Cleanroom attracts criticism on several points: Does the discipline amount to a "strait-jacket"? Would loosening up some aspects, e.g., allowing execution, provide some rewards and trade-offs? How strongly does the choice of formal method matter? How much does the degree of formalism matter?

While Cleanroom is used in several IBM projects, it has hardly spread like wildfire, despite the positive COBOL/SF results. Are there inhibitors or does it just take time?

Cleanroom pragmatism is notable. The simplest formal techniques are sought and shorn of language concerns, such as schemas in Z. Management thinking is then applied to draw personnel motivation, measurement, and process structure around the formalism. Transfer is promoted by backing up the formalism and management process with a broader industrial education base.

In contrast, the NASA Goddard experience provides a perspective on Cleanroom where the formal methods usage is much less "hard-line." Less training is expected and available than in IBM with the anticipation that, if it proves itself, Cleanroom will be phased in gradually. As well, formal methods are viewed in the Goddard environment more as a review structuring mechanism rather than as a verification technique.

3.4.5 Formal Methods R & D Summary

3.4.5.1 Methods

Specification

From top to bottom, functions were the basic description, mixed with grammatical formalism. A newer approach, the box structure, aims at providing a more systematic top-level and decomposition technique.

Design and Implementation

Following the function-theoretic line of formalism, design is a process of (1) choosing more concrete data structures and refining functions and (2) decomposing functions to move toward standard language constructs (loops, conditionals, sequences, and assignments).

Validation and Verification

Validation comes from two perspectives: (1) the design team must agree that the specification says the right thing with regard to the problem before designing to it and (2) the test team must characterize the environment adequately to come up with usage profiles and therefore integrate other views of the requirements. Verification is split into (1) the review process based on correctness questions and mental proof and (2) testing guided by a reliability model.

3.4.5.2 Tools

Language processors

Only simple editors are required or used. However, some tools like VC Generators would be useful to track the verification process (see [4], Appendix A).

Automated Reasoning

Proof checkers and theorem provers were considered less important than the mental verification that is the basis of reviews. Some mechanical support is currently under consideration.

Other tools

The wastebasket as the repository of overly complex or out-of-control designs is a useful concept. Easily implemented, too.

3.4.5.3 Recommendations to the FM Research Community

The results of Cleanroom warrant more study by the formal methods community than has occurred. While papers have been published on functional verification and box structures, these are not usually considered in the common lists of formalisms. Given the extent of experience, the specific packaging provided by box structures and functional verification approaches should be compared and contrasted with other approaches. The weaknesses of the methods should be identified, as well as the intrinsic reasons for why they work when they do. For example, we do not have an adequate characterization of the claim that function-theoretic methods scale whereas axiomatic methods do not.

Given the extensive reliance on process, tool developers may not find much of interest to support in the Cleanroom methodology. However, there may be various toolsets, extending the traditional theorem prover, that would prove valuable and interesting.

Reliability models are not well understood in the formal methods community. Their interaction with the end results of applying formal methods is worth more attention. For example, does the gain come from the usage analysis process or from the reliability model? Is the implicit "closed specification" an important factor?

Reviews and inspections are inherent in industrial process but less attractive in academic-oriented software engineering. Can formal methods be taught in the context of reviews, e.g., following the correctness questions? Can reviews be taught as "structured reading according to the formal methods structures"? The Goddard Cleanroom Guidebook provides an accessible industrial process description.

3.5 Conclusions

Cleanroom is an ambitious methodology that interweaves several software development practices. Its premises are that teams of software engineers can apply these practices, following a discipline that drives toward zero defect software. Because the methodology runs against the grain of many companies' practices, it requires significant commitment and time to change an organization. For our purposes, because Cleanroom incorporates so many practices, it is difficult to assess the specific contributions of formal methods. To date, these appear to have been primarily in the form of guidance for designing and checking software as it is developed. Better understanding of the specific techniques (functional verification, box structure specification) would help clarify their value and determine the range of appropriate alternative notations and approaches.

4 DARLINGTON: TRIP COMPUTER SOFTWARE

4.1 Case Description

The Darlington Nuclear Generating Station (DNGS) is a four reactor unit station under development on a site east of Toronto. Each reactor will have two independent shutdown systems. The first shutdown system, SDS1, operates by dropping neutron-absorbing rods into the core; the second, SDS2, operates by liquid poison injection into the moderator. Summaries of SDS1 and SDS2 functionality are described in Section 0.

The shutdown systems are safety critical and require high levels of confidence. Obtaining confidence in any shutdown system has been of particular concern in licensing nuclear generating stations. In this instance, the issue was further compounded by the decision to fully implement the decision-making logic on computers. Consequently, issues pertaining to the reviewability of the software, amongst others, became crucial. Difficulties in obtaining an operating license for the DNGS by Ontario Hydro (the operator of Darlington) and the AECL (the developer of the shutdown systems) because of a lack of confidence by the regulator, the Atomic Energy Control Board of Canada (AECB), in the shutdown software resulted in the application of formal methods techniques.

4.1.1 Product and Process Profile

Size:

Specifications, code and proofs require 25 three-inch binders for each of the shutdown systems.

Code: 6,000 lines of assembler distributed across SDS1 and SDS2.
SDS1 -- 7,000 lines of Fortran code.
SDS2 -- 13,000 lines of Pascal code.

(See Section 0 for clarification on code size.)

Stages:

June 1982 AECB informs OH that it has no objections, in principle, to computer-based shutdown systems.

Feb 1983-mid 1987 Implementation proceeds. AECB audits process and the shutdown system is identified as a potential licensing impediment.

mid 1987 AECB awards a research contract to Dave Parnas.

mid 1987-end 1988 Parnas identifies areas of concern; Ontario Hydro uses IEC880 to assess shutdown system.

Jan 1989 Parnas proposes walk-through to reduce licensing impasse.

May 1989 Agreement on mechanism for formal inspection reached.

Aug 1989 Formal inspection of SDS1 starts.

Nov 1989 License for 1% power operation issued following completion of SDS1 inspection.

Jan 1990 Formal inspection of SDS2 starts.

Feb 1990 Completion of formal inspection. License for 100% full power is issued.

4.1.2 References

1. ``The Canadian Process for Regulatory Approval of Safety Critical Software in Nuclear Power Reactors." G.J.K. Asmis, J.D. Kendall, H.O. Tezel. In *Proceedings of the International Conference on Control and Instrumentation in Nuclear Installations*, Glasgow, Scotland, May 1990.

2. ``Reviewable Development of Safety Critical Software." David L. Parnas, G.J.K. Asmis, J.D. Kendall. In *Proceedings of the International Conference on Control and Instrumentation in Nuclear Installations*, Glasgow, Scotland, May 1990.

3. ``Verification of the Shutdown System Software at the Darlington Nuclear Generating Station." G.H. Archinoff, R.J. Hohendorf, A. Wassyng, B. Quigley, M.R. Borsch. In *Proceedings of the International Conference on Control and Instrumentation in Nuclear Installations*, Glasgow, Scotland, May 1990.

4. ``Licensing Safety Critical Software---The Good, The Bad and The Ugly." G.H. Archinoff and R.A. Brown. Undated manuscript, Ontario Hydro.

5. ``Software Requirements for the A-7E Aircraft." K.L. Heninger, et al. NRL Memorandum Report 3876, U.S. Naval Research Laboratory, November 1978. (This document introduces the Software Cost Production (SCR) method.)

6. ``Evaluation of Safety-Critical Software." D.L. Parnas, et al. *Communications of the ACM*, Vol. 33, No. 6, June 1990.

7. ``An Application of Fault Tree Analysis to Safety-Critical Software at Ontario Hydro." W.C. Bowman, et al. In *Proceedings of the Conference on Probabilistic Safety Assessment ad Management*, April 1991.

4.2 Interview Summary

Interview Profile

1. Date: 10 June 1992 (1 day)
   Respondents: Glenn Archinoff, Dominic Chan, Rick Hohendorf, Paul Joannou and Alan Wassyng
   Organization: Ontario Hydro, AECL and a consultant
   Description: Interviewed by Robin Bloomfield and Dan Craigen (author)

2. Date: 12 June 1992 (1 hour)
   Respondent: Richard Taylor
   Organization: AECB
   Description: Interviewed by Robin Bloomfield and Dan Craigen (author)

3. Date: 9 June 1992 (half day)
   Respondent: David Parnas
   Organization: McMaster University
   Description: Interviewed by Robin Bloomfield and Dan Craigen (author)

4. Date: 3 June 1992 (30 minutes)
   Respondent: Nancy Leveson
   Organization: University of California at Irvine
   Description: Interviewed by Dan Craigen (author)

(Note 1: In this summary, we will use the interview with personnel from Ontario Hydro and AECL [Interview of 10 June 1992] as the primary source of information. The interview reports on Taylor, Parnas and Leveson are used only to augment the information obtained from the first interview.)

(Note 2: The material presented in this section has been augmented with information extracted from the references.)

(Note 3: Most of the experiences discussed here pertain to the development of SDS1.)

4.2.1 Organizational Context

The following organizations and consultants were involved with the DNGS shutdown systems:

- Ontario Hydro (OH): The operator of the DNGS. OH had contracted AECL to develop the shutdown systems. Ontario Hydro is a provincially run utility (owned by the province of Ontario).

- AECL: The developer of the shutdown systems (and, more generally, of the CANDU[1] reactors). AECL is a crown corporation (i.e., owned by the Canadian government).

- Atomic Energy Control Board of Canada (AECB): The licensing authority for nuclear power stations in Canada.

- David Parnas (then at Queen's University; now at McMaster University): Principle investigator of a research contract awarded by the AECB to Queen's University. Parnas and his research group were to recommend areas for improvement and approaches for gaining confidence in the software.

- Nancy Leveson (University of California at Irvine): A consultant hired by Ontario Hydro to review and comment on the software used in the shutdown systems.

4.2.2 Project Context and History

Most of the individuals from OH, AECL, and AECB have engineering degrees, and, in fact, the first two versions of the software were developed by individuals who had limited software engineering training. They had received training in safety system engineering and control systems. Latitude in the requirements specification was permitted so as to allow for engineering judgement. During the development of the second version of the software, and through the third, an increasing number of individuals with software backgrounds were hired. These new individuals did not necessarily have backgrounds in nuclear engineering. The general approach at OH has been to populate projects from various disciplines.

Parnas and Leveson have Ph.Ds and are active researchers in computing science (Parnas also has an engineering degree). Both individuals are well-known for their concerns about the use of computers and software in critical contexts. Some of Parnas' graduate students (notably, A.J. van Schouwen) were also involved with the project.

As may be noted from the case description (Section 0), the development of the shutdown software has a long history: starting, effectively, from being approved in principle by the AECB in June 1982, through to licensing in 1991. A substantial amount of learning occurred for all parties involved. In particular, the assessment process was unclear. The report by Archinoff and Brown [4] comments that the perceived major advantages for using a software based shutdown system were lower cost; increased flexibility to make late changes; better and more flexible displays of information for the operator; the ability to monitor the computer and hardware device connected to it, and to turn failures of these

[1] CANDU is an acronym derived from ``CANadian Deuterium Uranium."

devices into safe failures, thereby increasing the reliability of the system; and the ability to implement more complex trip parameters which could lead to increased operating margins. ([4] concludes that the key perceived advantages of reduced cost and increased flexibility were not met in this instance.)

A specification that consisted of English and mathematical algorithms (in pseudo-code) existed prior to the formalization effort. There was also an extensive testing regime (including test scripts and a requirement that each statement in the functional specification be exercised by at least one test). The project did have a software development plan. For example, there were design principles, test overview documents and safety principles.

Between 1985 and 1990, OH and AECL had to react to AECB concerns and were successful in resolving a number of them. For example, they developed an elaborate metric based on safety, functionality, reviewability, maintainability and reliability; and audit teams used checklists. About twenty concerns were resolved, but an impasse was reached in 1989.

In [1], Asmis et al., noted a number of specific difficulties with the use of computers in safety-critical applications. These include:

i) ``Software failures, particularly under first use, seem relatively common, . . ."

ii) ``No detailed codes of practice exist, despite both national and international efforts to produce software standards," and

iii) ``exhaustive testing is not an appropriate technique to gain confidence in a software product."

In [2], Parnas et al., identify three components that should always be present in the assessment of safety-critical software: testing, inspection, and qualification of design personnel.

With respect to DNGS, AECB concerns, reinforced by Parnas, dealt primarily with reviewability and the difficulty of maintaining the system. In particular, Parnas felt that the documentation was not of adequate quality as it did not make review possible. Consequently, it could not be determined if the software was adequate. Parnas recommended that OH and AECL redesign and re-implement (using information hiding) the software. OH and AECL rejected the recommendation feeling that the code had a good history; that no safety-related issues had been discovered; and that modelling the software on the hardware shutdown system was a sound evolutionary development. Finally, it was agreed to perform a walk-through of the code. The walk-throughs were to show that

i) the code was consistent with the specifications, and

ii) the code had no additional functionality.

If the walk-throughs successfully demonstrated (i) and (ii) then the AECB agreed that the software would no longer be an impediment to obtaining a license for start-up of the DNGS. Other recommendations by Parnas on re-documentation and random testing were accepted, though Parnas is still not satisfied with the testing that was performed. On-line documentation is to be implemented in a future version.

At this juncture, two points are worthy of note. According to Parnas, the review process did find a major error in the code---subsequently corrected---that had escaped previous tests and reviews. The error was due to an ambiguity in the documentation. Secondly, by walk-through, we do not mean a conventional walk-through, but one based on formal methods and formal documentation.

During this time, OH hired Leveson to review the software. In general Leveson felt that the code was reviewable, suggested a safety analysis be performed and suggested some modifications to the way the code was written, for example, to use fully guarded if-statements and to remove some self-checking code.

As is discussed below, this verification effort became a bottleneck (from the OH perspective) in the licensing of the DNGS and was much more extensive in scope and effort than anyone had imagined. It is estimated that 35 person-years of effort was expended on the walk-throughs. (For example, as was noted by the contractor Wassyng, he was initially hired for a period of six weeks; this was ultimately extended for a period nearing ten months.)

The following are brief characterizations of the two shutdown systems. Figure 2, presented by Rick Hohendorf during the interview, diagrammatically represents the shutdown systems.

Add diagram

Figure 2 Shutdown Systems

The SDS1 shutdown system consists of the following (from [1]):

- Operates 32 spring-assisted gravity drop shut-off rods.
- For each of three channels, neutronic parameters are measured by multiple in-core flux detectors and one ion chamber. Process parameters for heat transport pressure, steam generators and pressurizer levels, heat transport flow, steam generators feedline pressures and moderator level are derived from pressure or pressure differential transmitters.
- SDS1 is independent of SDS2 and of the digital control system.
- SDS1 votes on two out of three channels (general coincidence). Voting is external to the trip computers and is done using relays.
- The unavailability requirements are: < 1.0E-03 (years per year).
- Testing of the SDS1 is performed through the Display/Test computer.
- Calibration data is entered at the Control Room panels.
- The trip computer transmits internal status through a serial link to the Display/Test computer.
- The trip computers are General Automation hardware.
- The required SDS1 loop time is approximately 50ms.
- No `operating system' is employed.
- [Implemented in Fortran and Assembler]

The SDS2 shutdown system consists of the following (from [1]):

- Operates a poison-injection (solution of gadolinium nitrate) shutdown system into the bulk moderator through eight horizontally located nozzles.
- For each of three channels, measurements of reactor and process parameters are similar to SDS1. Sensors and all cabling is totally separate.
- SDS2 is independent of SDS1 and of the digital control system.
- SDS2 votes on 2 out of 3 channels (local coincidence). Local coincidence voting is carried out by software.

- Power Rundown Discriminators (on certain parameters) block the action of SDS2 if SDS1 trip has been successful.
- The reliability requirements are the same as SDS1.
- The trip computers use DEC hardware.
- Communications to Test/Display computers similar to SDS1.
- The SDS2 loop time is approximately 65 ms. Multiple calls to high priority modules are required within one loop cycle.
- A "kernel" for real time clock processing, library functions and fatal error processing has been specifically written for SDS2.
- [Implemented in Pascal and Assembler]

(Note: [1] actually reports the number of lines of code in SDS1 and SDS2. However, these figures are at variance with other references and verbal reports. See Section 0)

4.2.3 Application Goals

Hohendorf, in responding to the initial questionnaire, described the application goals:

> ``OH and AECL have a long history of using digital control computers for control of nuclear reactors and other systems within nuclear generating stations. It was a natural evolution to extend this to shutdown systems. Shutdown systems in the past had already incorporated computers for process monitoring. Use of computers to implement the trip logic was felt to have economic advantages, and would allow design modifications to be more easily incorporated later in the project."

This project became crucial to OH once it became an impediment to receiving a license from AECB. In general, the shutdown system is one of four special safety systems at Darlington and hence is important regardless.

Shutdown systems were not a new application domain to OH and AECL. However, there was new technology through the transition to software, and formal methods were certainly a novelty.

4.2.4 Formal Methods Factors

OH and AECL had no knowledge of formal methods prior to this work and, as a result, it became an on-the-job training exercise with substantial in-house technology development. As part of the project, OH also investigated some U.K.-developed tools.

In writing the software, a general design principle of emulating a hardware design was followed. (In this instance, the hardware shutdown system at the Bruce Generating Station.) When Parnas recommended that the software be re-implemented, OH and AECL objected in that their code had a good history and no safety-related issues had been discovered. In their view, there was increased danger from re-implementing. Hence, and as stated in Hohendorf's response to the first questionnaire,

> ``The use of formal methods was chosen as an alternative to redesigning the software using `information hiding,' a software design technique recommended by the consultant to the regulator, David Parnas."

OH and AECL also concluded that the AECB was listening carefully to Parnas and that it was not worth their effort to investigate alternative formal methods technologies to those being proposed. (AECB has, throughout the exercise, been careful in obtaining other opinions and comparing them with Parnas' opinions.)

Respecting the quote from Hohendorf, Parnas comments that he recommended formal methods and that, unless OH and AECL restructured, the formal methods would be more expensive and time consuming than necessary. Restructuring is not an alternative to formal methods, but a means for making formal methods more practical.

4.2.5 Formal Methods and Tools Usage

As is described in [3], the software development process consisted of the following Verification and Validation activities: unit testing, integration testing, validation testing, in-situ trajectory-based random testing, software assessment, software hazard analysis, and formal verification.

The formal verification effort, in the terms of [3], consisted of a document (SWDS: Software Design Specification) that describes software requirements in a tabular format with the contents being expressed mathematically. Program function tables are developed for the various code routines, which describe, in a tabular manner, the functional effects of a routine on its parameters. The consequential `proof obligations' are that the SWDS and program function table descriptions of a routine are in consonance. Linkage tables were used in the SWDS and the program function tables so as to identify dependencies. Means for handling asynchronous processes were also developed [3].

In applying the formal methods, the effort at OH and AECL was divided as follows:

i) Formalizing the informal requirements. This work was led by Dominic Chan (AECL).

ii) Developing the program function tables for the code. This work was led by Rick Hohendorf (OH).

iii) Demonstrating the consistency of the code with the specifications. This work was led by Glenn Archinoff (OH).

During our interview, we asked the aforementioned for their comments.

Dominic Chan was responsible for formalizing the requirements on SDS1. Jim Kendall (then of AECB) sent some pages from the original Software Cost Reduction specification [5] to AECL. There had been complaints about ambiguity in the informal specifications and Kendall stated that the Software Cost Reduction style was what AECB wanted. Hence, AECL went about rewriting the requirements with the limited information that they had. They modified the notation marginally. Chan recommends the formalization of requirements, as the formalization does not require excessive effort and there are definite benefits in improving clarity. No problems with the design were found and there were no major discrepancies. Chan also felt that the proofs did not add much (except to convince AECB of consistency). However, Parnas notes that discrepancies were found and convinced AECB that inconsistencies were present. At most, three people were involved in developing the specification.

Rick Hohendorf reported on developing the proof function tables, but found it difficult to discuss because there were concerns that impacted the effort. For example, it was a moving target and they had to define a base process and evolve. Effectively, they started with the second version of the software and updated with the changes leading to the third version. There were major problems with the design methodology. For example, they had some assembler code, but the technology was directed at higher-order languages. (Parnas disagrees: the technology can be directed at any imperative language.) As well, they had an operating system kernel on SDS2 and had difficulties with calling procedures.

Hohendorf found the effort of performing the analysis to generate proof tables a valuable exercise that helped to identify extra complexity and some problems with timing. However, there were a number of "arbitrary rules" that were frustrating. For example, Hohendorf saw little utility in presenting the variable isolation tables, especially since Archinoff's group did not make use of these tables. (Variable isolation tables specify the effects of a routine on a specific routine parameter.) Assumptions were identified and analyzed.

It had been OH's experience that no one outside of the development group would take a hard look at the code. Consequently, Hohendorf did find a use in the development of the program function tables and proofs in that it forced a detailed review of the software.

Glenn Archinoff was in charge of the "proof" effort. Each system variable was identified with a corresponding software variable. Linkage tables were used to navigate through the specification and program function tables. Module-by-module checking was performed. Many of the specification tables were in event format and had to be converted to condition tables. For the most part, he felt that the process worked well. The actual comparison process was somewhat more ad hoc. All of this was labour intensive. At one point, up to 25 to 30 people were working on different aspects of the verification and, when it was clear that the verification was delaying licensing, Archinoff had his pick of the "best" staff at Ontario Hydro.

No specific formal methods tools were used; analysis was manually performed and Microsoft Excel was used to produce program function tables. There was also the problem that the process kept changing and completion was always "two weeks away." As reported in [3]:

> ``The process, as applied to the DNGS software, is extremely labour intensive. Moreover, the codes were not particularly large, comprising about 20,000 lines of high level language, and about 6,000 lines of assembler, in total. There is a clear need for tool support to make the process practical."

4.2.6 Results

AECB agreed that the shutdown systems were no longer an impediment to licensing. All parties involved believe that the system should be licensed; however, the software will have to be redesigned for long-term use if there were any functional modifications. As a consequence, Ontario Hydro and AECL, in conjunction with AECB, are designing a new set of standards for future software developments.

In issuing the license on February 23, 1990, the letter from AECB to OH stated:

> ``The Board has requested me to advise you that it has concluded that the existing shutdown system software, while acceptable for the present, is unsuitable in the longer term, and that it should be redesigned. Before redesign is undertaken, however, an appropriate standard must be defined. Accordingly, Ontario Hydro is requested to propose, by 30 April 1990, a program to allow this to be achieved."

The overall cost of the Darlington shutdown computer systems (including hardware) was about $16 million (Canadian). Of that total, 25% was spent on the verification effort. The verification effort included OH and AECL working out the process and refining Parnas' recommendations. (It was initially predicted that by computerizing the shutdown process OH would save $1 million (Canadian). There was also consideration, once problems with licensing arose, to revert back to a hardware system; however, it would have taken a year to procure the appropriate specialized hardware.)

It is estimated that there was at least a two to three month delay (and perhaps up to a year) in obtaining the license because of the problems with the shutdown system. Each month's delay cost Ontario Hydro $20 million (Canadian) in interest payments. The complete shutdown system would have cost in the order of hundreds of millions of dollars. Hence, the shutdown computer systems are only a small component of the overall price.

A number of consequences have resulted from this project. OH and AECL believe it is clear that they will have to use similar techniques on other safety-critical projects. Additionally, on the positive side, OH and AECL management is significantly more aware about software engineering issues. On the other side, some managers are concerned about making changes (perhaps migrating to software) because of the potential for uncontrollable costs. Even though the changes could be justified on economic and technical grounds, they could not be justified on licensing grounds.

Both OH and AECL felt that they had achieved an appropriate quality system without using formal methods (as, from their perspective, no safety-related problems have been discovered); formal methods were required for licensing purposes.

[4] also summarized various lessons:

1. Unless the issues that arose from the SDS experiences are resolved, it will be difficult to use software in safety-related applications.

2. There must be an agreed upon standard for software engineering.

3. OH and AECL feel that the AECB had too great a role in influencing the design of the software. (Parnas disagrees. AECB did not influence the design of the software, but did influence the inspection process.) Agreed standards would resolve this problem.

4. Changes to the licensing ground rules during the design process must be avoided.

4.2.7 Interview with a regulator: Richard Taylor

Richard Taylor is now an employee of the AECB and at the time of this project was a consultant who was hired to review the material coming from OH and AECL. We interviewed Taylor as a representative of the AECB perspective on the project.

Taylor observed that the AECB has substantially less staff than at both OH and AECL. Hence, the onus for safety is with the latter two organizations and that, in general, AECB routinely only audits the processes used. On the most part, AECB prefers to use existing Canadian Standards Association (CSA) standards and, hence, an existing CSA standard on software quality assurance was used.

Shutdown systems are always a point of concern. On this particular development, AECB concerns were raised by their perception that the software quality assurance plan was not being followed; that the software was more complex than they had expected; and that the software was not of good quality.

One of the AECB staff (Kendall) identified that there was a problem here and that they needed to bring in outside expertise and discuss the developments with other regulators. Kendall suggested that Dave Parnas be approached. Additionally, the code and documentation were reviewed by international experts and found to be deficient. On the other hand, OH thought that they were approaching acceptability. They did not understand why what they were doing was insufficient and they also felt that they had fixed the quality assurance problems.

Moving from the CSA standard, it was then required to check the development against an international standard of the International Electrotechnical Commission (IEC80). This check uncovered further problems with coding, documentation, etc. At this point, OH could not demonstrate that the software was of appropriate quality and Parnas was brought in to help resolve the situation.

OH and AECL then had to develop practical methods for writing specifications in the Software Cost Reduction style and to reverse-engineer the documentation for the code. They also had to convince the AECB that the code met the specifications, especially as non-safety critical discrepancies were uncovered between the high-level requirements and the pseudo code. The Software Cost Reduction-style specification introduced yet another interpretation of functionality.

Taylor thought that one of the benefits of the formal specification was that it became easier to understand what was happening. This was much better than the English and pseudo-code specifications. On the other hand, the specifications were more laborious to write as the formal specification forced decisions to be made and reduced ambiguity. There was the additional benefit that all of the important decisions about the shutdown system were brought into a single (albeit, large) document.

It took Taylor one to two months to build his expertise on the style of specification. This learning curve includes time to understand the underlying philosophy. Unfortunately, many of the ideas were still under development. OH made significant contributions to the specification technology, for example, answering questions such as: What does an event table really mean? How does polling implement a concurrency mechanism? What is a boundary?

The proposed walk-through became a "guided inspection" that became something close to a "proof." At this point, Taylor's role became one of checking the proofs and, as a consequence, he ended up writing many of his own proofs. If his results were at variance

with reports from OH then questions were asked.

By the time the third version of the software was out, everything was proceeding smoothly as far as the process was concerned. Unfortunately, discrepancies and errors (in the program function tables and with the software design specification) were found.

As it stands, AECB is not fully confident that everything is OK, though no safety-critical problems with the software have been found. According to Richard Taylor, the official position of the AECB is that while the software is safe to run in the short-term, it is not maintainable and must be redesigned and redeveloped according to better software engineering standards.

Taylor feels that OH should have rewritten the code, but it is possible that OH did not know how to perform the rewrite. Clearly, the software was not implemented with formal methods in mind. In the future, one should keep in mind what verification tasks need to be performed and develop the software within that context.

4.2.8 Interview with David Parnas

David Parnas' participation in this project was as a Principal Investigator of a research contract between Queen's University (Kingston, Ontario) and the AECB.

Parnas outlined the technical approach taken and how it evolved. The initial motivation, following a review by himself of the software, was to get the software into a reviewable state (i.e., to document it so that it was comprehensible); comprehension being necessary but not sufficient for assurance. The problems of the loose specification were illustrated by the example of the trip level comparison discussed in [2]. In addition, a systematic review process was required of the code and its specification. Although Parnas recommended a rewrite of the software, this was rejected by OH as they felt the software already had a degree of assurance that would be jeopardized by rewriting. The information hiding principles were not implemented.

Parnas explained that his role, and that of one of his graduate students, A.J. van Schouwen, was to advise AECB and OH on what should be done and how to do it. Their role was not to undertake the detailed analysis.

The software shutdown system was modeled on the hardware version of the shutdown system. Ontario Hydro and AECL felt that they were emulating well-known techniques, but software people were writing the software and did not appreciate some aspects of the informal specification. As well, they complicated the code by including meaningless self-checking code. AECL finally agreed to remove the self-checking code. The initial specifications were in English.

Discrepancies found during the proofs of equivalence between function tables had to be

resolved by the safety division of Ontario Hydro. Code changes were made only if safety concerns arose. Parnas suggested the use of "formal model-based inspection" (some, for example, [6]) for the checking between tables (rather than proof). AECB has a list of discrepancies that were found during the formalism process.

Although the system had been under test for some months running on site, Parnas doubted the effectiveness of these tests. During a visit to the station, Parnas asked to see the results of a test and found a discrepancy. The test results were being processed manually and the staff doing this were several months behind schedule.

Parnas feels that the Pascal version (SDS2) was more complex than the Fortran (SDS1).

There was debate on whether the shutdown system should be periodically re-initialized. This would limit the test trajectories that would be required in the statistical testing regime. An example of where re-initialization was required, but not carried out, was with the Patriot missile system and, consequently, led to a failure during the Gulf War. Re-initialization was implemented, to some extent, on the shutdown system.

As to general results, the mathematical analysis did find errors and led to their correction. It resulted in increased confidence in the software and provided new insights into productive research directions.

Parnas observed that the system was licensed and that everyone involved felt that the system should be licensed. It is likely one of the most exhaustively analyzed pieces of code ever written.

Parnas noted that other problems at Darlington had caused delays in putting Darlington into service: vibration and problems with bearings. He feels certain that the shutdown system would have remained an impediment to licensing in the absence of the formal methods work.

Parnas would have rewritten the software as he feels this would have reduced the verification time. He would have also specified in terms of physical variables (system requirements) rather than I/O variables (software requirements).

In conclusion, Parnas felt that designers of specification languages should emphasize readability. In addition, he feels that well-known and well-understood mathematics can do most software jobs.

4.2.9 Interview with a consultant: Nancy Leveson

Leveson was hired by Ontario Hydro to evaluate their software and to determine what was required to make the code licensable.

Much of Leveson's work focused on improving the code structure, for example, removing aliasing and nested "if-then-elses". In the latter case, she was arguing for a transformation towards an approach akin to Dijkstra's fully guarded "if" command. Leveson appears to have been happy with the approach of writing the software in a manner modelled from a hardware shutdown device.

A software fault tree analysis was performed on the code [7] and, while no errors were found that could lead to a system hazard, 42 changes were made to the code as a result of the analysis and, according to Leveson, Ontario Hydro felt that the software and system were safer.

4.2.10 Project Milestones (OH Perspective)

Rick Hohendorf, in his response to the first questionnaire, provided a detailed set of milestones for the project. We include the summary here so as to provide an overview of the project. The "message communicated" line indicates when specific concerns or views were officially communicated to/from Ontario Hydro.

Date	Message Communicated
June 1/82	AECB has no objection in principle to computer-based shutdown systems. But, keep the system as simple as past designs.
Feb 23/83	CSA software standard & licensing guide referenced; document requirements identified.
June 8/84	Need for a well-defined SQA program reiterated.
Sept 17/85	QA Audit by AECB found that the software development plan was not being followed in some cases.
Oct 4/85	AECB concern that coding began before spec was issued.
Nov 29/85	Areas needing improvement: software documentation, testing, management responsibility.
Feb 13/86	Audit Corrective Actions deemed acceptable by AECB.
Feb 27/86	AECB approves installation of trip computers, contingent on completion of: - P-code revised to a standard - independent review process - long-term maintenance plan - outstanding documents
Mar 16/87	AECB review of documents revealed discrepancies; uncertainty whether software implemented requirements correctly. Identified as a potential licensing impediment.
May 5/87	General agreement reached with AECB staff as to changes to documentation.
June 18/87	AECB QA audit. No major faults found.
Mid-1987	AECB hired Dr. D. L. Parnas.
Sept /87	Initial Parnas review finds that software is not of adequate quality for a

safety critical application.

Nov 25/87 Preliminary Parnas report identified specific concerns.

Dec 21/87 Meeting to explain to Parnas his "misconceptions" about the shutdown system design.

Mar 9/88 Proposed corrective action did not address fundamental concerns: major software restructuring required. AECB lost confidence that effective corrective action would take place.

April /88 IEC-880 identified as an appropriate standard for safety software. Subsequently, OH carried out an in-depth software assessment against IEC-880, identifying many areas for improvement.

May 9/88 Final Parnas report and AECB statement of what they believe is required:

1. Restructuring,
2. Redesign for testability and reliability,
3. Re-documentation,
4. On-line documentation control system,
5. Independent Verification and Validation,
6. Independence study.

Jan 5/89 Software is not "reviewable." Parnas proposed a walk-through to resolve impasse.

4.2.11 Project Milestones (AECB Perspective)

Richard Taylor provided a set of milestones from the AECB perspective. While consistent with those above, there is a difference in emphasis and thus it is included.

1982 - OH asked AECB if there were fundamental reasons why fully-computerized shutdown systems could not be used at Darlington. AECB indicated that there were not. Advisory Committee on Nuclear Safety came to similar conclusion.

1982 - AECL feasibility study initiated

1982-85 - OH and AECL made decision to go ahead with Darlington SDS design (before feasibility study complete).

Late 1986 - AECB found recently available documentation difficult to review. By end of 1986 AECB convinced a serious problem existed, but unsure of its exact nature.

Early 1987 - AECB informed OH that software problems had potential for causing delays to Darlington.

1987 - AECB engaged David Parnas of Queen's University.

Oct. 1987 - Parnas advised AECB that serious design deficiencies existed in the software for SDS1 (and probably SDS2)

Nov. 1987 - Parnas recommended re-structuring, re-documentation, random testing

Jan. 1988 - AECB advised OH that SDS1 software should be re-designed.

Feb. 1988 - OH expressed confidence in existing software. Harlan Mills confirmed Parnas findings.

Mar. 1988 - AECB contacted nuclear industry software experts in US, UK, France and Germany. All advised that a serious problem may exist with Darlington SDS software.

May 1988 - Final Parnas report on SDS1 and SDS2 issued

July 1988 - OH advised AECB that software would be revised

Nov. 1988 - OH internal review (Hicks Report) concluded that the software did not meet original design standards and compared poorly with intent of IEC880

Dec. 1988 - Revised SDS1 code available - review showed only minor changes incorporated.

Status in Jan. 1989:

- SDS1 and SDS2 had been re-documented and random testing had been introduced.
- The code had not been re-designed to the extent that it could be easily reviewed.
- It was clear that major coding changes could not be made without incurring a major delay to Darlington.

Jan. 1989 - Revised SDS2 code available - review showed only minor changes incorporated.

Jan. 1989 - Parnas proposed method of reviewing code based on Mills program functions in table form. OH accepted Parnas proposal, again expressed confidence in their software, and engaged Nancy Leveson of UCI.

Mar. 1989 - Leveson reported findings on SDS1: basically similar to those of AECB. Recommended some code changes and fault tree analysis.

May 1989 - Agreement on mechanism for formal inspection reached.

Aug. 1989 - Formal inspection of SDS1 commenced.

Nov. 1989 - Licence for 1% power operation issued following completion of SDS1 inspection.

Jan. 1990 - Formal inspection of SDS2 commenced.

Feb. 1990 - Completion of formal inspection. Licence for 100% full power issued.

4.3 Evaluation

4.3.1 Background

Product: Computer-controlled shutdown system for a nuclear generating station.

Industrial Process:

Post-development mathematical analysis of requirements and code. Code developed based on an emulation of a hardware shutdown system. Substantial testing and review. System needed to be approved by the AECB before DNGS could be licensed.

Scale:

Specifications, code and proofs require 25 three-inch binders for each of the shutdown systems.

Code: 6,000 lines of assembler distributed across SDS1 and SDS2.
SDS1-- 7,000 lines of Fortran code.
SDS2-- 13,000 lines of Pascal code.

(See Section 0 for clarification on code size.)

Formal Methods Process:

Use of Software Cost Reduction-style tables to specify requirements and used in program function tables to describe functionality of routines. Corresponding specification and program function tables had to be shown (informally) to be consistent. Program function tables were reverse engineered from code.

Major Results:

DNGS was finally licensed and the shutdown system was no longer an impediment. OH, AECL, and AECB using their experiences to develop a standard for future development of safety-critical software. Research spin-offs.

Interview Profile:

Interviews with developers (AECL and Ontario Hydro; 1 day), regulator (AECB; 1 hour) and consultants (Parnas; 1/2 day) (Leveson; 30 minutes)

4.3.2 Product Features

Client satisfaction	+	(1)
Cost	n/a	(2)
Impact of product	+	(3)
Quality	0	(4)
Time to market	n/a	(5)

(1) Viewing the client as being the AECB, the AECB was finally convinced by the

arguments produced from the formal methods work that the code is adequately correct and safe.

(2) The information needed for determining this is unclear. It could be argued that the formal methods delayed licensing; but, on the other hand, licensing may not have been achievable otherwise.

(3) With the development of the product and the successful demonstration that the code is adequately correct and safe, the shutdown system was no longer an impediment to the licensing of the DNGS.

(4) Safety-related discrepancies were found in an earlier version of the software; none are known in the present version. However, AECB does not believe that the quality of the software was improved.

(5) The information needed for determining this is unclear. It could be argued that the formal methods delayed licensing; but, on the other hand, licensing may not have been achievable otherwise.

4.3.3 Process Features

General process effects

Cost	-	(6)
Impact of process	0	(7)
Pedagogical	+	(8)
Tools	n/a	(9)

Specific process effects

Design	+	(10)
Reusable components	n/a	(11)
Maintainability	n/a	(12)
Requirements capture	+	(13)
V & V	+	(14)

(6) The formal methods technology evolved as the project progressed; a fixed process was not in place at the start of the effort. Hence, tool support was absent and efforts had to be directed at developing the technology; not solely applying it.

(7) OH and AECL management have a greater awareness of software issues. Also, new standards and processes are being developed for safety-critical software. Conversely, some managers have been delaying replacing hardware systems with computers because of the experiences with the shutdown system.

(8) All involved learned and have been acting in response to their lessons. Parnas found interesting research areas; OH, AECL and AECB are more aware of formal methods, software engineering, and are working on new standards and procedures.

(9) No formal methods tools were used or available.

(10) While the design did not change, the increased precision and readability of the formal documents were deemed advantageous. Hence, there was an increased ability to review the system for purpose of licensing.

(11) Not applicable.

(12) Insufficient information.

(13) The procedures clarified the requirements and removed some safety-related ambiguities.

(14) Without the formal methods, the proofs of correspondence between specification and program function tables would not have occurred. The formalism was also extensively used in review and led to increased confidence in the system.

4.3.4 Observations

Given the inconsistencies on code size being received from different sources, we explicitly asked OH and AECB to distinguish between commented and uncommented code. For SDS1, the figures received are as follows:

Fortran

Executable lines of code	1,362
Comments	13,065
TOTAL	14,427

Assembler

Executable lines of code	1,185
Comments	6,521
TOTAL	7,706

During the interview with OH and AECL, there was a discussion on readability of the formal specification. There was the view that the original informal specification was easier to read for obtaining a casual understanding of the system. However, the formalism provided far better real understanding of the system; the amount of detail did not really matter. As to learning the formal notation, it was claimed that a couple of days were sufficient for familiarization, but a couple of months to be proficient.

It was also noted that while the specification style is based on the Software Cost Reduction specification [5], actually developing the program function tables and proving consistency with the specifications had not been previously done.

There appears to be ongoing disagreement between OH and AECL, on the one hand, and AECB and Parnas, on the other. The former still view that their approach of emulating the Bruce hardware implemented shutdown system was a sound, evolutionary engineering approach. Furthermore, they disagree that the difficulties in demonstrating consistency between requirements and proof function tables were a result of their approach and their refusal to use information hiding (Leveson is in agreement with OH and AECL on this point). AECB and Parnas have taken the view that the software is not maintainable at reasonable cost in the long term, and that software engineering techniques such as information hiding (thereby resulting in well-defined boundaries) should be used.

It was also our observation that all parties found the entire process onerous. It is clear to us that if reverting to a hardware solution had been a viable alternative, OH and AECL would have jettisoned the software shutdown system.

4.3.5 Formal Methods R & D Summary

4.3.5.1 Methods

Use of tables to describe requirements and programs. Generally, I/O behaviour described using a predicate calculus-like language. Program function tables are essentially a functional description, a la Mills, of a routine. Consistency proofs between specification and program function table were performed by hand. Testing and Hazard Analysis was also performed. Substantial reviews by independent bodies. For example, AECB and their contractors reviewed 30 to 40 percent of the routines. Asynchronous processes were analyzed independently with shared data objects viewed as inter-process I/O. Interaction of processes was handled separately from the function table analysis.

It appears that the majority of the verification cost came as a result of the manual proofs of consistency and learning a new technology. In our view, savings would have been made if the proofs and program function table derivations had been automated; solid automated theorem proving tools exist and the kinds of analysis needed for deriving program function tables is well-understood and, in a variant, is implemented in many existing verification systems.

4.3.5.2 Tools

There were limited tools available and none that would be viewed as being `formal methods' tools. There was an obvious need for language processors, cross-references, automatic determination of linkage tables and automated deduction.

4.3.5.3 Recommendations to the FM Research Community

We are left with the impression that the use of a formal notation (and, in particular, domain specific notations) for describing requirements is beneficial, especially as it may lead to clarifications earlier in the lifecycle. The lack of tool support aiding the proofs of consistency or in deriving proof function tables was a problem and substantially increased cost. (Note, however, that included in this effort was establishing the correspondence between variables---not usually what one considers as a proof obligation.) Our impression is that the technology already exists to help with such proofs. OH and AECL are already developing other tools (primarily to support documentation and to be consistent with their new proposed standard) to help in future projects where safety-critical software is to be used.

The OH and AECL position on modelling the software structure after a hardware device appears, on the surface, to be a sound engineering approach. On the other hand, it was argued that the insertion of various software engineering principles would have been of benefit (for example, by substantially reducing the amount of code). It is difficult to identify the specific research issues here, though certainly the structuring techniques in real-time systems are worthy of investigation. Regardless, the decision not to rewrite the software resulted in an evolutionary approach to the use of formal methods and in code that was more complex than necessary.

The importance of applying research ideas to realistic applications is demonstrated by this project. Although the SCR specification style had been in existence for a decade, this project helped to extend and better comprehend the technology to proof function tables and the "proofs" of consistency with routine specifications.

4.4 Conclusions

Though at a high cost, the application of formal methods to the shutdown system appears to have been successful in that discrepancies were discovered and analyzed and, in a few instances where safety was an issue, code was modified.

5 LaCoS ESPRIT PROJECT
Building Large Complex Systems using RAISE

5.1 Case Description

RAISE/LaCoS Background

This case includes interviews with two separate organizations, both involved in the ESPRIT II LaCoS project. LaCoS (Large Complex Systems) is intended to be a demonstration of the feasibility of using formal methods for industrial software development, based on the RAISE method and tools developed in an ESPRIT I project. LaCoS consumers of RAISE include Bull SA, Inisel Espacio SA, Lloyd's Register of Shipping, Matra Transportation SA, Space Software Italia SpA, STC Technology Ltd. (now part of BNR Europe Ltd.) and Technisystems Ltd. Based on this consumer experience, evolution of RAISE is performed by the producers: Computer Resources International, STC plc, and SYPRO A/S. Each consumer has a particular application area involved closely with its business interests. RAISE experience is being carefully monitored through consultants and questionnaires following an extensive criteria and a formal evaluation report series.

RAISE is a large language and toolset, which evolved from the Vienna Development Methodology (VDM). The RAISE Specification Language (RSL) is intended to support multiple styles of specification and levels of design refinement. For example, specifications may mix applicative and axiomatic styles. A "method manual" describes how one requirements analysis technique, CORE, may work in conjunction with RSL and prescribes an overall process for a complete formal methods life cycle.

The toolset is based around the Cornell Synthesizer Generator and a commercial data base. The Cornell Synthesizer Generator provides structure editing, transformation definition and execution, and other language support. One specific tool of current interest is a Justification Editor, which will link the RSL proof rules with a support system (editor plus rewriting plus simplifiers) for discharge of proof obligations. Computer Resources International (CRI), which bought out the formal methods organization of Dansk Datamation Center, provides commercial support for RAISE, which is now available in North America as well as Europe. RAISE has many of the same objectives with respect to formal methods as CASE does to structured methodologies.

The interview process and subsequent case analysis for this case focus less on the applications and more on the technology transfer process. Few applications are complete at this point (mid-1992) but this case study should provide a basis for interpreting results as they flow from the ESPRIT project. This case also represents a large current investments in a broad formal methods approach, including the commercial support from the CRI and the ESPRIT Consumer-Producer partnership.

Lloyd's Register partner

The Lloyd's Register case centres around the Condition/Performance Monitoring and Predictive System, an engine management advisory system for medium-speed, four-stroke ship diesel engines. It consists of modules for data acquisition and processing, engine condition assessment, fuel characterization, performance optimization, and prediction of maintenance. A generic and modular design is required and the implementation environments are limited to C, FORTRAN, and a specific AI language. The data acquisition and management module is the focus for RAISE experimentation. Requirements were acquired from a prototype system and client interviews and are viewed as typical engineering requirements. Thus, configurability and reliability are the application objectives.

Matra Transport partner

Matra is another LaCoS consumer partner, specializing in the same domain as the SACEM project: the use of computers for railway signalling. Like Lloyd's Register, there is a group that experiments with various formal methods on their applications. This interview was somewhat rushed and produced more sketchy information about the application but covered some points of view that correlate with the Lloyd's Register experience.

5.1.1 References

Current reports for the ESPRIT project:

1. "Experiences from Applications of RAISE." LaCoS project reports LACOS/CRI/CONS/13/V1 and LACOS/CRI/CONS/20/V1, June 1991 and March 1992.

2. "Criteria for Evaluating Applications in RAISE", LACOS/CRI/CONS/4/V2, November 1991.

3. Reps, T.W.; Teitelbaum, T. *The Synthesizer Generator.* Springer-Verlag, 1988.

5.2 Questionnaire 1

Lloyd's Register Engineering Services
Lloyd's Register House
29 Wellesley Road
Croydon CR0 2AJ, UK

Approach

(Note: As Richard Granville's response to the first questionnaire summarizes Lloyd's Register work, we include it here.)

When the project started, Lloyd's was mainly a "consumer partner", meaning that we used RAISE on our applications but did not develop any RAISE material (e.g., tools). We have also started on the work package "Systems Engineering with RAISE", which is being coordinated by SSI (Tarranto, Italy). We initially aim to investigate standards, generation of correct code and specialised RAISE methods.

A. Organizational Context

The Applied Information Engineering Group at Lloyd's Register is divided into two parts: services and research. Research projects are divided into three areas: knowledge-based systems, dependability engineering and formal methods. I am coordinator of projects in the formal methods area. From next month there will be four projects: LaCoS, MORSE (also based around RAISE), B User Trials and AFRODITE (which will use an object-oriented extension to VDM).

The LaCoS project is divided into a number of work packages. We participate in several. One work package is concerned with case studies (applications). We have six people working on LaCoS (none completely full-time).

Our goals include learning as much as possible about the practical application of formal methods. We aim to offer training and consultancy in RAISE and other formal methods.

B. Project Context and History

We are currently using RAISE on three applications:

1. The Bell and LaPadula security model. This is not a "real" application; its purpose is to investigate modelling security in RSL (and also as a training exercise). We have spent about two person-months on this work. So far, we have produced a draft report on the work. We intend to make minor changes and then issue the report. After this, we hope to use the RAISE Justification Editor to prove certain key properties that are required.

2. The Signal Condition Processing case study is part of a larger application, the Condition/Performance Monitoring and Predictive System (CPMPS). CPMPS is an engine management advisory system for medium-speed, four-stroke diesel engines. We have spent about eight person-months on this work so far. Progress has been a little disappointing, due to a number of factors.

3. The Safe Monitoring and Control System is a distributed controller, for use in mining and other industrial applications. We have spent about six person-months. We have used RSL to

formulate the requirements, which have evolved during this time. We are currently awaiting an updated statement of requirements from our client.

C. Application Goals

1. Mainly as part of our "systems engineering" work package - as an input to "specialised RAISE methods" and standards relating to security.

2. To rewrite part of the software, which is safety-critical (indirectly).

3. To produce a new generation of such controllers, which are certainly safety-critical.

D. Formal Methods Factors

In some ways, RAISE was chosen for us and we accepted this when we joined the LaCoS project. In addition:

1. Lloyd's believe that formal methods of specification are useful for a large class of systems, partly because the need for formulation forces people to consider the requirements carefully. Formal development (a much more difficult and time-consuming process) should be applied to critical systems.

2. On LaCoS and MORSE we use RAISE. On the other formal methods projects we shall be using B and VDM++. In addition, we may use Z for specifying some software in other projects.

3. We prefer RAISE because the tool support is good, RSL is a wide-spectrum language and there is the potential for formal development (refinement).

The last of these will be facilitated by the Justification Editor.

E. Formal Methods and Tools Usage

We regard the "RAISE method" as a collection of approaches to formal development.

One problem is that, because RSL is a large language, it is not obvious which style(s) of RSL to use at the various stages of development. We prefer to separate the concerns of formal specification and formal development. Thus the first RSL specification should be purely addressed at requirements capture, without any regard for development. Then one should outline the development route and write a new top-level specification that is suitable for refinement. This should be validated against the original specification (not verified as this will generally be impossible). Then one can (aim to) carry out formal development as in the RAISE method.

We have made extensive use of the syntax/type checker.

F. Results

We have found RSL good for formulation of requirements. See above for how we might have acted differently (on the Signal Condition Processing application).

No performance or productivity metrics were kept during the project. In any case, it would have been difficult to separate development time from training time.

G. Additional Information

None.

5.3 Interview Summary

5.3.1 Lloyd's Register

Date: April 16, 1992, 3 hours
Respondent: Richard Granville (Project Manager) and Tony Darlinson
Organization: Lloyd's Register
Description: Interviewed by Susan Gerhart (author)

5.3.1.1 Project Context (both Lloyd's Register and LaCoS)

Organization: The Lloyd's Register LaCoS project operates within a larger organization, the Performance Technology, Applied Information Engineering group, which is being developed into a consultancy operation.

The Performance Technology Department of Lloyd's Register is charged with both research behind and doing the assessments of systems for ships. They are now moving into chemical and nuclear systems as well. Three-quarters of the Performance Technology Department is the Application Engineering group (25 persons) which consists of 10 persons working on LaCoS, MORSE, B user trials, and Aphrodite. The other branches are Knowledge Engineering and Dependability Engineering, plus assessment services. Their strategy is to use collaborative projects to expand their consultancy expertise over a five-year period.

Sponsored by the U.K. Department of Trade and Industry (DTI), MORSE focuses on technical aspects of meeting standards for systems using microcontrollers in mining, shipping, and other process control. Aphrodite is a new ESPRIT project dealing with VDM++. Figure 3 summarizes the various projects.

Primary Organization	Lloyd's Register Engineering Services			
Business Unit	Performance Technology, Applied Information Engineering			
Products/ Services	Assessment Services	FM Research		
Projects		Information Security	Condition/Performance Monitoring and Predictive System	Distributed Controller
FM Projects		Security Model	Signal Condition Processing, and Safe Monitoring and Control System	
Funding		LaCoS (ESPRIT II)	MORSE (DTI)	Aphrodite (ESPRIT)
FM usage		RAISE	RAISE, B, Z	VDM++, Z

Figure 3 Lloyd's Register Projects

Educational Backgrounds: Richard Granville's background is Royal Signals and Radar Establishment, and Rex Thompson partners on a static analysis tool (MALPAS). He has been at Lloyd's Register for two years. Six team members are working on various parts of LaCoS and other assessment projects. Tony Darlinson (the other interviewee) is full-time on the RSL projects (LaCoS and MORSE); his background is a Ph.D. in physics. Another project member, K. Fox, has worked on Z tools and has been doing one of the applications. Granville manages the group and has reviewed specs, but has written them only as exercises in recent courses. They hope to build up the expertise to operate a consultancy around formal specs and standards. In summary, six people were working 70-80% on formal methods projects.

LaCoS Activities: Lloyd's Register has just made a major contribution to the LaCoS project in putting together a course on the justification handbook (~180 foils). This was previously the task of BNR (Bell Northern Research, which took over STC). On the tools side, Lloyd's Register is providing some consulting to CRI on algebraic simplification. BNR is continuing to work on the method manual.

The LaCoS project is starting to develop a library of reusable RSL components, but they are just at the stage of defining the criteria for the collection, which is expected to cover the usual data types, windows interfaces, etc. Another aspect of the LaCoS project is the Italian SSI effort in measurements, which is still in an early stage. They are looking at project management, metrics, training, etc.

Standards: Lloyd's Register is working with SSI on standards for code generation in the context of the U.K. Ministry of Defence's Draft Interim Defence Standard 00-55, e.g., does the RSL strategy meet the standard? and how does static analysis fit in? There has not really been any effect felt at Lloyd's Register of the Software Engineering Institute maturity model. They are more covered by IEC 65 and are very interested in 00-55. They are not in agreement with the 00-55 concern for formal development or static analysis, however. They see the possible use of 00-55 for safety-critical systems outside defence. Their basic belief is that the system engineering work package of LaCoS could lead to building systems that meet 00-55.

5.3.1.2 Applications

Three applications (see Section 0) are underway.

The security application is a short effort to look at RSL in the context of ITSEC requirements. They are examining the issues of combining the functional and security requirements.

The Signal Condition Processing application is a re-implementation of a part of the Condition/Performance Monitoring and Predictive System, partly using AI techniques. One difficulty encountered was that the system existed in several variations, making it impossible to integrate the formal methods-implemented subsystem. Lloyd's will take this down to C code. The application had no particular novelty -- it dealt with "batch processing" of data from sensors during a run of an engine.

The Safe Monitoring and Control System application is the most extensive project, being conducted under both MORSE and LaCoS. MORSE is looking at what the standards are requiring, particularly with a "safety platform". Currently, there are home-grown operating system and standard hardware components. MORSE hopes to build more standard software platforms, e.g., the schedulers in the executive. The MORSE partners are Transmitton (mining & utilities), Dowty Controls (airframe landing gears), and Middlesex Hospital (pathology). Research partners are SRI Cambridge and the Center for Software Reliability.

Lloyd's is also looking, in the context of system engineering, at the construction of tests for symbolic execution of specs. No particular method is used for capturing requirements, e.g., CORE.

5.3.1.3 Formal Methods and Tools Usage

Preliminary Conclusions: Their general conclusions about FM so far are that

- initial specification use is fully justified -- it makes you think more systematically

but

- formal development is much more difficult and they are not sure it pays off.

Is the value of formal methods in the methods or the controlled processes? Granville and Darlinson feel that, given the formal spec, many notations would suffice to carry through the development.

RAISE Method-specific Issues: Regarding life cycle issues in RAISE, the Lloyd's project sees a big difference (in styles) between requirements capture and refinement. One lesson along these lines was learned from the experimental Signal Condition Processing project, where the specifier was forced to fluctuate between specs for requirements purposes and specs that would lead readily into refinement. The variety of RSL styles permitted, but also exacerbated, this difficulty. They would now separate into

two levels (1) system modelisation (describing the system) and (2) refinement. It is easier to do the modelisation level in RSL, with its types and concurrency.

Lloyd's Register believes that if the top-level specification is sufficiently abstract, the style is not the big issue. They started using concurrency but dropped it because the RSL model was hard to understand, in favour of a single station point of view. Understanding RSL's concurrency model is an apparent barrier and training issue.

The executives are cyclic so the main concern is meeting deadlines within some number of cycles, and they tend to use static scheduling. This means a calculus of time is not required. The only issue was the overrun of slots within a cycle, in which case the system shuts down. Their general rule is that if you use interrupts you cannot reason about time. The application goal is to carry through formally this part of the executive within the MORSE project, looking more generally at how much of the platform to specify formally. Another goal is to get the clients up to speed -- Lloyd's Register is the intermediary for formal methods in MORSE.

A general problem is requirements iteration where the client can't see what is wanted. Often a specification is too "black and white" for clients and they are not prepared to deal with the precision. Lloyd's noted that frequent misunderstandings have occurred in the translation from informal to RSL spec.

RAISE Tools: So far the specs are just a few hundred lines -- about 10 modules -- but would be very hard to maintain. The tools help to maintain consistency through type checking and version management. Versioning in RAISE is becoming more "open", permitting the use of the Portable Common Tool Environment or other mechanisms.

Syntax-directed editing is considered discredited as a useful programming aid: acceptable as a pedagogical tool but not for real use. However, the implementation tool, the Cornell Synthesizer Generator, has been a good productivity tool for managing the language.

The Justification Editor tool is just coming out. Recently, Lloyd's Register held a three-day course on it. The meta-language is considered adequate. It allows mixing forward and backward proofs, e.g. selecting applicable rules, then rewriting and using facts/lemmas. An algebraic simplifier is desirable. One use of the Justification Editor might be to expand the concurrency ordering. RAISE probably will end up with a closed set of simplifiers working in the current mode of rule selection and application. The plan is to get experience with the bottlenecks in the Justification Editor and then improve it. An Animator, a specification executor, might be built around the Justification Editor as well. The RSL applicative style translates well into the functional languages. The Confidence Condition Generator, a tool for checking language construct usage, is desirable as a kind of automated inspector.

The reusable component effort is just starting. RAISE modularity is "object-oriented".

Code Level Experience: Technisystems (Greece) carried through a lower level implementation into Ada, of several thousand lines with some development steps, as an attempt to demonstrate the code level capabilities of RSL.

5.3.1.4 Results

- In general, Lloyd's Register felt that the requirements level use of RSL was fully adequate; however, concurrency has not yet been investigated.

- Competing with CASE tools is a concern for technology transfer of formal methods, although the extent of dissatisfaction with CASE methods and tools suggests that formal methods could have a good chance.
- It takes six months of using RAISE to be productive. LaCoS supports a review process with a consultant to each (Consumer) partner.
- Lloyd's Register is impressed with CRI and considers the RAISE trials very thorough so far. This is, for them, good ESPRIT leverage. Lloyd's Register is interested in both internal assessment and external business opportunities.
- Darlinson reported that the personnel like using RAISE, and therefore they will do a better job on their projects.
- Lloyd's Register wants to give some papers during the next year on their experience.

5.3.2 Interview with Matra Transport

Questionnaire #1 was not completed prior to the interview. Some of that data was gathered during the interview itself.

Date: April 16, 1992 (2 hours)
Respondent: Patrick Bern (Project Manager)
Organization: Matra Transport
Description: Interviewed by Ted Ralston and Susan Gerhart (author)

5.3.2.1 Project Context

Structure: The context of discussion was the LaCoS project, of which Matra is one partner. Matra Transport, like GEC Alsthom (the SACEM case), produces systems that must satisfy RATP (the Paris rapid transit authority) in the form of a specification, with complete derivation from it using formal methods. Matra is organized in matrix form.

Process: The Matra process model is represented by a "V" (see Figure 4), with a plan to use four steps of refinement, between specification and detailed design. Matra expects to add an annex to the specs and designs containing formal material. Experience with Meteor showed that that formal methods did not fit the standard V stages, hence the additional levels.

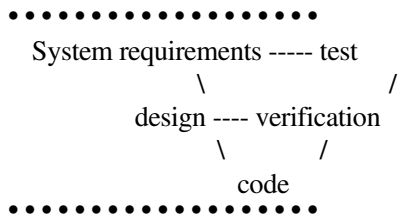


Figure 4 The "V" Process Model

5.3.2.2 Application

The rail systems are built using no interrupts. The systems maintain a "picture" of the track: passing through a segment of track, they build a picture of the next several segments and compute the stop point for controlling the train speed. The code looks like:

```
loop
        get track information;
        compute stopping point (and speed);
        deliver data to the track about the train performance
end loop
```

The Matra approach is to use formal methods for

(1) modelization, i.e., formal descriptions of the systems; and
(2) sequential code level specification, derivation, and proof.

5.3.2.3 Formal Methods Experience

Criteria: SACEM had followed the classical "code + assertions" route; later, they reverse engineered the specification using B with manual proof linking the new specification and the old assertions. Matra faced the problem of choosing its methods for the long-term, specifically to satisfy RATP, which wants formal proof. They looked at VDM and Z for model-oriented approaches, B and LaCoS for overall methodology, but did not look at North American approaches. Their criteria for considering various formal methods were:

- Derivation must be supported.
- Ordinary engineers must be able to become proficient.
- Proofs must be possible.

They were happy with B's use of classical set theory, which was familiar to most engineers from French universities. B also provided proof tools, but B was seen as appropriate only for sequential code and for dealing with issues close to code. They also wanted a method that could handle concurrency modelling, which is why they chose RAISE.

Training: There were 10 people using B in the Meteor ESPRIT project. The B engineers are recent grads, receptive to training in B. There have been no surprises in B training courses that consist of one week of each: Introduction; and Advanced (case studies, etc.).

A two-day course is available for managers. An engineer is expected to be proficient after using B for three months, that is, at the level of using the language. Matra uses this three-month period as an expected start-up cost. However, it is more complicated to learn formal proof -- the syntactical transformations require more training. Engineers also take longer to learn how to model using B, i.e., "how to use mathematics in a useful way". After three months, an engineer is integrated into projects, while receiving more training.

The RAISE/LaCoS experience is growing. In the first phase, Matra found training to be very expensive. Part of the problem was documentation, part was the price of a wide spectrum language. So far, Matra has had a positive story with regard to tools: tools were easy to use after learning RSL. They started with no tools and were concerned with which style and level to work at within RSL. The problem was that the documentation is heavy on syntax but the trainee does not gain a strong intuition of RSL semantics. Misunderstandings of the constructs are encountered frequently, which is of concern because "a specification language should be simpler than a programming language."

LaCoS/RAISE: No development projects have yet begun to use RAISE, only the methods/tools group as part of the LaCoS evaluation effort. The evaluation phase has been 1.5 years with parallel experimentation starting now. They expect to do a parallel development with some of the projects using B. The aim is to use B at the system level, both for comparison with B and for additional capabilities that they feel B lacks (modelisation strength).

Matra feels it is still too early to have measurements; indeed, this seems true across LaCoS. The tools available (not including the Justification Editor) are considered to be simple. So far, the content of the LaCoS project has been more on evaluating the language than doing applications. Interesting parts include:

- concurrency modelling as a mixture of Calculus of Communicating Systems (CCS) and Communicating Sequential Processes (CSP)
- modularisation schemes (where intuition was a particular problem)
- refinement

Because of mixed styles, a specification could look familiar but actually be meaningless. (Mixed styles means sometimes imperative, sometimes declarative, e.g., Lambda expressions for concurrency are troublesome because it is not always possible to tell which model you are in: CSP or CCS.) The justification tool is not yet available and is anticipated to add new dimensions to the evaluation.

LaCoS at Matra proceeded first bottom-up, working on components and "management units" (the basis for the train protection system structure). This gave low-level specs that were felt to be really too low level. They then worked on more abstract system level models (as well as going to Ada code). They wanted to look at specific properties, e.g., concerning the stop point prediction. It was easy to express this in a physical model of the system, including the train, tracks, etc. This is a picture of what a person looking at the system might see. This includes train movements and the pictures at time *t*, *t+1*, etc. In addition to histories, the specification is written as state based models. But the detailed software picture is expressed in terms of cycles and data.

There is no formal connection between the RSL description of the system, the software spec, and the Ada program. This is where an animation tool would be used.

5.3.2.4 Results (Preliminary)

- RSL seems to be good for modelling.
- Impose formal methods early, at system level.
- Formal methods help because you are forced to completely define and understand consequences (which has not been true of other methods they have tried).
- Size/length of RSL hinders formal proving (among other problems owing to its being a wide-spectrum language).

5.4 Evaluation

5.4.1 Background

Product:

Technology transfer experience base focused on formal methods and safety-critical commercial process control systems.

Industrial Process:

ESPRIT funds collaborative projects (50-50 shared funding) to pursue technology development and application, here the LaCoS project. Lloyd's Register is expanding its long-existing business base in shipping system development and assessment into the areas of safety-critical process control system. The Lloyd's Register strategy for building its expertise is through participation of a dedicated group in multiple collaborative projects. Matra Transport is also building its experience in RAISE and other formal methods with target of satisfying the RATP authorities.

Scale: LaCoS is a five-year project, now in its second year. The experiments that form the basis for this case study are of a few months duration, although the products under development are multi-year. LaCoS includes nearly a dozen partner organizations.

Formal Methods Process:

RAISE is made available and supported through consultancy. Experiments are conducted using the RSL wide-spectrum language, the RAISE method manual, and a suite of tools that primarily support management of language objects. Both groups are studying other formal methods, primarily B.

Major results:

The two organizations and several applications studied did not announce any major results. These cases did provide insight into some technicalities of the RAISE approach; the LaCoS technology transfer experience, particularly in the collaborative funded ESPRIT industrialization context; and background for analysis of the application results and the RAISE experience.

Interview profile:

Lloyd's Register: 3 hours and lunch at Lloyd's Register House, Croydon, U.K. Ted Ralston, Susan Gerhart (author).

Matra Transport: 1.5 hours at Matra Transport, Paris, France. Ted Ralston and Susan Gerhart (author).

Background included the RAISE references and a three-day course by CRI that the interviewers attended at MCC in March 1991.

5.4.2 Product Features

Client satisfaction	+	(1)
Cost	n/a	(2)
Impact of product	+	(3)
Quality	n/a	(4)
Time to market	n/a	(5)

(1) The Lloyd's Register and Matra Transport participants are satisfied that they are expanding their experience base through RAISE and other method, e.g., B.

(2) ESPRIT projects are a cost-effective way of transferring a technology. However, the LaCoS project is still in its early stages.

(3) Viewing the ultimate product as a Lloyd's Register consultancy, the LaCoS experience is a major contributor, with others being their increasing experience with the types of systems being built in the MORSE project. In this case, the formal methods intermediaries are the RAISE language and toolset, which are at a usable level with sufficient support.

Matra Transport is developing formal methods proficiency in order to satisfy the RATP client.

(4) Not Applicable.

(5) Not Applicable.

5.4.3 Process Features

General process effects

Cost	0	(6)
Impact of process	0	(7)
Pedagogical	+	(8)
Tools	0	(9)

Specific process effects

Design	+	(10)
Reusable components	n/a	(11)
Maintainability	n/a	(12)
Requirements capture	+	(13)
V & V	n/a	(14)

(6) Since the product is experience, learning the formal methods is the main consumer of resources, with the cost being at a reasonable level.

(7) It is too early to tell what effect the LaCoS experience will have on the Lloyd's Register and Matra Transport business strategies or the resulting systems.

(8) The structure of the LaCoS project and the extensive pedagogical materials available for RAISE make this an effective learning environment.

(9) Tools are available but are not yet playing a major role in the evaluation.

(10) New design techniques are being learned, as codified in the RAISE approach.

(11) Developing reusable specification components is recognized as important, both for reusability and for assessment of RSL's modularity capabilities. However, insufficient information currently exists.

(12) This seems not to be a concern for the applications and contexts we examined. Getting the system right at the start is more important.

(13) Considerable experience is building up through the use of RSL for modelisation and the interaction with clients.

(14) Proof experimentation is just beginning. It is too early to tell.

5.4.4 Observations

This case is different than the others, which are more focused on the applications themselves. Here, the applications are the basis for improving the RAISE approach and increasing expertise in the partners. This case provides information about a technology transfer model from which major lessons will be available in the next few years.

The LaCoS company partners are staffed by experienced, well-educated professionals. These individuals have the sophisticated computer science background to learn the complex wide-spectrum language approach. However, these individuals are finding considerable difficulty with some of the language style options of RSL. They also find themselves not understanding certain language features, e.g., concurrency. This suggests that RAISE may have an essential steep learning curve that will prohibit many safety-critical systems groups from learning it. However, if the mastery of RAISE does indeed yield productivity and quality results sufficient to convince authorizing agencies, there may be a strong competitive advantage to having been through the LaCoS project.

Feelings are surprisingly neutral about the RAISE toolset. One would expect stronger feelings about the editors, proof assistants, and other system support. Perhaps this is due to lack of experience. However, it appears that the main vehicle for transfer of RAISE is not the toolset but the RSL language and methodology.

The applications chosen are solid and important; therefore, the learning experience is distributed between the RAISE approach and the application domain itself. This adds to the value of the LaCoS technology transfer approach and further justifies the time spent learning a formal method. In other words, the application matters, so learning time is justified. However, the learning experience is also subsidized 50% by ESPRIT.

It will not be easy for other formal methods groups to assimilate the LaCoS experience. Background in the RAISE approach is essential and the applications are difficult to appreciate. Therefore, the general lessons about formal methods effectiveness and the value of the LaCoS technology transfer model must be carefully defined. The experience covers more than just RAISE, with both groups evaluating and experimenting with the B method and tool.

5.4.5 Formal Methods R & D Summary

5.4.5.1 Methods

Specification

Lloyd's Register and Matra Transport have experimented with various ways of using RSL, finding it a challenge. Separation of concerns between a good requirements model and a top-level specification for refinement is seen as very important. Indeed, there is a term for this top-level concern: modelization.

Design and Implementation

Design in RAISE is still a nebulous process. Emerging more clearly is the role of "justification" and the technological and other means of achieving it. Another partner, Technisystems, had a good experience taking RSL to Ada for their application. Formal connections with code are a major concern for both partners.

Validation and Verification

However, experience with a Justification Editor for a complex language such as RSL is just beginning. There may be major barriers or a significant advance may be made in achieving reliable and justifiable implementations. The methods are embodied in the proof rules, which also require an assurance process.

5.4.5.2 Tools

Language processors

The RAISE toolset uses a very sophisticated language processing system, the Cornell Synthesizer Generator. This appears to make it possible to perform difficult semantic tasks and to control representation of the language. A syntax-directed editor is a red herring as far as formal methods productivity is concerned.

Automated Reasoning

The RAISE developers decided to build their own theorem prover (a justification editor) so that it more closely mirrored the RAISE language design principles. It is expected that simplification and other requirements will be defined through experience and then inserted as the Justification Editor technology advances. A Confidence Condition Generator is another tool felt necessary but not yet fully available.

Other tools

Configuration management is supported within RAISE and becoming less built-in as other platforms become available. Considerable attention was paid to the objects involved in process, e.g., documentation and versions. No reports were received of experience in large scale applications, however.

5.4.5.3 Recommendations to the FM Research Community

1. Compare the architecture of the RAISE toolset with theorem provers such as EVES. What tools have been identified as necessary? How are they inter-linked? Consider the Confidence Condition Generator and the Justification Editor.

2. The RAISE methodology attempts to cover the full life cycle. How much of the methodology

applies to other notations? How well does RSL support the methodology?

3. The wide-spectrum language approach is carried through in RSL. How does that compare with other formal methods approaches?

4. The concurrency model of RSL (derived from both CCS and CSP) is presenting problems for system designers. What is the essential difficulty?

5. Many formal methods courses are being developed by the LaCoS partners. That material may be usable by other methods or establish a baseline of educational requirements.

6. What metrics would allow comparison of RAISE with other formal methods approaches, e.g., to characterize complexity of specification?

5.5 Conclusions

The LaCoS project at both Lloyd's Register and Matra Transport is primarily an evaluation and broadening project. In both organizations, RAISE is being evaluated as an alternative or complement to a major competing approach, B. Education and training are in-place at a significant level. Application experimentation is just beginning, with Lloyd's Register further along in its targeted applications.

The experience, so far, is that RSL is quite complicated and tricky. The methods/tools teams are strongly interested in the system level modelling issues and have developed a respect for the use of RSL at this level. They are also, but to a lesser extent, concerned with derivation and accountability from requirements through code. The substantive toolset behind RAISE is used but has not yet been shown to be either a barrier or a significant assistant. Considerable promise is placed upon a forthcoming Justification Editor as a means of both controlling the language semantics and exploring proof strategies.

In our view, the LaCoS project (at the two-year stage) is achieving its objectives of training its partners in the RAISE method over a planned period of several years, with partners concentrating on their own applications and sharing results. The long-term durability of RAISE is still unclear: its complexity is recognized but a considerable workforce at CRI and in the partners is interested in mastering RAISE. Should that mastery, difficult as it may be, lead to groups with significantly higher ability to control product quality, ESPRIT may also achieve a considerable competitive advantage from its formal methods investment in general and the specific LaCoS partnership. It could take any outside formal methods group several years to attain the same level of capability gained within LaCoS. However, should RAISE founder in its theorem proving capability, a considerable simplification of the approach may be required. Technically, the RAISE project puts the wide-spectrum specification approach on the line for evaluation.

The significance of this case in the context of this study is that it lays the groundwork for our future understanding and assessment of the RAISE/LaCoS project outcome and for expansion of the LaCoS transfer experience.

6 MULTINET GATEWAY

6.1 Case Description

The Multinet Gateway System (MGS) application is the major computer security-related case study of our survey. The MGS consists of gateways and networks. It is an internet device that provides a protocol based datagram service for the delivery of datagrams between source and destination hosts (see diagram). In addition to the delivery service, the MGS provides security mechanisms (in consonance with U.S. Department of Defense security criteria) to prevent compromise of sensitive information.

A main aspect of the MGS project was to achieve a certification from the U.S. National Computer Security Center. In December, 1985, the U.S. Department of Defense published a set of criteria [4] through which security-related products could be measured against a hierarchy of evaluation classes. The higher the evaluation class, the more confidence one could have in the product's security controls. There are four divisions of evaluation classes: D, C, B, A, ordered hierarchically, with the highest division (A) for systems achieving the most comprehensive security.

••••••••••••••••••••

Add diagram

••••••••••••••••••••

Figure 5. The Multinet Gateway System as an Internet

Quoting from the criteria [4], a broad description of the classes is as follows:

- Division D: Minimal Protection
  This division contains only one class. It is reserved for systems that have been evaluated but that fail to meet the requirements for a higher evaluation class.

- Division C: Discretionary Protection
  Classes in this division provide for discretionary (need-to-know) protection and, through the inclusion of audit capabilities, for accountability of subjects and the actions they initiate.

- Division B: Mandatory Protection
  The notion of a TCB (Trusted Computing Base) that preserves the integrity of sensitivity labels and uses them to enforce a set of mandatory access control rules is a major requirement of this division. Systems in this division must carry the sensitivity labels with major data structures in the system. The system developer also provides the security policy model on which the TCB is based and furnishes a specification of the TCB. Evidence must be provided to demonstrate that the reference monitor concept as been implemented.

- Division A: Verified Protection
  This division is characterized by the use of formal security verification methods to assure that the mandatory and discretionary security controls employed in the system can effectively protect classified or other sensitive information stored or processed by the system. Extensive documentation is required to demonstrate that the TCB meets the security requirements in all aspects of design, development and implementation.

The MGS project targeted Division A (specifically, level A1) as the division at which they hoped to be certified. The formal methods aspect of the project was in the use of mathematics and the Gypsy Verification Environment (GVE) [5] to elucidate a formal security policy model and to prove that the model satisfied various security properties.

6.1.1 Product and Process Profile

Size: 10 pages of mathematics and 80 pages of Gypsy specification.
The underlying operating system was about 6,000 lines of code.

Stages:

1981-4 Design and refinement phases.
1984-5 First development models built (for both hardware and software components).
1985-8 Certification contract. The contract resulted in the development of the "Advanced Development Model" and supporting evidence.

6.1.2 References

1. ``Multinet Gateway --- Towards A1 Certification,'' George Dinolt, Peter Baker, Richard Neely and James Freeman, IEEE Symposium on Security and Privacy, 1984.

2. ``A1 Assurance for an Internet System: Doing the Job,'' P.C. Baker, G.W. Dinolt, J.W. Freeman, M. Krenzin and R.B. Neely, 9th National Computer Security Conference, September 1986.

3. ``An Internet System Security Policy and Formal Model,'' J.W. Freeman and R.B. Neely, 11th National Computer Security Conference, October 1988.

4. ``Trusted Computer System Evaluation Criteria,'' DoD 5200.28-STD, U.S. Department of Defense, December 1985.

5. ``Gypsy Verification Environment User's Manual,'' Robert L. Akers, Bret A. Hartman, Lawrence M. Smith, Millard C. Taylor, William D. Young, Computational Logic Inc. Technical Report 61, September 1990.

6.2 Interview Summary

1. Date: 19 March 1992 (1 day)
   Respondent: George Dinolt
   Organization: Loral Western Development Labs, San Jose, California
   Description: Interviewed by Dan Craigen (author) and Susan Gerhart

2. Date: 5 June 1992 (1 hour)
   Respondent: Jim Williams
   Organization: Mitre Corporation, Bedford, Massachusetts
   Description: Interviewed by Dan Craigen (author) and Susan Gerhart

Note 1: All the Multinet Gateway System (MGS) work was performed while the people involved were with Ford Aerospace. After the project was completed, all of Ford Aerospace (FACC) was sold to Loral.

Note 2: The material presented in this section has been augmented with information extracted from the references.

6.2.1 Organizational Context

The portion of Ford Aerospace involved with the MGS was primarily involved in defense contracting and had expertise in networking and C^3 I. Much of the MGS work was performed at Ford Aerospace's Colorado Springs operation. (Dinolt commuted between San Jose and Colorado Springs.)

Generally, Ford Aerospace consists of individuals with BA, BSc, BEE, etc., degrees. Ford Aerospace operated on a matrix management organization and generally adhered to government-specified methodologies to develop their products.

6.2.2 Project Context and History

The educational background of those involved in the formal methods portion of the MGS was higher than the norm at Ford Aerospace. There were three Ph.Ds and one individual with no degree (but with extensive experience). As to learning new technologies, the group adopted formal methods on their own. With respect to transition issues, a Gypsy Verification Environment course was given to MGS implementers. It was a disaster; some of the students were unfamiliar with the concept of a set.

The MGS is an Internet device that, in conjunction with other Internet components, provides a datagram service. In addition to the datagram service, it provides security protection mechanisms to prevent the compromise of sensitive information. The overall value of the MGS contracts was about $12 Million (U.S.) with about 2/3 of the total expended during the certification phase (see above). During the certification contract, three individuals worked full-time on the formal methods aspects of the MGS.

6.2.3 Application Goals

A major application goal was to develop the MGS and to achieve A1 certification.

A number of novelties were associated with the MGS effort. While there had been some experience with formal methods within Ford, they certainly had not been applied to networks. While Ford had experience with networks, the security requirements were a new wrinkle. In addition, this was a first attempt to distribute protocols over separate processors and to use micro-computer boards.

In some senses, the MGS security policy model can be simply described. While there are various security assertions, the primary three assertions were:

> is_securely_accepted: the system accepts data only from input wires and only if the data is consonant with the security level(s) of the input wire.
>
> is_securely_derived: the system delivers data only to an output wire only if it is "derived" from data received at input wires. In addition, the security levels of this data must be the same.
>
> is_securely_delivered: the system delivers data only to output wires and only if the data is consonant with the security level(s) of the output wire.

A form of genericity is achieved by leaving certain definitions unspecified and having the definitions instantiated on a case-by-case basis. Hence, with respect to the MGS, they explicitly instantiated the definitions and showed that the MGS satisfied the policy.

6.2.4 Formal Methods Factors

In [2], the authors describe four development mechanisms that they used in the development of the MGS:

> Trust domains: A rigorous approach to describing networks. Trust domain analysis provides an abstract language to describe the partitioning of networks into nodes and links.
>
> Constraint monitors: A means of identifying the minimum constraints necessary to maintain the security of the MGS.
>
> Gypsy Verification Environment: A tool that supports the specification and verification of assertions.
>
> Documentation: An integrated means of conveying the MGS design in a consistent and understandable manner.

The MGS team spent over a year determining what formal methods to use. Included in their investigation were InaJo, Boyer-Moore, Affirm and work by SRI International. They chose the GVE as it permitted code proofs and was supported by the National Computer Security Center.

6.2.5 Formal Methods and Tools Usage

The MGS development attempted to adhere to standard DoD procedures where possible. They used Yourdon-like design methods, data flow diagrams, Program Description Languages, and walk-throughs.

There was no Independent Verification and Validation, though there was close analysis of the formal methods and security aspects of the MGS by Mitre, the National Computer Security Center and Computational Logic, Inc.

One of their main difficulties was integrating formal methods with the standard lifecycle and developing a documentation tree that would be useful for certification. For example, in [2], they refer to the "System Security Description" that describes the system and security architecture as an implementation, a functional, a trust domain, and a Gypsy summary specification. The "implementation" includes physical locations of components, their interconnections, and allocation to software, firmware and hardware. The "functionality" document describes the kinds of functional processing and the associated relationships. The "trust domain" document describes the MGS security policy, independent of a specific formal specification language (and seems to have been used, in part, to hide the mathematics from those unfamiliar with Gypsy and mathematical notations). Finally, the GVE is used to provide a summary specification of the MGS, focusing on security attributes. All these descriptions are then shown to correspond informally.

The MGS development progressed in two phases. In the first phase, they used formal methods and prototyping to develop their understanding of the MGS. In the second phase, they implemented the system, used mathematical modelling and used the GVE. The formalism identified 50 axioms that were to be true of the MGS implementation for it to meet the security requirements.

The tools used by the project team included Source Code Control System, GVE, and various UNIX tools to help manage, manipulate and coordinate the source code and Gypsy. Part of the motivation for using the GVE was to handle the complexity of the system through the use of bookkeeping. However, there were also some limitations imposed by the GVE; for example, the lack of generics.

The GVE was robust (within the perimeter of usage), complex and not of production quality. Ford contracted Computational Logic, Inc., to improve the GVE interface. Dinolt had concerns about the soundness of the GVE, but feels that they used a portion of the GVE in which they could have confidence. The GVE was not easy to learn; there was a steep learning curve.

6.2.6 Results

A model of security for the MGS was developed and, through the use of formal methods, they were able to better understand the system and to achieve performance improvements. Dinolt has more confidence and assurance in the MGS than with other systems of similar size and complexity. While an A1 certification was not achieved (for reasons not relating to formal methods), they did achieve a "developmental evaluation"[2] and the MGS has been deployed. One problem they did run into was that communicating the formal ideas to the implementers was difficult. The implementers did not understand the precision and were used to have more latitude in interpreting functionality.

For the most part, Dinolt reported that they would not have proceeded much differently. He would like to have had more tool support and would be interested in studying the relationship of fault analysis with formal methods. He would also like to better understand how to integrate formal methods with standard development processes.

With respect to training and education of formal methods, Dinolt observed that before proceeding with a development, try to understand what you are modelling, express that understanding, and learn the ethos of the tools and methods that you are to use. Dinolt felt that integrating formal methods into an organization like Ford Aerospace is difficult, since they are a contract-driven organization and they must be conservative in how they respond to contracts.

6.2.7 Interview with an assessor: Jim Williams

Jim Williams is a long-term employee of the Mitre Corporation, Bedford, Massachusetts, who has had extensive experience in computer security and formal methods. Williams was on the team that assessed the MGS with respect to the TCSEC. Our interview with Williams was to complement our interview with one of the developers (Dinolt). The U.S. National Computer Security Center and Computational Logic Inc. (which includes developers of the GVE) were also involved in the assessment process. The team that assessed the MGS had difficulties doing so; up to this development, they had not experienced assessing a system of similar or greater complexity.

Williams reported that the initial specifications of the security model were generally unreadable by the assessors. However, after a one day meeting that was held at Computational Logic Inc., Ford Aerospace rewrote the specification and developed a much better model. Williams also reported that there were some problems in using the GVE. For example, Ford had to develop an "extractor" that determined the minimum information required to prove a putative theorem. Some distortions to the security model resulted from trying to express it in Gypsy. Interestingly, Williams found the formal Gypsy specification to be more intuitive than the English presentations.

The proofs were of an information flow characteristic and security requirements down to the level of a CPU were sufficient for proving the network security property. However, Williams' was not convinced that the effort in using the GVE prover brought much additional benefit to the development. At no time was there an instance of a good specification that had a false theorem. Williams did note that there was a flaw in the specification approach that disallowed the proof of a relevant security property. The security property was demonstrated by other means (static flow analysis).

---

[2] There are two phases of an evaluation: developmental and formal. In developmental, the government works with the contractor to help meet the evaluation criteria. The formal evaluation usually consists of a new government team that performs the actual evaluation, which may result in a certification.

Williams' main lessons from the experience was that one should chose an expressive specification language and perform informal proofs. Expressibility is desirable since it reduces the likelihood of introducing strange distortions to the specification.

6.3 Evaluation

6.3.1 Background

Product:

The Multinet Gateway System (MGS) is an internet device that provides a protocol based datagram service providing service for secure delivery of datagrams between source and destination hosts.

Industrial Process:

Certified against U.S. TCSEC [4] and included tempest and communications security analysis. Use of rigorous mathematics and the Gypsy Verification Environment to develop and model security functionality. Overall development consistent with DoD standards. No Independent Verification and Validation, though the security model was evaluated by U.S. government assessors.

Scale:

From the formal methods perspective, 10 pages were needed to describe the security model and a further 80 pages for presenting the Gypsy specification of the MGS. The underlying operating system was about 6,000 lines of code. Equivalent of nine person years of effort over a period of three calendar years.

Formal Methods Process:

Use of rigorous mathematics to model security aspects of the MGS. The Gypsy Verification Environment was used to mechanize the mathematical descriptions and to prove security properties.

Major Results:

Development and fielding of the MGS. While an A1 certification was not achieved, the MGS is being used by the DoD and received a "developmental evaluation." Successfully developed an approach for handling multiple protocols, while maintaining acceptable throughput; and elucidating techniques for modelling and implementing security concerns within an Internet framework.

Interview Profile:

One day interview with George Dinolt (a developer of the MGS) and a one hour interview with Jim Williams (an evaluator).

6.3.2 Product Features

Client satisfaction	+	(1)
Cost	n/a	(2)
Impact of product	+	(3)
Quality	+	(4)
Time to market	n/a	(5)

(1) While the MGS did not achieve an A1 certification (probably because the MGS was not viewed as a widely available commercial product), the U.S. government seems to be generally satisfied with the development. Both the developers of MGS and the assessors feel that the system is of high quality and that their confidence is based, partly, on the clarifications arising from the formal methods efforts.

(2) Contractual effort.

(3) MGS has been fielded and is being used in sensitive contexts.

(4) Both the assessors and the developers feel that the product's reliability was enhanced by the application of formal methods.

(5) Contractual effort.

6.3.3 Process Features

General process effects

Cost	n/a	(6)
Impact of process	0	(7)
Pedagogical	+	(8)
Tools	0	(9)

Specific process effects

Design	+	(10)
Reusable components	+	(11)
Maintainability	n/a	(12)
Requirements capture	+	(13)
V & V	+	(14)

(6) Contractual effort.

(7) We did not determine any long-term effects of process resulting from the MGS contract.

(8) Effort resulted in new approaches for describing security functionality of communication networks. Substantial exercise in applying the U.S. TCSEC to a significant application.

(9) William's commented that the GVE proofs did not add to his understanding of the application domain. In addition, there was resource limitations with the GVE that had to be worked around. Source Code Control System, GVE and various Unix tools were used to manage, manipulate

and coordinate source code and Gypsy.

(10) Dinolt felt that the use of rigorous mathematics helped to clarify the underlying concepts and resulted in well-modularized system. The developers were able to identify "constraint monitors" within which the integrity of the security functionality was embedded.

(11) The MGS resulted in the development of a reusable security policy model that could be applied to other Internet devices.

(12) We did not ask any questions pertaining to maintainability.

(13) The formalism helped to clarify the security requirements for networks that are to be in consonant with DoD security policies.

(14) The formalism helped to raise the confidence of the assessors and the developers of the reliability of the MGS. Dinolt specifically mentioned that his confidence in the MGS is higher than with other systems of similar complexity. In addition, the Verification and Validation team were able to use the formal specifications to generate tests that uncovered coding errors.

6.3.4 Observations

The MGS development is one of the major computer security applications of formal methods carried out to date. In this project, they used mathematics and the Gypsy Verification Environment to express their understanding of the security requirements of the MGS and then to demonstrate in a mechanised manner that the MGS met the security requirements. The use of formal methods was mandated by their contract and is mandatory for systems aspiring to the A1 level of evaluation. Transition of formal methods technology into Ford Aerospace, in general, did not occur. Use of formal methods was a contract-driven requirement and was not viewed as a fundamental technology for Ford Aerospace.

The case study is suggestive of the influence of regulatory practices on using formal methods for critical systems. Much of the expenditures was related to the "certification contract" and it seems apparent that formal methods would not have been used if not required by contract. On the other hand, having used the technology, it appears that Ford Aerospace (and the assessors) have developed a deep understanding of the MGS that has resulted in the deployment of the system.

There appears to be some disagreement on the value of using the Gypsy Verification Environment for proof. Williams reports that the proof efforts never raised problems with the security specifications.

6.3.5 Formal Methods R & D Summary

6.3.5.1 Methods

A rigorous graphical notation was used to specify much of the functionality of the MGS and to communicate the ideas to those who were unfamiliar with Gypsy. Learning Gypsy and the ethos of the GVE was not easy. From the production perspective, there were concerns about the soundness of the GVE (especially the theorem prover) and expressibility constraints arising from the language (e.g., lack of generics).

6.3.5.2 Tools

Primarily the Gypsy Verification Environment. Ford Aerospace found limitations of expressibility with the language and had to build an extractor tool so that only the information needed for a proof was resident in the Gypsy environment. Uncertainties pertaining to soundness were voiced. Computational Logic, Inc., was also contracted to improve the interface. On the other hand, the tool was found to be robust (within the perimeter of usage).

6.3.5.3 Recommendations to the FM Research Community

This case study suggests that issues of expressibility and soundness are of importance in production developments. Interface issues are also worthy of concern. A tool with a poor interface can be exceedingly difficult to use. In a manner analogous with the development of good notations which are understandable to the potential user communities of a formalism, system interfaces must result in the capabilities of the system being accessible. It appears that there has been little advance, over the past decade, in the style of interface with the formal methods tools. In addition, tools developed by the formal methods community must be able to handle large and complex problems. Communication of the formal ideas to project participants who are not as knowledgeable about the underlying mathematics is a problem that needs to be better considered.

6.4 Conclusions

This case study is a significant exemplar (in terms of both process and product) of the application of formal methods to a security-critical application. It is also demonstrative of the potential influence of regulators in the use of formal methods.

Formal methods, both in the rigorous and mechanized use of mathematics, were successfully applied to this project; though there were difficulties. A clear understanding of the security requirements for the MGS was developed and both formal and informal proofs were produced. The GVE tool enforced the use of a particular syntax and semantics, and, in conjunction with other tools, helped in handling complexity. On the other hand, the GVE hindered some aspects of the project as contortions to the specification, concerns about soundness and tool limitations, all impacted the work. Various work-arounds were necessary. Communication problems between the formal methods team and the implementers also arose.

From a transition perspective, while the formal methods team is rather better informed on the advantages and disadvantages of formal methods and the application of the GVE, there was no general dissemination into Ford Aerospace. Contracting and personnel realities (e.g., lack of mathematical sophistication) mitigate against a general acceptance of the technology.

7 SACEM - A RAILWAY SIGNALLING SYSTEM

7.1 Case Description

The product developed is a certified safety-critical railway signalling system which reduced train separation from 2 minutes 30 seconds to 2 minutes, while maintaining safety requirements. The successful deployment of the signalling system has removed the need for building a new third railway line. The developers of the signalling system were required to convince the Paris RATP (the Paris rapid transit authority) that the system met safety requirements. Amongst the numerous techniques used to demonstrate system safety (e.g., fault analysis and simulation) were B and Hoare Logic (descriptions of B and Hoare Logic can be found in Appendix B of Volume 1).

The system consists of 9000 lines of verified code and 120,000 hours of formal methods effort. The new system allows for 60,000 passengers to be carried per hour.

7.1.1 References

1. G. Guiho, C. Hennebert, "SACEM Software Validation". *International Conference on Software Engineering*, 1990.

2. C. Da Silva, B. Behbonei, F. Mejia, "Formal Specification in the Development of Industrial Applications: The Calcutta Subway", April 1992, submitted for publication.

3. J.-R. Abrial, et al, "The B Method", BP International Ltd., 1991.

4. "Introducing the Encoded Processor", GEC Alsthom.

7.2 Questionnaire 1

Project: SACEM
Respondent: Mr. L. Lugand
Organization: GEC Alsthom Transport, Signalling Division
Response date: April 9 1992

A. Organizational Context

SACEM has been developed by a consortium of three companies: GEC Alsthom, Matra Transport, and CSEE (Compagnie de Signaux et Entreprises Electriques), in which GEC Alsthom was the leader. This project had two customers: SNCF [Societe Nationale de Chemin de Fer] (the French National railways) and RATP (the Paris transportation authority). (Footnote: The initial consortium was Jeumont-Schneider, Interelec and CSEE, respectively. Industrial takeovers and mergers change the names of the companies, but not the teams involved in SACEM.)

B. Project Context and History

SACEM means "Driving, Operation and Maintenance Aid System". It is primarily an automatic train protection (ATP) system. It has been designed from scratch for the RER (regional subway) Line A. This line is the most overloaded subway line in Paris. It was operated under manual driving with conventional trackside signalling. The main purpose of the system is to reduce the train interval from 2min30s down to 2min. This provides a 25% increase in throughput, thereby saving the construction of another line. The significant milestones are:

early 1982	start of the project
end 1985	prototype completion
Aug. 1988	commercial operation at 2min30sec interval
May 1989	commercial operation at 2 min interval (full capacity)

The project has cost 170,000 man hours for the prototype, and another 145,000 hours for the commercial systems. All the SACEM hardware and software was entirely new.

C. Application Goals

SACEM had two fundamental goals: increase operation safety and increase the line throughput. Safety had an essential influence on the development methods used. Line throughput increase was a fundamental functional requirement.

D. Formal Methods Factors

SACEM was the first French railway system where software has a safety-critical role. Formal methods were chosen to enhance the credibility of the software validation.

At first, Hoare's proof of program method was used: there was no other method available in our knowledge in 1982. Later, Abrial's B method was used.

E. Formal Methods and Tools Usage

Hoare's method: manual writing of pre- and post-assertions associated with each procedure.
B: a posteriori re-specification (no tool) control of the validity of Hoare's pre- and post-assertions.

F. Results

Hoare's method has been found efficient for discovering bugs, some of which were safety-critical. It was very cumbersome. B has confirmed the validity of Hoare's method.

Main lessons learned: Using directly the specification refinement tool during the development could be more efficient.

Trade-offs: No savings were made on testing.

The formal proof was not driven back to the highest level, but was limited to the main functions.

G. Additional Information

SACEM was followed by two other projects, described below:

7.3 KVS

A. Organizational Context

KVS is a project developed by GEC Alsthom for SNCF. This is actually a revamped version of a speed control system (KVB = beacon speed control) being installed at SNCF. The overall market calls for the equipment of all the electrified lines of the SNCF (around 8,000 km) and of the trains running on them (around 4,000). About half of this will be delivered in the KVS version.

B. Project Content and History

KVB/KVS uses beacons placed between the rails to send the speed information of the conventional trackside signalling to the train. The on-board equipment checks the speed in real-time and generates an emergency braking if the train speed is excessive. KVB/KVS is a "safety net" for trapping human driving failures.

Major milestones (KVS):

end 1988	start of project
now	under test
end 1992	commercial operation

The effort (excluding all hardware development) is around 30,000 man hours.

C. Application Goals

KVS uses the same principles as KVB and is compatible with it. However, it uses more modern hardware, and the software has been redesigned.

D. Formal Methods Factors

KVS has been used as a testbed for the application of the B method. The experience gained with SACEM indicated that this method is cost-effective and does not need to be confined to safety-critical systems, because the software is easier to validate. Formal methods are seen as a safety argument for export markets.

E. Formal Methods Usage

Method: B method for software specification and development.
Tools: Under industrialization by BP Research (UK) and GEC Alsthom.

F. Results

The quality of software developed with formal methods is good (no major problems found). It is an efficient method (but metrics were not used). Tool improvements are still needed: part of the proof was sacrificed in order to meet the deadlines. Formal methods are now part of the GEC Alsthom's software strategy.

7.4 CTDC Calcutta

A. Organizational Context

CTDC Calcutta is a system developed by GEC Alsthom for MRC, the Calcutta transportation operator.

B. Project Content and History

The CTDC Calcutta (Controlled Deceleration Train Control) is a subway ATP. CTDCs are in operation on the subways of Montreal, Marseilles and Belo Horizonte. These latter systems are hard-wired. For Calcutta, a major architecture change was decided, because of the hardware obsolescence and of the high costs of the unavoidable adaptations. The CTDC Calcutta uses SACEM's hardware, plus a few new boards. The CTDC principles are mostly unchanged. The software is of course entirely new. The safety principles (Vital coded processor) are that of SACEM. The milestones are:

end 1990	decision to adopt a numeric solution
end 1991	end of development
March 1992	prototype operation in Calcutta

C. Application Goals

Adapting the existing hardware was deemed more expensive than rebuilding the system in software. Cost, safety and time were the prime concepts.

D. Formal Methods Factors

The method used was the full B method.

E. Formal Methods Usage

The software has been entirely specified, refined and proven with the B formal method, with the latest BP research/GEC Alsthom tools.

F. Results

The application of formal methods is a full success. The software generated was efficient and of good quality. No trade-offs were necessary due to the formal methods. Productivity will be estimated using man-hour sheets.

G. Additional information.

A paper has been submitted for publication.

7.5 Interview Summary

Date: April 9 and 10, 1992 (1.5 days)
Organizations: GEC Alsthom and RATP
Attending:
Pierre Chapront (Deputy Technical Director, Signalling Systems, GEC Alsthom)
Dr. Laurent Lugand (Manager, Reliability, Signalling Systems, GEC Alsthom)
Fernando Mejia (Formal Methods Manager, Signalling Systems, GEC Alsthom)
Dr. Gerard Guerin (Engineer, Transport Div., Signalling Systems, GEC Alsthom)
Jean-Paul Rubaux (Direction Generale, Ingenierie Generale Technique, RATP)
Interviewers: Ted Ralston and Susan Gerhart (author)

7.5.1 Organization

RER is one of three train systems operating in the Paris subway system. RATP is the Paris transport authority which commissions trains and related systems and is responsible for certifying their safety and operating effectiveness. GEC Alsthom is a private company that builds trains, train subsystems, transformers, power stations and a variety of large electrical motors and related systems. GEC Alsthom was the lead contractor in the SACEM project (other partners are Matra Transport, and CSEE).

The SACEM project built a "driving, operation, and maintenance aid system" which is what SACEM means in French, for Line A of the RER.

7.5.2 Project Context

History: SACEM started at the conceptual stage in 1978, but didn't seriously get underway until 1982. The main problem was twofold: first, to relieve congestion on the two lines of the Paris subway by building a third connecting line and shortening the separation time between trains in order to increase capacity by 25%. The requirement is expressed in terms of minutes, i.e., reducing the "headway" from 2:30 to 2:00. Headway means the separation time between two successive trains reaching a given location. SNCF and RATP had been exploring what to do about the rapidly growing congestion on the (at that time) relatively simple but highly congested N/S-E/W crosslines of the Paris subway, and had reached the conclusion that entirely new up-to-date microprocessor and embedded software technology needed to be used.

After several years of studies on the transmission of data, safety principles, and available technology, RATP selected two competing teams to develop the first phase: one from Matra and one from GEC Alsthom. Shortly thereafter, however (c. 1981-2) Matra, GEC Alsthom, and CSEE combined efforts in a joint venture to carry out the project, largely because their individual approaches were turning out to be the same.

Versions: The SACEM project, at the beginning, had two parts. The first was the realization of a prototype Automated Train Protection System (ATP), to be installed on 7 km of Line A, to be followed by a real system on the same 7 km stretch. The prototype was produced in 1985, and put on trials. Testing and completion of the "real" system, based on the prototype, was concluded in March 1986. RATP then spent some 18 months studying/verifying (more on this below) it prior to approval, and the first version was commissioned in August 1988 (this version at the 2:30 headway). A second version satisfying the 2:00 headway was commissioned in May 1989. This latter system was deployed on 20 km. of track and 200 trains (or 60K passengers per hour).

Follow-on Projects: SACEM was completed in 1989. GEC Alsthom has been commissioned to use their approach on two other systems since then. The first is a project called KVS for the SNCF which is a transponder-based speed control system to be installed on half of the entire SNCF system (8000 km of track throughout France and on 4000 SNCF trains). The second project is a simplified KVS-type speed control system for the Calcutta subway.

7.5.3 Application

Domain background: There are three general parts of the SACEM system: an Automated Train Protection (ATP) system; an Automated Train Operation system; and a maintenance unit. In addition, there are two other components, that, while not automated, are connected to the system: the emergency brake, which is activated when speed plus headway exceeds limits, and the driver, who is signaled by the SACEM ATP when there is a problem.

Prior to SACEM, no computer control system existed at RATP. The metro (not RER) was equipped with hardware consisting of analog sensors on the rails which sensed and transmitted speed data, and signal lights for indicating speed and other warning information. SACEM uses a special "encoded" microprocessor (a specially modified 68000 i series) with embedded software, a series of trackside advanced sensors and transponders which communicate with the SACEM computer, as well as the older conventional signalling system. The SACEM ATP computer calculates a value for the speed/position of the train relative to the preceding train which is compared to a pre-defined safety curve, outside of which the train is unsafe and the brake is applied to bring it under the safety curve.

The main functions performed by the SACEM ATP are *Localization* (calculated from data received from the train's tachometer which is connected to the train's wheels); *receive/transmit* from trackside sensors (involves interaction of both SACEM sensors and older conventional track sensors); *speed monitoring*; and *cab signalling*. SACEM software consists of 15,000 lines of Modula 2. There are 500 fixed parameters, and only a few variables. Variants or invariants (in the sense of data) are expressed in an 80 bit word (ELEMENT). Message size can vary from 2 to 8 elements, and represents sections of the track. Invariant sequences may be interrupted by signals which are received every 0.3 seconds from sensors 2 km ahead of the train. The message structure can vary depending on the track situation. About 50 lines of code are special module used for trackside monitoring software.

An important requirement was that SACEM must be able to control different types of trains with different braking characteristics, etc. There were two types of trains in the RATP. SACEM must support trains that are unequipped or have failed.

In summary, all protection is in software using standard microprocessors following a data encoding and self-checking scheme devised for SACEM. The science and technology for this was maturing at the same time as SACEM.

In Calcutta, each section of track has predefined speed limits. The train has to calculate its speed and adjust the speed to those of segments ahead. The only information is a timer and velocity -- if the train velocity is not respected then the emergency brake is applied. The Calcutta client found that they had to accept a change from hardware to software and were sensitive to the mathematical approach of GEC Alsthom. The certification of proofs is not yet done. KVS is also a speed control system used with conventional signalling. The beacon has to maintain a database to understand not only the speed limit but the distance to the next beacon, as well as spacing information if there is a train ahead.

The KVS system is not fail-safe; the driver gets very little information and remains fully responsible. The Calcutta and SACEM cab displays are not fail-safe, but the ATPs are fail-safe, and will trap any display or driver-error.

Approach to developing SACEM: Three teams were used: design, implementation, and safety. RATP required a two-level approach in their validation/verification, using simulation in addition to the formal proofs. They were concerned about copying code into the validation process, so they also required B to be used on the proofs done using the Hoare method (see below on formal methods factors).

The RATP validation process: RATP, as the client, divided their validation into three parts: (1) validation of the operational need; (2) verification and testing; and (3) fitting and operations and maintenance. The activities associated with these three parts are diagrammed below.

Validation of Operational Needs (natural language)	Verification and Test	Fitting and Operation & Maintenance
A1 Simulation	**A2** Validation of software architecture	**A6** Layout check
	A3 Validation of model	**A7** on-site tests
	A4 Validation of hardware and software safety interfaces	
	A5 Safety studies and Hazard Analysis	
		A8 Dossier of SW Certification
		A9 National Commission on Safety review and approval

In brief, the explanation of the above diagram[3] is as follows:

A1: The operating need (or high level requirement) was expressed in natural language. This was transformed into a Finite State Machine representation using Grafcet (a standard finite state automaton tool) and simulation was used.

In **A2:** The software architecture for SACEM was represented in Structured Analyses Design Techniques and validated using Verification and Validation in accordance with Structured Analyses Design Techniques. A different team (**A3**) modeled the SACEM system, performed tests in the lab, and did simulations using ADA (the French version of Structured Analyses Design Techniques with Finite State Machine). The encoded microprocessor was validated by testing only in **A4**. Safety studies were performed using a French version of Failure Mode and Effects Analysis, and hazard analysis using Fault Tree Analysis. Operation and Maintenance were carried out in **A6** and **A7** through physical checkout of the layout of the trackside systems and on-site tests. Lastly, **A8** and **A9** were the certification steps. A ten-person committee was convened especially for SACEM (since nothing like this had been done before) which included members from SNCF (the French national railway), RATP, the French FAA, and academic and research experts on safety. A Dossier on Software Safety was produced.

Specific Validation Activities: 600 operational scenarios were devised and checked for completeness. These were elaborated into a Finite State Automaton and checked for coverage at the transition level. The RATP team found many specification anomalies through their testing. The main problem was in specifying degraded modes. The real-time simulation checked the interleaving of cycles and that

[3] AEEL = Analyse des Effets des Erreurs du Logiciel = Software Errors Effects Analysis

variables used in one cycle had been produced properly in a preceding one. This was all done by tests.

The RATP wanted to "validate the validation" by re-specification at the end. This led to the use of B to obtain pre-post conditions (see more on formal methods factors below). 15-20 differences were found. The re-validation also served to validate their *method* as well as their *results*. See the formal methods process discussion below.

The most significant problems in the specs were on the application side, where they did not know what to say about SACEM behaviour. The RATP found the formality of the specification helped to control its coherence. User communication was good when they were writing the spec but that was late in the process. They would have liked direct animation of the spec.

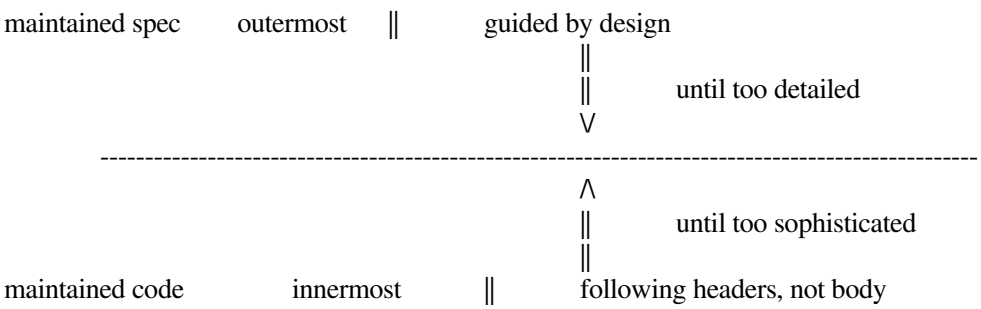
7.5.4 Formal Methods Usage

Criteria: Chapront indicated that he was the one who decided to use some sort of formalism in developing SACEM because of the completely new nature of the approach and technology. Mainly, he was worried about the amount of software and unpredictability and reliability issues, together with cost. To him, safety concerns meant testing the new system exhaustively, which would be impossible. In 1980, or thereabouts, he began reading the literature and discovered the Hoare assertions, which he felt might be appropriate. Others present at the interview supported this.

In response to a question about the utility of a high-level formal specification earlier, Chapront indicated that this would have supplied a little help but, in reality, they really didn't know what to specify and learned only as they got more into the application. SACEM was evolving, starting from scratch with novel technology.

Re-validation: After awhile, they had three sets of proven software using the Hoare assertion approach, without knowing if these were consistent. An external evaluation was performed, involving Jean-Raymond Abrial. He brought their attention to the B method around November 1987. There was apparently some question about what had actually been proved (also reported by Marie-Claude Gaudel).

They didn't know of any proofs of the scale they were attempting or what process to follow so they began a top-bottom re-specification and refinement along with a bottom-top re-specification and verification (along the Hoare line) with the hope of meeting in the middle. The goal was to go down to assertions following the original design structures. They developed a technique for extracting the pre-post assertions from the space and, when they got to a level close to code, these were matched with the bottom-up effort. The general process was:



Specifically, the Hoare assertions were worked upward using just the headers of procedures, to abstract toward higher level operations to meet the results of the B refinement.

The B refinement team had ten people, some with the Hoare background, some from RATP and SNCF. The state machine structure was followed by both approaches. This was all before the B tools were developed, which are now used all the way through (e.g., as in the Calcutta system).

Training: GEC Alsthom's internal training consists of 21 sessions on train signalling, one on formal methods (now B). This training program was started after the SACEM project was over. The RATP had to train ten people to handle their responsibility to monitor projects using formal methods. French railway engineers might have been exposed to formal methods, e.g., at Grenoble.

FM Issues: Several issues were extracted from the SACEM experience:

First tools: After learning the Hoare method, GEC Alsthom developed a tool to manipulate the substitution of variables (the language was Modula 2). This operated on a procedure basis -- bottom-up. The verification person was not participating in the design but had detailed specs and code. This was a very intricate process to follow.

Timing constraints: B was not able to deal with real-time constraints. These could only be dealt with through simulation.

Traceability: Modifications were made by identifying differences between old and new to assure only modifications that were required were made. They also produced modification descriptions, then re-proved procedures and the calls on the procedures.

More tools: GEC Alsthom developed a transformation tool for the Hoare assertions, and an Ada and C translator for B. They already had the B tool from Abrial/BP. Tools wanted were a simplifier and better user interfaces. They could use the tool OK, but the interfaces are not suitable to a wider industrial use).

Scalability question: They believe the B tool is able to do up to around 100 KLOC but the real question is not the ability of the tool to handle this size but the overall resources available. Proof obligations for refinement are the hardest, because B semantics entails a double negation.

Relation to other methods: The Structured Analyses Design Techniques functional decomposition may not map into B and is complicated by the fact that the methods do not use the same terminology. GEC Alsthom finds that the decomposition is organized, although customer goals disappear within the decomposition.

Fault Analysis: The Failure Mode and Effects Analysis is linked to refinements. Operational scenarios may be derived from invariants. RATP has to read the specifications to establish their completeness and the traceability through the system. They find the SACEM formal specifications to be structured and trustable. Failure Mode and Effects Analysis is done by a separate group and is very useful. It is carried out on refinements and often points out errors (before proofs). Analysis proceeds from both a functional and a safety point of view.

Design principles: The SACEM principles were very general so they had design confidence. The requirements often dealt with peculiarities and variations. There were also issues such as what should the driver be informed.

Research needs: B cannot specify concurrency or communicating processes. Simulation for the B Abstract Machine Notation is desirable. GEC Alsthom is working on animation of the formal

specifications.

FORSE toolset: GEC Alsthom demonstrated their FORSE environment, which includes a type-checker, proof obligation generator, simplifier. These tools are written in the programming language of the B-tool.

7.5.5 Results

RATP point-of-view: As the customer, RATP started out somewhat sceptical of the formal process, initially with the Hoare assertions and later with B. They consider this effort on SACEM to have been eye-opening and instructive on the value of combining formal methods with prototyping, simulation, and testing. Since RATP is in the business to validate, the ability to read the formal specifications and prove properties is a plus. They would like to see more work on animating specifications and tying formal methods to quality assurance.

GEC Alsthom data: Data was collected from the beginning of the project on hours spent on particular tasks, resources committed to achieve these tasks, and error rates for the various tasks (for both the simulations and for the formal specification-verification stages). We were given digests of this data with real figures. The overall time figures may be disclosed but we were asked to keep the itemized breakdown of hours spent confidential. Error discovery and fix rates are extracted from the validation report prepared for RATP by GEC Alsthom, Matra, and CSEE (in French, the "Bilan de la Validation").

They see a potential Return On Investment of 30 person-years on such projects. The only way to save money in safety-critical systems is through new technologies. A key characteristic is the cost: Verification and Validation is 1.5 times the cost of design.

Effort Data: The total number of hours on developing the prototype was 170,000 hours. The application of the prototype to commercial service on Line A took 145,000 hours. The breakdown in percent of the effort for the formal and non-formal aspects is as follows:

formal proof	32.4%
module testing	20.1%
functional testing	25.9%
re-specification	21.6%

Validation Effort

	Train	Track
Number of procedures proven formally	111	21
Number of procedures covered in semi-global tests	120	33
Number of procedures tested semi-globally	79	67
Number of procedures tested globally	180	167

7.6 Evaluation

7.6.1 Background

Product:

A certified safety-critical railway signalling system. Specifically, the Automatic Train Protection subsystem deployed on the Paris subway system Line "A".

Industrial Process:

The developer works closely with an authorizing organization to produce the required system and validate its safety. Together, they use a variety of different techniques to assure themselves of the system's safety.

Scale:

Lines of verified, safety-critical code is around 9000 lines with an additional 6000 non-critical lines. The more relevant measure is number of passengers carried per hour (60000) and costs entailed had it not been possible to deploy the system ($millions to create a new line at an earlier time).

Formal Methods Process:

After an initial application of the classical Hoare method, a new and higher level technique, B, was brought in. A fuller top-to-bottom proof was approximated by recapitulating the design as a verified refinement that was reconciled with a recapitulation of the Hoare proof abstraction. Later projects are using the B method and tools from top to bottom, including generation of code.

Major results:

Formal methods have been integrated with other methods (fault analysis, simulation, operational testing) to produce an overall process acceptable to both builder and assessment authority. The methodology is assimilated as part of the GEC Alsthom process and used on successor projects.

Interview profile:

2 days, including tools demos and subway excursion.

7.6.2 Product features

Client satisfaction	+	(1)
Cost	n/a	(2)
Impact of product	+	(3)
Quality	+	(4)
Time to market	n/a	(5)

(1) RATP has approved the SACEM system and supports its use by the Paris Transit system. The system is apparently functioning properly with respect to the ultimate client -- the Parisian commuter.

(2) No figures were available but they believe the cost was not increased, although this is a complex function of sociology and technology.

(3) SACEM helped the RATP to improve its subway capacity. For GEC Alsthom, additional products are being sold using the same approach.

(4) The system meets its quality requirements as established by the RATP, increasing throughout and achieving certification.

(5) Not applicable. The system did take several years, but this is common for large urban projects.

7.6.3 Process features

General process effects

Cost	0	(6)
Impact of process	+	(7)
Pedagogical	+	(8)
Tools	+	(9)

Specific process effects

Design	+	(10)
Reusable components	+	(11)
Maintainability	n/a	(12)
Requirements capture	+	(13)
V & V	+	(14)

(6) Data suggests that the additional activities required for assimilating and using formal methods did not add to the overall process costs, given that the RATP accepted the verification as part of the certification.

(7) A significant process was established over several years to support the use of software in the context of safety-critical railway systems.

(8) GEC Alsthom learned to apply the methods they found available: first, Hoare assertions; then, B.

(9) Working with only crude tools at the beginning, GEC Alsthom managed to complete significant specification and verification. This then lead to the creation of their own FORSE tools, which are planned for eventual commercialization.

(10) The B Abstract Machine and refinement approaches have become part of the GEC Alsthom design methodology on KVS and Calcutta. However, formal methods did not affect the design (only the verification) of the earlier SACEM.

(11) The SACEM safety model and process specification outline is used in their other products. Component use was not discussed.

(12) The discussion did not cover maintenance.

(13) Requirements understanding and negotiation were assisted by the formal specifications.

(14) V & V was a major requirement. Although formal methods were not the only means of assurance, they played a major role in certification.

7.6.4 Observations

Although not required of SACEM, its process and results appear to meet the spirit of the U.K. Ministry of Defence's Draft Standard 00-55. Mathematical specification are used, proof obligations are discharged, and hazard analysis is incorporated.

The response to evolution of formal methods is interesting. A heavy investment in the Hoare method and a verification using it was preserved and extended by a pragmatic re-validation effort. This may have been a turning point in that had a suitable compromise not have worked out either the formal methods might have been abandoned or an unwieldy one (Hoare assertions) continued, with later problems of cost and doubt.

7.6.5 Formal Methods R & D Summary

7.6.5.1 Methods

Specification

The B Abstract Machine approach became the primary specification structure. Its inability to fully express the model by lack of concurrency notwithstanding, Hoare assertions followed by specification in B led to an acceptable high level system model. However, see the efforts of the LaCoS project to add improved modelling capability.

Design and Implementation

These products are being developed using the "classical" (abstraction function) refinement approach.

Validation and Verification

A full scale verification has been carried out to a low level of detail. With the capability to generate code from tools being developed, this will yield a verified layered system. Validation is separately performed via simulation and operational scenarios using the specification.

7.6.5.2 Tools

Language processors

An early tool performed the substitution inherent in the Hoare assertion method. The B tool provides refinement expression and proof obligation generation. Some additional editors and object managers are under development.

Automated Reasoning

The early tools lacked the desirable simplification capability. The B tool offers a more effective

theorem proving environment.

Other tools

The FORSE toolset incorporates what GEC Alsthom considers necessary for a specification environment.

7.6.5.3 Recommendations to the FM Research Community

This is one of the largest scale applications of formal methods known. Its scale is achieved by integrating additional techniques that complement the formal specification and verification. However, the technical interactions of these techniques are still somewhat mysterious. It would be interesting to know:

- Can the specification be used to assist the generation of operational scenarios?
- Can the specification be used to animate the execution of those scenarios?
- How does fault analysis trace through layers of use of the B method?
- Is automated code generation a valid claim? Is it specific to this domain?
- How are real-time properties stated? When simulation is used, what claims can be made that justify that properties are met? Could these properties be stated in a real-time logic?

7.7 Conclusions

SACEM is perhaps the most advanced industrial application of formal methods that we have seen in the case studies. The success of the product depends upon its certification and, of course, its quality. GEC Alsthom and RATP collaborated to produce a mutually acceptable industrial process that made heavy use of formal methods, among other techniques. The process is being applied to other products, some with significant export implications. In addition, the experience is leading to a collaboration to produce a toolset that may incorporate some aspects of the process developed for SACEM.

8 NIST TOKEN-BASED ACCESS CONTROL SYSTEM (TBACS)

8.1 Case Description

NIST Token-Based Access Control System (TBACS) is a smartcard access control system with cryptographic authentication. TBACS was developed by the U.S. National Institute for Standards and Technology (NIST) to replace traditional password-based systems. TBACS was formally specified and verified (with respect to a security model) using the Formal Development Methodology.

8.1.1 Formal Methods Background

The formal methods usage in TBACS follows a venerable process established from 15 years of research and practice. A security policy enunciates a set of permitted and prohibited actions, with emphasis on information access and transmission. This security policy is captured in a formal model expressed in mathematical logic. The system, viewed here as the 17 TBACS commands, is expressed as a state transition system about which the security (model) assertions may be proved as invariants on transition histories. This process is supported in a tool-based methodology called FDM (Formal Development Methodology, from UNISYS) using an interactive theorem prover.

In this case, critical functions (see functions box) were identified and specified following FDM and then verification was attempted.

8.1.2 References

1. "Formal Specification and Verification of Control Software for Cryptographic Equipment", D. Richard Kuhn and James F. Dray, 6th Computer Security Applications Conference, Phoenix AZ, December 6-8, 1990.

2. "SmartCard Technology: New Methods for Computer Access Control" Martha E. Haykin and Robert B. J. Warnar, NIST Special Publication 500-157, Sept. 1988.

8.2 Questionnaire 1

Note: As Rick Kuhn's response to the first questionnaire summarizes the TBACS work, we include it here.)

A. Organizational Context

The smartcard was developed by the Computer Security Technology group, under DARPA contract. The goal of this group is to develop and obtain technology for computer security for civilian Federal and commercial use. Its products include identification and authentication technology, cryptographic techniques (e.g. DES, DSS), and publications advising U.S. Federal agencies on use of this technology. The group has about 10 employees and is part of the Computer Security Division.

The formal methods work was done within the Software Engineering Group. The purpose of this group is to provide standards and advice on software engineering for Federal use. The primary products of the group are related to open system standards, particularly POSIX and related standards. These products include conformance test suites, test specifications, publications on open systems. We represent the Federal government on POSIX and other IEEE and ISO standards groups. This group has about 12 employees and is part of the Systems and Software Technology Division. (Number of employees is hard to define, because there are several part time faculty associates and co-op students.)

Larger organization: NIST is the US's measurement and testing laboratory, chartered to assist US industry and government by providing standards and technology for materials science, computing systems, chemistry, electrical systems, physical metrology and other fields.

B. Project Context and History

The application is a cryptographic authentication system using a smartcard and card reader, used for controlling access to a network of computer workstations. Detailed description is in [1].

Spring/Summer 1988:	TBACS simulation written in C.
November 1988:	Started TBACS prototype firmware implementation.
Fall 1989:	Completed TBACS prototype firmware implementation.
Fall 1989:	Started SACS simulation.
December 1989:	Started formal specification and verification of TBACS.
May 1990:	Completed formal specification and verification of TBACS.
July 1990:	Started formal specification and verification of SACS.
August 1990:	Completed formal specification and verification of SACS.
Fall 1990:	ASACS development started. AFSACS now also in progress.

C. Application Goals

The product was developed to explore ways of providing improved authentication techniques for network access control (i.e., better than passwords).

D. Formal Methods Factors

The previous two years I assisted the US Treasury with their testing and evaluation program for crypto devices used in electronic funds transfer. To provide a more precise definition of security requirements for these devices I proposed developing a formal specification and looking into requirements for stronger evaluation criteria based on formal methods (but not with all the baggage associated with NCSC A1 evaluation level). My sponsor lost funding for this program, but I was still interested in trying formal methods on a real project. I found out about TBACS and talked to Jim Dray, the principle developer. He and his manager, Miles Smid, agreed to cooperate in the effort to do a formal specification and verification.

Method chosen: FDM/InaJo. There were two systems that I could get free from National Computer Security Center: FDM and Gypsy. Gypsy required a Lisp compiler that I didn't have at the time, but FDM was completely self contained and available for VAXen.

E. Formal Methods and Tools Usage

I used FDM on a workstation with the X Window system. By having xterm save 400 lines of backscroll, it was possible to look back at things that scrolled off the screen. I can't imagine using FDM without this capability. I usually kept a printed copy of the criteria (state invariants) and constraints (state transition constraints) available, on which I wrote down the assertion numbers of the criteria and constraints.

F. Results

Advantages: developing the specification brought up several questions that resulted in improved security features. Others were found during the verification. The verification uncovered one significant security flaw before the firmware was completed.

Disadvantages: FDM worked fairly well for this application. The biggest disadvantages are the cluttered screens and large complex expressions it generates. Better fonts and use of colour might help the readability. The complexity of expressions is probably unavoidable, but there were times when a the system probably could have done more. About 80-90% of the proofs were trivial and could probably have been done by the system if it had more capability. The other 10-20% required a lot of thought and human interaction with the system. The easy ones were not a significant problem, though, because they became almost automatic after a little experience.

Done differently: With more resources, I would have spent more time looking at other tools. I was lucky that the only tool available was a good match for the application.

G. Additional Information. None.

8.3 Interview Summary

8.3.1 Organizational Context

NIST
- Systems and Software Technology Division
  - Software Engineering Group (D. Richard Kuhn)
- Computer Security Division
  - Computer Security Technology group (James F. Dray)

NIST: NIST is the United States measurement and testing laboratory, chartered to assist industry and government by providing standards and technology for materials science, computing systems, chemistry, electrical systems, physical metrology and other fields.

Security Division: The TBACS smart token was developed by the Computer Security Technology group, under DARPA contract. The goal of this group is to develop and obtain technology for computer security for civilian Federal and commercial use. Its products include identification and authentication technology, cryptographic techniques (e.g., DES, DSS), and publications advising U.S. Federal agencies on use of this technology. The group has about 10 employees and is part of the Computer Security Division.

Software Technology Division: The formal methods work was done within the Software Engineering Group. This group provides standards and advice on software engineering for Federal use. The primary products of the group are related to open system standards, particularly POSIX and related standards. These products include conformance test suites, test specifications, publications on open systems. This group has about 12 employees (there are several part-time faculty associates and co-op students).

8.3.2 Project Context and History

Prototypes: TBACS was one of a series of prototypes, with the formal methods work for TBACS overlapping the design and simulation on its successor, SACS (Smartcard Access Control System). The TBACS firmware was used in SACS and moved to a Smartcard with a Hitachi H8 processor.

2Q/3Q	1988	TBACS simulation written in C.
4Q	1988	Started TBACS prototype firmware implementation.
4Q	1989	Completed TBACS prototype firmware implementation.
		TBACS sent to Datakey for manufacturing.
4Q	1989	Started SACS simulation.
4Q	1989	Started formal specification and verification of TBACS.
2Q	1990	Completed formal specification and verification of TBACS.
		Serious error discovered April 4, requiring SACS modifications.
3Q	1990	Started formal specification and verification of SACS.
3Q	1990	Completed formal specification and verification of SACS.
3Q	1990	ASACS ("advanced SACS") development started for DARPA.
	1992	AFSACS (Air Force) now also in progress.

Evolution: Simulation was the very first step, initially because no smartcard existed. Once the smart token had been produced (by Data Key, the manufacturer), and the C code along with it, then there was real code to use as the basis for the formal specification and verification. At the time, this code was also undergoing functional testing. The shift from TBACS to SACS required a complete re-implementation

because of the change of hardware. Moving from SACS to ASACS was much easier. There were no changes in authentication procedures, but work in compressing the public key algorithm.

Educational Backgrounds: Jim Dray is a hardware and software engineer with a Masters of Computer Science from Johns Hopkins University. Rick Kuhn is a software engineer with a Masters of Computer Science from the University of Maryland. Kuhn performed the formal methods work on this project and he took a formal methods course while at Maryland. The primary part of his job is to support open systems and standards and he is deeply involved in the POSIX standard. About 25% of Kuhn's time is spent on technology exploration.

Personnel: The project (SACS) team was: Dray, a technician, a mathematician, a Manager (Miles Smid), and a part-time hardware tester. An overall level of effort on the above was in total about two person-years for TBACS and SACS, but project personnel were working on several projects at once. Rick Kuhn participated in the latter stages with the formal methods experiment.

8.3.3 Application Goals

When a microprocessor (e.g., one embedded on a "smartcard") replaces a simple password system for network access control, security may be improved but at the cost of increased complexity. The NIST Token-Based Access Control System (TBACS) was designed and manufactured as a prototype to push the smartcard industry and to demonstrate new security approaches for government agencies.

The credit-card sized TBACS token carries a microprocessor with 17 commands implemented in firmware that is stored in the token's non-volatile memory. A Security Officer uses certain commands to initialize the card with the PIN (Personal Identification Number), cryptographic keys, and network IDs. After initialization, during an authentication, a workstation "login manager" assures mutual authentication of the user to token, token to user, token to workstation, and token to remote host. This process is achieved through cryptographic operations that minimize exchange of physical data and IDs, here, using the DES (Data Encryption Standard) algorithm and data stored on the card. A critical step in the scheme is that the token must deactivate itself after three failed login attempts or reaching an expiration date.

Goals: The product was developed to explore ways of providing improved authentication techniques for network access control (i.e., better than passwords). The objective was to push smartcard technology and to have cryptography occur on the smart token. More generally, there is an interest in exploring the applications of cryptography in computer security.

This project did have time-to-market concerns in that NIST was trying to push the smartcard industry. Timing was a concern in the development since the algorithms used could be quite slow on the small smartcard CPU.

The value of finding the serious error was that there would have been a reduction in confidence by NIST clients if the problem had been disseminated. Hence, the quality of the application was of importance.

Several verification failures exposed design changes that would improve TBACS, including one significant flaw that would have permitted a user to reactivate a deactivated token, violating a requirement that only the Security Officer (SO) could do this:

> "An error in the token PIN change procedure allowed a user to reactivate a token that had been deactivated as a result of expiration. The token pin change procedure checked that either the

> user or SO had been authenticated, without checking to see if the token was deactivated. In the original design this would have been acceptable, because the token was authenticated first, then the user. The token authentication procedure checked for a deactivated token and the sequence would have been stopped by this procedure. With the change to authenticate the user first, the token PIN change procedure could be invoked before the token had been authenticated."

In other words, a design decision changed regarding authentication order but the reactivation command was not changed accordingly.

Evolution: TBACS required the development of hardware by NIST. The primary SACS goal was to move firmware into the smartcard package and to support some checking of the command set. The current work on ASACS is the inclusion of public key cryptography on the card. (The problems here are primarily one of implementing public key efficiently on the restricted CPU and memory of the smartcard.) AFSACS is aimed primarily at performance enhancement.

8.3.4 Formal Methods and Tools Usage

Motivation: The previous two years Kuhn assisted the U.S. Treasury with their testing and evaluation program for cryptographic devices used in electronic funds transfer. To provide a more precise definition of security requirements for these devices, he proposed developing a formal specification and looking into requirements for stronger evaluation criteria based on formal methods (but not with all the baggage associated with NCSC A1 evaluation level.) The sponsor lost funding for this program, but Kuhn was still interested in trying formal methods on a real project, found out about TBACS -- through a paper in the 12th National Computer Security conference -- and talked to Jim Dray, the principle developer. Dray and his manager, Miles Smid, agreed to cooperate in the effort to explore formal specification and verification.

Criteria: FDM/InaJo was chosen on the basis of availability. There were two systems free from NCSC: FDM and Gypsy. Gypsy required a Lisp compiler that they didn't have at the time, but FDM was completely self-contained and available for VAXen. Documentation for FDM is adequate and about the right size and coverage.

Specification: Using FDM's "criterion" and "constraints," Kuhn specified the security requirements of TBACS (and SACS). The formal methods exercise was a reverse engineering task in that existing source code was analyzed and then specified using FDM. Only the security critical functions were treated in this manner. In attempting the proofs, a number of proofs failed, and enhancements to the code were needed. Dray was able to read the InaJo notation but required help from English marginal annotations.

• • • • • • • • • • • • • • • • • •

Verified:

> Enter SO PIN, Authenticate SO, Enter User PIN, Load Key,
> Authenticate Token, Generate Challenge, Authenticate User,
> Change Token PIN, Workstation Verify and Respond,
> Host Verify and Respond

Not Verified:

> Reset, Output ID Table, Read Zone, Write Zone,
> Append Zone, Call DES, Test

Figure 5 Verified and not verified smart token commands

In moving from the formal specification to code, Kuhn indicated that some reverse engineering was involved because the C code was already available. But there was not a direct mapping. The process used was to read the C code, break it into chunks, and prioritize these based on talks/reviews with Dray as to importance. Also, much of the work was simply translating the C code into InaJo and deriving pre- and post-conditions. For this project, the state transition conceptual framework worked very well. He did, however, find that extracting the preconditions for the various routines (e.g., environmental assumptions) rather difficult. He was not particularly concerned with the soundness of FDM's logic since the prover does not do much for you; the inference steps are small. Kuhn feels that FDM would be good for systems that do not have concurrency and that are event driven, for example, X-windows.

Tool Experience: Kuhn used FDM on a workstation with the X Window system. Xterm saved 400 lines of back scroll, making it possible to look back at things that scrolled off the screen. (He can't imagine using FDM without this capability.) Kuhn kept a printed copy of the criteria (state invariants) and constraints (state transition constraints) available, on which he wrote down the assertion numbers of the criteria and constraints. Better fonts and use of colour might help the readability. The complexity of expressions is probably unavoidable, but there were times when the system probably could have done more. About 80-90% of the proofs were trivial and could probably have been done by the system if it had more capability. The other 10-20% required a lot of thought and human interaction with the system. The easy ones were not a significant problem, though, because they became almost automatic after a little experience. FDM/InaJo were not able to handle the timing considerations, particularly synchronization. Kuhn noted that conjuncts could be written for the clock timing, but not clock synchronization.

Kuhn found FDM easy to use; the documentation [specifically the FDM tutorial] was supportive for learning how to use the system (examples that were neither too easy nor too hard). Kuhn said his training in FDM/InaJo consisted of doing the InaJo tutorial provided by Unisys. In general, the tutorial is pretty good, although it definitely helped his understanding to have worked with logic and theorem proving in graduate school (at the University of Maryland in classes from Mark Weiser). However, there was too much clutter on the screen. The prover did perform reasonable chunking of the propositions (proof by refutation and use of disjuncts).

Tool Value: Regarding tool cost effectiveness, Kuhn estimated that cost-effectiveness was on the order of 3x over having no tool support at all (except for a test editor) assuming all the proofs were done. The gains were mainly in expression simplification and structure mechanization.[4] Kuhn's confidence was not increased just because the proof was mechanized. A previous effort to do a hand proof of a robotics algorithm showed him there were differences in types of proofs.

Testing: NIST used simulation techniques to check functionality. Dray claimed that if executable formal specifications existed they would have used them instead of proceeding with their usual simulation approach. However, even without execution, Dray indicated they did learn quite a bit about the system from the InaJo spec, plus detecting a serious bug that would not have been found in testing. Kuhn noted

[4] During the interview, we had some discussion about exactly what "mechanized" meant, basically the distinction between the predicate pushing involved in verification condition generation and the automated reasoning involved in their proofs. We are still not sure what this estimate means.

that if they had used a formal specification (without code) from the beginning, the Data Key developer would have been required to understand the notation. Live testing used a PC hooked up to a Sun. The test plan is available, but is now less formal because good communication exists between the tester at NIST and the person at Data Key. The formal spec was not put together with the test plan: the formal specification and verification work was done in parallel with testing and the serious bug was found before the part where the bug resided was tested.

8.3.5 Results

Effort expended on the formal specification and verification amounted to 6% of the overall design, simulation, and software development. Specifically, formal methods were applied to a portion (10 functions) of the software simulation (2500 lines of C) in the form of ~300 lines of FDM utilizing 92 hours over 5 calendar months.

Phase	Hours	Calendar Months
Design	320	2
Simulation	480	3
Hardware Development	751	2
SW Development	650	10
Acceptance test SW	140	1
Project Management	800	15
Verification	92	5

Verification / (Design + Simulation + SW Development) = 6%

Defect Identification: Overall, developing the specification brought up several questions that resulted in improved security features. Others were found during the verification. The verification uncovered one significant security flaw before the firmware was completed.

FDM worked fairly well for this application. Its biggest disadvantages are the cluttered screens and large complex expressions it generates.

FM (Dis)continuation: Dray considered the main results to be: (1) that SACS is usable in a production environment; and (2) the project demonstrated that cryptographic means can be used successfully in general in government for authentication.

Dray did not consider formal methods to be a high priority even though they discovered a serious fault ("bug"). If extra resources were available, he would spend them on other capabilities (e.g., object-oriented, algorithms). Dray acknowledged that formal methods are "nice to have" and that it was very important that the error was discovered before the system was delivered but, given the budgetary and other resource constraints, he would choose to add someone with other skills than formal methods first, and only if he had more than one additional person that could be added. His top priority was to implement TBACS, then SACS, with the then-current resources. In other words, if the project had included five or six people, Dray may have included a formal methods expert in the project. Formal methods have not been adopted on successor TBACS work. Dray indicated there was no external push by other U.S. government agencies or contractors to verify the firmware.

Kuhn would use more formal methods and is planning to on a new contract with the Air Force on a secure telephone switching system. Kuhn believes from his experience that formal methods are useful in

finding important properties. With more resources, Kuhn would have spent more time looking at other tools. He felt lucky that the only tool available was a good match for the application.

8.4 Evaluation

8.4.1 Background

Product:

a smart token prototype (actually a series of) for use in secure government networks.

Industrial Process:

NIST does (1) standards demonstrations and (2) technology explorations. The smart token was a demonstration of cryptography on a smartcard. The use of formal methods was for exploration of both methods and tools.

Scale: 1 year (total) effort on each of a series of smartcard designs. Partially completed formal methods use would have scaled to 6% of full project effort on TBACS.

Formal Methods Process:

A venerable tool and method was used -- state transition models and a proof checker. A security model was developed and expressed in this approach and major steps were fully specified and verified while others were omitted due to scale of the effort.

Major results:

The smart token is in the field and further prototyping is continuing in the NIST lab based on successful demonstration of cost-effective cryptography on a smartcard. The formal methods usage showed (1) how one approach could address major aspects of the secure token formalization and (2) discovery of a significant flaw that could have undermined the success of the product.

Interview profile:

1/2 day at NIST, interviewing the smart token designer and the formal methods experimenter. Additional background from case study paper. Additional materials include the specification, C code, and Kuhn's work log.

8.4.2 Product features

Client satisfaction	n/a	(1)
Cost	0	(2)
Impact of product	+	(3)
Quality	+	(4)
Time to market	n/a	(5)

(1) The clients for TBACS were DARPA, NIST and the U.S. Army. We did not interview any of the clients directly.

(2) There was no additional or reduction of cost to the TBACS project. The smart tokens were prototypes to drive the smartcard industry and identify government applications. Product costs were considered reasonable.

(3) There was a marginal positive impact on the organization in that further funds were allocated to proceed with the smartcard experiments. This is partly due to the quality of the product that was produced by the TBACS development. From another perspective, if the security flaw had not

been found (by testing using simulation or by proof), there could have been reduced confidence on the part of the clients. Additionally, if the TBACS system had been manufactured without the error being discovered, a recall would have been expensive.

(4) The quality of the product was clearly enhanced by the elimination of a firmware error before manufacture.

(5) We could not determine any positive or negative effect of the use of formal methods on the release of TBACS. Note that time to market was defined as getting a prototype in the field in order to drive smartcard vendors.

8.4.3 Process features

General process effects

Cost	0	(6)
Impact of process	0	(7)
Pedagogical	+	(8)
Tools	+	(9)

Specific process effects

Design	+	(10)
Reusable components	+	(11)
Maintainability	n/a	(12)
Requirements capture	+	(13)
V&V	+	(14)

(6) There was no significant additional or reduction of cost to the TBACS project. Factored into the process, the formal methods use would have been 6% of total effort [1]. Discovery of the security flaw was important, but no cost was attributed to it within the prototyping process being used.

(7) There was no immediate impact on the procedures used by NIST or by the ongoing smartcard developments. Though formal methods had found problems with the TBACS implementation and helped to develop the security model, the gains were insufficient to result in changes to the development processes of this or other NIST units. However, TBACS/SACS and the previous Treasury Department effort demonstrated the feasibility of formal methods (at reasonable cost) for cryptographic devices. As a result, Kuhn told us, formal methods will be required for higher assurance levels in FIPS 140-1 "Security Requirements for a Cryptographic Module".

(8) One of the purposes of the project was for a NIST researcher to learn about formal methods. It is clear that he made the most of his opportunity and positively influenced the TBACS development. In particular, the experience was well documented in an open case study paper and the specifications are available for formal methods research use.

(9) Kuhn claimed that the tools helped to manage the proof and greatly expanded his capabilities. He was able to learn how to do this application and how to use the tools on his own from the tool documentation.

(10) The formal methods use was post-design, although productive questions were asked of the designer by the formal methods specialist in the latter stages of the design.

(11) The use of formal methods led to the proposal of a security model that could be used (in a modified manner) with other systems of similar intent. This is both a research and a practical contribution.

(12) TBACS was a prototype in a series of research development and the resulting system was not developed with long-term maintainability in mind.

(13) Requirements expression was clarified because formal methods helped in the development of the security model for TBACS. However, pre-design requirements capture was not accessible.

(14) A significant addition to the smart token V & V was in suggesting cases where the implementation could fail, resulting in the identification of one significant failure to be corrected.

8.4.4 Observations

NIST is neither a certifying agency nor a commercial firm. Rather its industrial perspective is in demonstrations that advance the state of the practice and lead the industry toward appropriate standards. Thus, this case mixes concerns that are both research and product oriented. The process used at NIST may be typical of that to be found in a smaller commercial firm.

There were no formal methods specialists within these branches of NIST (although others existed in ISO protocol areas). Part of the motivation for this experiment was to help discern the potential role of formal methods within NIST standards activities. The TBACS experiment was a great success from a pedagogical point of view, if not convincing enough to modify the processes of a unique organization.

Smartcards are a good match with the capabilities of formal methods. They require high assurance because their deployment invites abuse and misuse while simultaneously offering unique capabilities. As technology, the limits of the smartcard limit the scale of the application and therefore fall within the range of formal methods capabilities. This experiment may help broaden the perspective of the formal methods research community.

This application illustrates a well-established formal methods practice (at least in terms of number of U.S. applications performed), in the realm of computer security although there is little widespread knowledge of these results. Thus, the smartcard application illustrates some experience with security verification.

The formal methods tools experience was one of the most positive reports so far in that the tools were not obstacles and did help the process. However, this is said of one of the oldest and simplest toolsets available and there is a good match with the smart token application.

The Security Applications paper [1] illustrates a good reporting style for formal methods cases, including brief descriptions and some data. However, the paper is too brief to replicate the experiment and somewhat cryptic in its discussion of the error. In particular, the interaction of the different TBACS-SACS-etc. versions did not become clear until the interview. The choice of tool was strongly influenced by being on the U.S. Computer Security Certification approved tools list and, therefore, its "off-the-shelf" availability.

8.4.5 Formal Methods R & D Summary

8.4.5.1 Methods

Specification

Security assertions and some history (transition sequences) properties were proved. This particular system fits the state transition paradigm quite well. However, the overall system model may be questioned (does the model capture all the essential properties of the smart token network access?).

Design and Implementation

Programs were addressed as C code used as the basis for simulation. These were abstracted into data types and state transitions to express the formal top level specification. No obstacles were mentioned, although the security model was essential before the code could be interpreted. Better understanding of this reverse engineering process into a state transition specification is worthy of investigation.

Validation and Verification

The validation process for the security model was informal communication between formal methods specialist and smartcard designer. Verification was undertaken with the objective to get through the critical functions and see what would be learned. Close communication got them through the process (note that this would not have worked with the more remote Data Key developers). Verification was a standard use of the theorem prover that did not seem to present any obstacles.

8.4.5.2 Tools

Language processors

No specific tools other than the InaJo processor were available or created.

Automated Reasoning

The interactive theorem prover was used in the classical way, with most proofs being trivial and a small portion requiring longer chains of reasoning or interaction with the designer. Scrolling and paper-and-pencil cross-reference were the only additional logic support tools. For this project, the prover did not present insurmountable obstacles or unbearable costs, although its usage was difficult.

Other tools

No specific process support tools were used. Personal communication was the key.

8.4.5.3 Recommendations to the FM Research Community

- The reverse engineering process for taking the Smart Token simulation code to InaJo specification was essential but has not been studied. How does one systematically extract the various state machine components, e.g., pre-conditions and constraints?

- Could the error found in this verification exercise have been found by some systematic testing regime? by inspection? by symbolic execution?

8.5 Conclusions

This is a modest scale experiment that taught its participants a great deal. First, a significant and realistic flaw in a version of the smart token was found and removed in an opportune way but without excessive cost. An available method and tool were readily adapted and put to use. The commonly used state machine model worked well and the tools did not present barriers.

However, the experiment was not encouraging in that formal methods remained low on the list of priorities for capabilities to have on future projects. The TBACS experiment may have influenced the longer term development of related standards.

9 TEKTRONIX: USE OF Z METHOD ON OSCILLOSCOPES

9.1 Case Description

In early 1987, Tektronix, a manufacturer of electronic instruments with headquarters in Beaverton, Oregon, began an R & D project to design a reusable software framework for oscilloscopes. Results from this R & D project were incorporated in 1988 into a specific product development effort of a successor family of oscilloscopes to a then current product offering. The product development effort involved three or four parallel but related development efforts involving three divisions and the research labs (a total of some 10-12 people) and frequent meetings with all the product engineers (40-50 people total) were held. It was a joint development effort between Tektronix' research labs and one of their business groups (the Test and Measurement Product Division). The results of this project became the basis for a product development effort with high corporate priority.

Issues of interest to this study in the Tektronix case concern the use of a formal method as a communication medium among the various members of the development team with different roles, backgrounds and expertise.

9.1.1 References

1. Delisle, Norman and David Garlan, "A Formal Specification of an Oscilloscope", IEEE Software, September, 1990, pp. 29-36.

2. Garlan, D. and Delisle, N., "Formal Specifications as Reusable Frameworks", VDM90: VDM and Z!, Springer-Verlag, New York, 1990, pp. 150-163.

9.2 Questionnaire 1 (Delisle)

A. Organizational Context

The project was a joint effort between Tek's research lab and one of Tek's business groups. The business group has a goal to develop manufacture and sell advanced products for measuring electrical signals; the primary product of this group is the oscilloscope. The research lab (now called Software Technology Research Lab, formerly called Computer Research Lab) has a goal of developing (1) software technology targeted at significantly increasing the productivity of Tek's software engineers; and (2) software technology that enables new product opportunities for Tek's businesses. This project was aimed at increasing productivity and shortening time to market.

B. Project Context and History

The project was the design of reusable software for oscilloscopes. The subsequent implementation and use/reuse were separate efforts which I will not describe. The reusable software design effort included about five people---far less than full time---over a duration of about two years.

We started around June of 1988. Formal methods, prototyping and extensive group discussions/reviews were the primary design activities.

Major milestones included publication of internal design documents, formal design reviews, tutorials and executable prototypes.

C. Application Goals

The reusable oscilloscope software was developed to shorten the time-to- market for a family of oscilloscope products. Concerns centered around execution time and space performance, issues specific to reusable software (learnability/useability, extensibility), and issues specific to oscilloscope software (functionality, dynamics, suitability).

D. Formal Methods Factors

The formal method we used was Z. Primary tools were Fuzz (a type-checker), and Latex macros.

We used Z as a design aid---a language for constructing non-executable `prototypes' of designs. Our intent was not formal correctness or non-ambiguous or `provably' complete documentation. We used Z as a concise, precise notation for exploring our design ideas.

E. Formal Methods and Tool Usage

No response.

F. Results

Our use of Z did indeed help us focus our design ideas. Perhaps the major disadvantage is that there is a trap we occasionally fell into: elegant formal expression of an idea does not imply that the idea is elegant. Under similar circumstances I would certainly explore the use of formal methods again. But I am not yet ready to advocate the use formal methods for all production software design efforts. More research is definitely needed.

No metrics were kept.

G. Additional Information

Read the paper "A Formal Specification of an Oscilloscope" by Delisle and Garlan in IEEE Software, Sept 1990.

9.3 Interview Summary

Date: February 26, 1992
Respondent: Prof. David Garlan
Organization: Carnegie-Mellon University, Pittsburgh
Description: Interviewed by Dan Craigen, Susan Gerhart and Ted Ralston (author)

Date: March 25, 1992
Respondents: N. Delisle and M. Schwartz;
Organization: Software Technology Laboratory, Tektronix
Description: Interviewed by Dan Craigen and Ted Ralston (author)

Note: We have merged information from the two interviews into a single summary.

9.3.1 Organizational Context

Tektronix is a company which manufactures a number of product families of advanced instruments, with annual revenues in excess of one billion dollars. In addition to six or seven product divisions, Tektronix has a research laboratory, which investigates new technologies for both hardware and software. Most of the staff in the product divisions are engineers, while the laboratory personnel represent a range of educational and professional disciplines from bachelor degrees to Ph.Ds in electrical engineering, physics, and computer science. Although Tektronix' business is hardware, an increasingly large amount of software is written in order to achieve diverse product functionality (for example, most electronic instruments have more than 100,000 lines of code).

9.3.2 Project Context and History

The following timeline provides a rough chronology of key milestones in the project.

pre-1987: CSP modelling of Tektronix 11000 oscilloscope

early 1987: Labs began R & D project and began working with product groups; investigated various formal methods

3Q/1987: Z chosen as specification language; developed and ran courses plus exercises

1988: Development effort joined by three other oscilloscope divisions; initial formal model developed (user-level model); broke into parallel groups on formal model, architectures, user interface, and signal processing

1989: Worked with product divisions to refine model; new formalization of signal processing and user interface models

1990: Product released; continued to work with product groups and add new clients.

When this project began, Tektronix sales were flat and markets were not growing. Consequently, the need for a flexible new oscilloscope product evolved to redress this worsening business situation, and a reusable software framework became one way to achieve this flexibility.

9.3.3 Application Goals

The main goals of this project were to increase productivity and shorten time-to-market. The business group has a general goal to develop, manufacture and sell advanced products for measuring electrical signals; the primary product of this group is the oscilloscope. The research lab (now called Software Technology Research Lab, formerly called Computer Research Lab) has a goal of developing software technology targeted at significantly increasing the productivity of Tektronix' software engineers; and software technology that enables new product opportunities for Tektronix's businesses.

In terms of the products sold, Tektronix is a hardware company but, like many "engineering" companies, a large percentage of the engineers develop software to achieve new or improved functionality. As an example, early oscilloscopes had a few hundred lines of software, whereas the then existing oscilloscope product from Tektronix had almost 200,000 lines, which were showing several difficult on-going bugs. At the time the project began, Tektronix had developed a lot of software both in the Labs and in the various product groups. The company had no clear idea of what software was there, and had had very limited success in achieving reuse by simply piecing together software that already existed. Management wanted a common software platform built on a reusable framework across the divisions. As a case in point, at the beginning of the oscilloscope project, three separate divisions wanted to share software and they had a problem of determining the right abstractions for this sharing.

Coupled with this problem, the management approach at Tektronix Labs was quite loose, with no general (let alone mandated) development process or lifecycle model. In some divisions software development was very ad hoc and hacker-oriented, which compounded the difficulty in achieving the common reusable-framework software platform goal.

A reusable oscilloscope software was intended to shorten the time to market for a family of oscilloscope products. A one-month delay in delivering the product would have cost Tektronix the equivalent of the entire product development cost. Concerns centered around execution time and space performance, issues specific to reusable software (learnability/usability, extensibility), and issues specific to oscilloscope software (functionality, dynamics, suitability).

There were four drivers on the application: (1) sharing, (2) time to market [the most important driver], (3) management of the software, and (4) incremental changes to the system should lead to incremental changes in software.

9.3.4 Formal Methods Factors

To meet their goals, Tektronix had to determine what sort of model would support them and, moreover, they did not want to have to swap 1MB of software on a regular basis. Hence, as a result of earlier experience with Smalltalk, there was interest in using object-oriented programming.

The research labs were (and still are) a "skunk works" for Tektronix, in the sense that it was part of their charter to investigate new methods and approaches. The history of the Labs has been one of aggressive investigation of new approaches and methods (e.g., Smalltalk, workstations) before their time. The decision to consider formal methods came from the grass roots in the Labs. Two of the team members from the Labs had investigated and noted that formal methods were having some benefits in Europe and they wanted to determine whether there were any advantages to Tektronix. Awareness of an outside example (the use of Z by IBM Hursley on CICS, which at the time was one of the few large industrial applications reporting results), also contributed to considerations on using formal methods. One of the principal members of the team had spent time visiting several European formal methods research centres investigating both the methods and any related tools.

The main problems to be solved in achieving the common reusable framework were principally ones of abstraction, specifying behaviour, and a means for rapid prototyping in order to explore design ideas quickly. This seemed to argue for a design method that stressed modelling, and seemed suited to the European-style formal specification methods, of which Tektronix was aware prior to the start of the project. The method chosen would also have to map well with prototyping.

Several toy examples (e.g., a candy dispensing machine) were developed in Z, LOTOS, CSP, and VDM. CSP was also used to model an existing oscilloscope product. Z was chosen in large part because it was easier for the engineers involved to understand the Z notation (schemas augmented with English annotations) than the other methods. A six-week Z course was developed and used to teach Z to the development team. The course was one and a half hours per week and was driven by the use of case studies so that the engineers started using Z almost immediately. A few key people (mainly engineers but also a couple of key managers) were convinced by this course that Z was worth the risk.

9.3.5 Formal Methods and Tools Usage

Z was used as a design aid---a language for constructing non-executable `prototypes' of designs, and as a concise, precise notation for expressing and reasoning about design ideas by modelling the behaviour. The intent was not to prove the system correct formally nor to produce "provably" complete documentation. The Z was used primarily to clarify concepts and to communicate ideas to the engineers, who at the beginning of the project did not have a good understanding of how their systems were working.

For a North American commercial company, this use of a formal method as a design aid for modelling the behaviour of the system was at the time novel and unique. As noted above, there was no defined or required process for system development followed in Tektronix at the time, and fitting a formal method into this sort of development was seen as "risky" by both management and the engineers.

The following "genealogy" of the specification illustrates the role Z played.

1. Definition of a common terminology (signatures plus some abstraction of an oscilloscope, i.e., what is a waveform).

2. An object-oriented description of the new oscilloscope existed but it was too superficial; a more detailed model of oscilloscopes was developed, combining elements of the object-oriented model and the notion of data flow computation, Object-Oriented Data Flow.

3. This Object-Oriented Data Flow model was further abstracted into a Higher Order Data Flow model (HODF) which was specified in Z, but the HODF did not model the timing and efficiency considerations well, nor the user interface at all. This model depicted additional properties such as the type of connectors between boxes.

4. A prototype implementation was used to investigate the timing and efficiency considerations.

5. Z was used to specify the user interface software architecture. Z used for user interface architecture -- not for the user interface itself.

6. Both the HODF and the user interface architecture specifications were each about 15 pages of Z schemas plus annotations in English.

7. The Z HODF was further modified into a new design notation based on code primitives that could be used in future developments to construct new systems. The word used by Garlan to describe this change was "transmogrification" to indicate the notation changed in order to move to implementation but the underlying theory remained).

With respect to the interplay between the Z specification and prototyping, having the abstract model helped define and express the numerous ties to the hardware. In this case, the hardware was a specialized microprocessor which involved some new complexities such as digital signal processing and concurrency. Z was not suited to handling these complexities, however, and because performance issues (time and space) were paramount a prototype was necessary in order to get feedback.

The Z specification has not, in itself, lasted. This was explained as a consequence of Z only being used for modelling rather than to produce a final system specification from which code could be written. The effect of the transmogrification has been to make the implemented code rather than the Z specifications become the primary "document" used in subsequent efforts to improve by adding performance-oriented changes or to develop further generations of the product. This reliance on the code as "boss" reflects an engineering preference for "the real thing" not an abstraction. There are also new people working on future releases who do not know the Z specification exists or how to use it.

At the start of the development, no tools existed. Shortly after the project started, a type checker for Z called Fuzz, which had been developed by Spivey at Oxford was brought in. Z schemas were printed using LaTex macros. Work was begun on a Z environment (ZEE) but was not finished. These tools were not adequate for many of the mundane tasks required in using formal methods. Garlan felt existence of theorem proving tools for Z would have been helpful. Schema expansion and cross-referencing utilities would also have helped.

Simplification tools and pre-condition calculator would have definitely helped and were needed. Delisle was not sure higher order operators would have been that much help, although he noted they did do some work at what he called the "meta level" (architecture of the architecture) for which some higher order operators could have helped.

Finally, Z notation is not suited to object-oriented design, and temporal properties are not naturally expressed.

9.3.6 Results

As characterized by one of the participants, the success of the project helped to save the research labs as it showed that the labs could have a financial benefit to the organization through development of a real product. This claim was modified by another participant to indicate that it [the "saved the labs" part] was somewhat overstated, and explained that Tek management had decided to use a "hard reset" on the labs as a result of business performance troubles, but this was more of a "refocusing" of work rather than "do away with the labs".

The course helped key people to decide to commit to formal methods as part of the project. "Key people" here meant the engineers and other technical people; a few managers sat in and took away some sense of confidence from the engineers but did not really internalize Z.

The project became important to Tektronix business goals and was based on "risky" technology. They were using formal methods and risky programming languages, pushing the state of the art on signal

processing hardware, and relying on outside vendors for some innovations. Tektronix was banking on the product; it was no longer a prototype exercise.

An advantage of the formal method was that it allowed the developers to think clearly about the software architectures, define a mathematical model with desirable properties, and present that model to the product engineers. The combination of formal modelling and prototyping helped identify design errors and fix them earlier.

In terms of what they would do differently, it was indicated that more interaction with the product engineers would be done to validate the formal model.

The Z model has not been used further, nor has Z been used again to date in Tektronix. There is still an interest in using formal methods again, but the right application has not come along.

9.4 Evaluation

9.4.1 Background

Product:
An oscilloscope product family and develop a reusable common software framework

Industrial Process:
Joint collaboration between Tektronix research labs and three oscilloscope product divisions.

Scale: software involved approximately 200,000 lines of code; at least two Z specifications produced (2 at 15 pages each).

Formal Methods:
Z used to model behaviour and transmogrification of formal specification

Major Results:
Oscilloscope product line successfully launched on time and has proved successful in the marketplace.

Interview Profile:
2 full day interviews were conducted with three members of the development team: Norman Delisle and Mayer Schwartz (Tektronix), and David Garlan (Carnegie-Mellon University).

9.4.2 Product Features

Client satisfaction	+	(1)
Cost	n/a	(2)
Impact of product	+	(3)
Quality	+	(4)
Time to Market	+	(5)

(1) The client in this case was Tektronix itself, and given the testimony that the product has been a successful moneymaker for Tektronix during the recession, we conclude they are satisfied.

(2) In this case, the product developed was "new" in that it added significant new functionality and complexity. Therefore, there was no base against which to compare reduction of cost or increase in profit.

(3) The product has been successful, and future generations of the product are in development, based on the reusable framework derived from the original development.

(4) Developers point to an absence of reported errors to date from the product. How much of this absence is because of the formal method, or the use of object-oriented, or lots of hard work, or some combination of all three, was not possible to determine.

(5) The product was delivered on time, with no delays attributed to the introduction of Z as a new design method (even with the six-week Z course).

9.4.3 Process Features

General Process Features

Cost	0	(6)
Impact of Process	0	(7)
Pedagogical	+	(8)
Tools	0	(9)

Specific Process Features

Design	+	(10)
Reusable components	+	(11)
Maintainability	n/a	(12)
Requirements capture	+	(13)
V & V	n/a	(14)

(6) The impact of the extra time spent in the short six-session Z course was balanced by some speed-up in coding once the modelling was completed, with a net neutral impact on amount of effort.

(7) The impact of Z on the "process" was seen as neutral because there was no defined process to begin with ("ad hoc, hacker-oriented development"), nor was there any evidence that the introduction and use of Z changed this to any significant degree (except in training - see below).

(8) The training in Z had a positive impact on the general processes within Tektronix by showing the value of the training course approach, especially when directly coupled to development of an important product.

(9) The absence of tools at the beginning, the barely adequate Fuzz type checker, and the hindsight testimony on the usefulness of more advanced tools had they been available, lead to the conclusion of an overall negative impact on the process.

(10) The effectiveness of Z as a communication means between designers and engineers in helping them to understand their design, plus the positive role of higher-level abstraction through the use of Z on achieving the reusable common software platform (even though the Z became transformed), had a positive impact on the design of the product.

(11) In this case, Z contributed positively to achieving the management goal of a common reusable software platform by providing a precise notation for modelling important components and expressing them better than the object-oriented method used previously.

(12) No data on maintenance history was collected.

(13) The Z language was used effectively to capture requirements that had either originally been modelled using an object-oriented approach or, more importantly, had not been able to be modelled using the object-oriented approach.

(14) We were told that V & V was not performed on this product.

9.4.4 Observations

This case illustrates the successful use of a formal method as a communication medium between members of a development team with different backgrounds, roles (researchers, engineers, implementers, etc) and levels of expertise. On the whole, Z proved to be effective in capturing and expressing complex concepts related to the operation of the oscilloscope.

The effectiveness of Z as a communication medium did not carry over to implementation, for which an intermediary notation was necessary which eventually gave way to the code as the primary "document" describing the system behaviour. Consequently, while the use of the formal method had a positive impact on the thought processes of those involved by giving them a means for higher level abstraction, it did not significantly change the system development process within Tektronix.

The use of the formal method did facilitate faster design decisions through better clarification of design decisions and decomposition of problems, which in turn were mapped into the prototyping effort by, among other things, defining the ties to the hardware. Z allowed more precision in reasoning about the various arithmetic operations performed on signals, waveforms, and traces.

We do not understand exactly how the formal method contributed to achieving the common reusable software platform other than providing a more precise notation for modelling the various components of the oscilloscope family. By "transmogrifying" the Z Higher Order Data Flow model into the new design notation that could be implemented, it is suggested that the underlying theory of the oscilloscope behaviour was preserved but new constructs for future oscilloscopes could be built that were closer to code. It is not clear that another similarly precise notation might not have a similar effect.

In this case, the availability of better, more capable tools would have improved the productivity. Given that conceptualization was the major intellectual effort involved; however, it is not clear that even simplifiers or pre-condition calculators would have speeded this effort dramatically.

This case also illustrated the role of several "intangible" factors. First, the quality of the personnel who made up the development team (the designers, specifiers, and engineers) was high, and their abilities definitely contributed greatly to the success. Secondly, partly because they were quality people, the openness to trying and using a new method also contributed to the success the method had in helping achieve the goals.

The fact the Z specification "transmogrified" to eventually be replaced by the code as the final specification could be seen as a failure of the formal method to enter into regular practice. When this interpretation was raised with the developers, the view was that the advantages of using the formal method for a limited purpose (e.g., modelling the behaviour) has stuck and they would be prepared to use it again should a suitable application arise. To date, one has not.

9.4.5 Formal Methods R & D Summary

9.4.5.1 Methods

Specification

This application shows that the properties not tractable to specification using Z were the oscilloscope operations related to concurrency and timing efficiency. Those arithmetic operations underlying most of the oscilloscope operations on signals, waveforms and traces could be adequately handled by Z.

The Z schemas by themselves were understandable to most of the team members, including the engineers. However, the English annotations were required for understanding very complex schema operators such as theta or xi. It was noted by the non-academic team members that elegance of Z specification can sometimes get in the way. It is one thing to strive for a deeply elegant specification oriented to formal reasoning, which only the specifier can read. This seems a goal of the academic specialist. The industrial/regulatory environment, on the other hand, is likely to be more concerned with readability and reviewability of specifications by people who do not necessarily have the goal of formal reasoning, nor the ability to do it, in mind.

Design and Implementation

The Z specification of the oscilloscope and the user interface architecture were at a high level of abstraction. No refinement to code was attempted because it was not seen as possible. It is difficult to say definitively whether it was desired, since the main purpose of using the Z was to build the model and reason about it at a high level, rather than take it to implementation. However, given the "transmogrification" history, it is possible to conclude that had there been a clear means for refinement to code it would have been used. This would argue that refinement for the industrial user is a research goal.

9.4.5.2 Tools

The only tools used on this case were an early version of a type checker (Fuzz) and some LaTex macros for printing schemas. While these were adequate, it was indicated that simplification tools and a pre-condition calculator would have been used if they had been available. With respect to pre-condition calculation, one of the developers indicated that, since this calculation can sometimes be tricky, theorem proving tools that could be used as needed would also be desirable.

Other tools that would have been useful were a schema expander and a cross-referencing tool (this latter tool especially relevant to large Z specifications). In addition, at the time of the project a full semantics for Z did not exist, but if it had, a semantic analysis tool would also have been useful.

9.5 Conclusions

We found that a formal method had been used on a significant commercial product which had both important scaling implications for the use of the formal method, and was important to the company as a business product.

The use of Z to model the behaviour of the new oscilloscope illustrates the value of doing a high level abstraction for both understanding the behaviour and communicating it across a staff of diverse roles and backgrounds.

It appears the formal specification effort has not spread to other development efforts or been used again. Tektronix personnel observed that this is because an application with similar aspects to the reusable framework project has not come along, and if one does they would use the formal method again.

At the same time, however, one participant in the development who is no longer with Tektronix observed that the formal specification effort has acted as a "carrier" of the ideas, namely, that while the code may be used as the main documentation for the system, the concepts captured by and expressed through the formal method still reside in that code. Without a detailed examination of the specification and code, which we were not able to do, it is impossible to decide this point definitively. It seems to us to be plausible that the "transmogrification" referred to above did preserve these concepts. This case also illustrates the importance of timing and advance preparation in terms of a successful technology transition. The R&D Labs at Tektronix had already been investigating methods that could be used to develop a reusable framework when the business need arose, and were therefore in a reasonably good position to address the problems in a timely and cost-effective manner (as opposed to starting from scratch). The history of the Tektronix R&D Labs as a "skunk works" also facilitated this situation, as there was a predisposition in the Labs to try new approaches and experiments.

Lastly, this case also reinforced our views with respect to tool development. Tools were not a vital necessity to achieving the results, and while some basic tools were used (and a few others would have been desirable), the need for tools evolved opportunistically.

10 TRAFFIC ALERT AND COLLISION AVOIDANCE SYSTEM (TCAS)

10.1 Case Description

As a result of a U.S. Congressional mandate (known as Public Law 100-223), the U.S. Federal Aviation Authority (FAA) has required the installation on all airline aircraft, with more than thirty (30) seats, of the Traffic Alert and Collision Avoidance System (TCAS II) by 31 December 1993. As of April 1992, half the U.S. airline fleet had TCAS II installed. The purpose of the TCAS family of instruments (of which TCAS II is a member) is to reduce the risk of midair collisions and near midair collisions between aircraft. TCAS functions separately from the ground-based air traffic control system.

There are three members of the TCAS family (as described in [1]):

- TCAS I: Provides proximity warning only, to assist the pilot in the visual acquisition of an intruder aircraft. It is intended for use by smaller commuter aircraft and by general aviation aircraft.

- TCAS II: Provides traffic advisories and resolution advisories (recommended escape maneuvers) in a vertical direction to avoid conflicting traffic. Airline aircraft, and larger commuter and business aircraft, will use TCAS II equipment.

- TCAS III: Still under development, but will provide traffic advisories and resolution advisories in the horizontal as well as the vertical direction to avoid conflicting traffic.

The object of our case study is the use of formal methods in describing the requirements of TCAS II.

RTCA Inc. (formerly called the Radio Technical Commission for Aeronautics) develops and publishes the "Minimum Operational Performance Standards" (MOPS) [2] to which TCAS II must conform. MOPS is essentially an English description of TCAS II requirements, except that the CAS Logic is also presented in pseudo-code. SC 147 (a committee within the RTCA) is charged with developing a specification for TCAS.

With the recommendation of the FAA Certification Office, the TCAS office commissioned Nancy Leveson (University of California at Irvine) to develop a formal specification. In particular, Leveson and her research group are formalizing the requirements of the CAS logic and is starting work on the surveillance system (to be completed by the fall of 1992). The use of formal methods was brought about by Leveson asking Mike Dewalt, an FAA resource specialist, whether he knew of a real application with which to try her safety analysis ideas and techniques. The FAA interest in formal methods appears to be a result of their interest in further clarifying the TCAS requirements and to obtain further confidence in the system.

10.1.1 Product and Process Profile

Size: 7,000 lines of pseudo-code for the CAS logic. Formal specification of CAS logic is about the same size as the pseudo-code.

Stages:

1981	FAA decision to develop and implement TCAS
1981-89	Ongoing development, simulation and evaluation of TCAS II. In 1983 RTCA publishes Minimal Operational Performance Specification (MOPS; [2]). MOPS description is in English with pseudo-code for the CAS Logic and a series of six revisions occur through to 1989.
1990 (June)	Leveson involvement starts as FAA and others become concerned with pseudo-code specification
1991 (Jan.)	RTCA committee adopts Leveson's approach to describing formally the CAS Logic requirements; dropping their own efforts using narrative English
1992 (April)	RTCA committee states that formal description of CAS logic ready for verification.
1992 (Spring)	Formal description of CAS logic and pseudo-code to undergo Independent Verification and Validation.

10.1.2 References

1. "Introduction to TCAS II." Federal Aviation Administration, U.S. Department of Transportation, March 1990.

2. "Minimum Operational Performance Standards for Traffic Alert and Collision Avoidance System (TCAS) Airborne Equipment." Volume 1, Consolidated edition, RTCA/DO-185, Radio Technical Commission for Aeronautics, September 1989.

3. "Experiences using Statecharts for a System Requirements Specification." N.G. Leveson, M. Heimdahl, H. Hildreth, J. Reese, R. Ortega, in Proc. of the 6th International Workshop on Software Specification and Design, Como, Italy, October 25-26, 1991, pp. 31-41.

10.2 Interview Summary

1. Date: 3 June 1992 (1/2 day) [Interview at U. Maryland]
   Organization: University of California at Irvine
   Respondent: Nancy Leveson
   Description: Interviewed by Dan Craigen (author)

2. Date: 8 April 1992 (1.5 hours)
   Organization: U.S. Federal Aviation Administration
   Respondent: Larry Nivert
   Description: Interviewed by Dan Craigen (author)

3. Date: 19 June 1992 (2 hours)
   Organization: Mitre Washington
   Respondent: Dave Lubkowski
   Description: Interviewed by Ted Ralston

4. Date: 24 April 1992
   Respondent: Craig White, Pilot
   Description: Interviewed by Dan Craigen (author)

Note 1: In this summary, we use the interview with Leveson as the main source of information. The interview reports on Nivert, Lubkowski and White augment the information obtained from Leveson.
Note 2: The material in this section has been augmented with information extracted from the references.

10.2.1 Organizational Context

The following organizations are currently involved with the CAS logic requirements (and, more generally, with TCAS):

- RTCA Inc., an organization that supports the development of aviation standards by facilitating exchanges between interested parties and issuing standards. The TCAS committee is run under the RTCA umbrella; the TCAS requirements working group is a part of the committee. The TCAS requirements working group consists of pilots, FAA representatives, airframe manufacturers, airline representatives, and other interested parties.

- FAA. At least two parts of the FAA are involved with TCAS. The TCAS program office is the R & D office responsible for TCAS development. These folks are aviation experts. There is also the FAA certification office, which must approve TCAS. Two "National Resource Specialists" (Mike Dewalt and Jim Treacy) are part of the certification office.

- University of California at Irvine, which has been developing the formal specification for TCAS under a grant from the TCAS program office.

- Mitre (Washington) and Lincoln Labs (Boston) have a long-term involvement with TCAS. Mitre and Lincoln Labs developed the initial TCAS II MOPS.

Leveson's group, who are responsible for writing the formal requirements document, consists of Leveson and some of her graduate students, all at the University of California at Irvine. There is a subgroup of

individuals of SC 147 who continually review and provide feedback on drafts of the formalism. At one point, the requirements working group was producing a less formal requirements document and this effort was viewed as being the official development. However, the less formal development was terminated and the work by Leveson's group was adopted as the official RTCA specification.

Most individuals involved with TCAS have engineering background. With the exception of a few select individuals and Leveson's group, there is very little in-depth understanding of software engineering principles and no knowledge of formal methods.

10.2.2 Project Context and History

Those involved in the development of the formal requirements for the CAS logic are Nancy Leveson (Ph.D.) and Ph.D. students at the University of California at Irvine. As noted above, domain experts from SC 147 regularly review and comment on the evolving mathematical specification of the CAS Logic and surveillance system.

The general purpose of the TCAS II system has been described above. From [1], seven TCAS system components are identified:

- Mode S/TCAS Control Panel: Selects and controls all TCAS elements.
- Mode S Transponder: Performs normal air traffic control functions of existing Mode A&C transponders. Also used for air-to-air data exchange between TCAS equipped aircraft to ensure coordinated, complementary resolution advisories.
- TCAS RF/Computer Unit: Performs airspace surveillance, intruder and own aircraft tracking, threat detection and resolution, and advisory generation.
- Antennas: TCAS includes directional and omni-directional transmitting and receiving antennas.
- Traffic Advisory Display: Shows the position of air traffic relative to the TCAS equipped aircraft so as to assist the pilot in visually acquiring an intruding aircraft.
- Resolution Advisory Display: A "vertical speed indicator" modified to indicate the vertical rate that must be achieved to maintain safe separation from threatening aircraft.
- Aural Annunciation: Displayed traffic and resolution advisories are augmented by synthetic voice advisories generated by the TCAS computer.

Also from [1], the CAS logic functions are described. Here, we present a brief summary:

- Tracking: Using surveillance reports each second, the CAS logic tracks target aircraft.
- Traffic Advisory: Range and altitude tests are performed and declaration of intruder.
- Threat Detection: Range and altitude tests are performed and declaration of threat.
- Resolution Advisory Selection: Once a threat is declared, a two-step process is used to determine the Resolution Advisory. First, select the sense (upward or downward) of the Resolution Advisory. Secondly, select the strength of the Resolution Advisory. For example,

"Don't Descend," "Don't Descend > 500 fpm."

- TCAS/TCAS Coordination: If there is an encounter between two TCAS equipped aircraft, the aircraft communicate with each other in order to ensure the selection of complementary resolution advisories.

- Advisory Annunciation: Used to alert pilots of a Resolution Advisory.

- Air/Ground Communications: Supports transmission of Resolution Advisorys to suitably equipped ground sites.

10.2.3 Application Goals

The purpose of the TCAS family of instruments (of which TCAS II is a member) is to reduce the risk of midair collisions between aircraft. TCAS functions separately from the ground-based air traffic control system. The U.S. Congress has mandated that TCAS must be installed on U.S. civil aviation airliners. Leveson, and her group, also had research interests in applying their techniques to a realistic application.

There are a number of novelties associated with the TCAS effort. It has been claimed that TCAS is the most complex avionics system in commercial aircraft. In addition, the formal methods being used (at least from a notational perspective) are new and evolving.

10.2.4 Formal Methods Factors

Formal methods were chosen because safety analysis requires a formal model and mathematical methods. Testing or execution of a specification does not provide sufficient assurance for most safety-critical applications.

The underlying mathematics of the formalism is a state machine semantics and the specification language is Statecharts-like [3]. While they have not performed any analysis to date, they do expect to use model-checking and theorem-proving techniques.

In selecting their approach, Leveson needed a model on which she could perform safety analysis. From her theoretical work on safety, she decided that a state machine was the most practical model to use for this goal. Among the existing reactive system specification languages based on state machines, Statecharts [3] was the closest to what she wanted and appeared to be the most useful for real systems (where large numbers of states would be involved). She also felt that Statecharts was the most useful for the complexity of the system being specified. Changes were made to Statecharts in accordance with criteria of readability, reviewability, writability, minimality (conciseness and compactness), completeness, and simplicity. Most of all, the specification had to be blackbox (no design information) and amenable to a formal mathematical analysis.

Besides changes to the Statechart language, Leveson's group has been using a tabular representation of "disjunctive normal form" to specify transition conditions. Experience to date has shown that this representation is more readable for engineers, and others on SC 147, to read than the usual symbolic style of predicate calculus formulae.

10.2.5 Formal Methods and Tools Usage

The formal methods work has been directed at the CAS Logic and now the surveillance system. While the techniques are based on a state machine representation, there have been a number of research issues pertaining to scalability, presentation style of specifications, and modifications to the Statecharts language. The formal methods work resulted in at least one significant change in the way the SC 147 committee was working: the informal English specification effort was terminated. To date, feedback on her language has been positive. Engineers and others not trained in formal methods are successfully reviewing and modifying the specification.

The only tools used on this project (by Leveson's group) have been LaTeX and some configuration management tools. There is a stated intention to experiment with various model checkers and theorem provers in the future. In the use of LaTeX, there has been a strong emphasis on producing readable documentation with extensive cross-referencing.

10.2.6 Results

In many respects, it is still too early to identify the main results of the project. The project is ongoing and the formal specification (along with the pseudo-code) has just made it into the Independent Verification and Validation phase. The purpose of the Independent Verification and Validation is to determine the degree of conformance between the pseudo-code and the formal description of the CAS Logic.

Leveson feels that the application goal of obtaining a specification of the CAS Logic which was still under intellectual control has been achieved. She feels that this is a major result, especially since errors have been found in the pseudo-code. Leveson reports that, in her opinion, there is a general increase in confidence of the requirements as a result of the work and, since it is more reviewable than the pseudo-code, quality is improving.

If Leveson were to do anything differently, it would be to maintain closer contact with the domain experts. Finally, she observed that the real time issues were rather easy (e.g., complete this task in x seconds) and that the notations and models used to formalize systems, should be presented in terms of the relevant cultural milieux.

10.2.7 Interview with Larry Nivert

Larry Nivert is an FAA employee who has been involved with TCAS for the past four years. Nivert works as part of an FAA R & D group which was asked, by the certification side of the FAA, to identify the requirements for the CAS Logic.

Nivert observed that the CAS logic had its genesis in prototypes of the mid-seventies. When work started on CAS, the FAA did not have any avionics software standards, though a standard (DO-178A) was effective in the early eighties. The FAA did not retrofit TCAS to meet DO-178A as they thought the effort would be onerous.

Nivert feels that Leveson's group has developed a sound draft of the CAS logic requirements. He noted that the requirements along with the pseudo-code will be sent out for Independent Verification and Validation in the spring of 1992. The Independent Verification and Validation effort will take eight months and will include 15 person-years of effort.

The formalism, as a result of the reverse engineering approach, appears to suffer from the inclusion of a substantial number of design decisions. There is a minor debate on whether a more abstract presentation of the requirements are needed. For example, industrial concerns want more abstraction so that they can have increased flexibility to implement CAS; currently, they are mandated to be in consonance with the pseudo-code. Nivert, on the other hand, feels that the current level of detail is sufficient for proceeding with Verification and Validation, and feels that there would not be a substantial benefit with proceeding with an abstraction.

Prior to the work by Leveson's group, most FAA specification work was in English and such specifications were continually picked apart. The formalism of the CAS logic has been readily accepted by the technical representatives of the committee as it fits with their engineering training. The use of formalism is a radical change for the RTCA and has strong support from industry (as represented by SC 147).

10.2.8 Interview with David Lubkowski

David Lubkowski is an employee of Mitre (Washington) and has been principally involved in the development of the pseudo-code specification of the CAS Logic.

The first TCAS specifications were written in English in the early seventies (for prototyping). These specifications evolved over the decade, based on the experiences arising from the use of the prototypes. In the early eighties, the English narrative was becoming difficult to handle and so the SC 147 committee decided to create a separate document that described the surveillance requirements and the CAS logic (only a brief description, not in great detail), which was used in bench tests and other evaluation testing. In the context of SC 147, an issue arose with the manufacturers on how flexible the definition of the CAS Logic should be. The manufacturers wanted a tight, strict definition to build to, while the developers wanted a looser logic. The manufacturers prevailed and this led to a second volume which was the original pseudo-code specification. The decision to use pseudo-code was based on what was considered to be "state-of-the-art" at the time and was driven by the manufacturers' needs for precision exceeding that of flowcharts. Mitre developed a special "design language," similar to PL/1, which was used for the pseudo-code, and the MOPS were expressed in this language. After awhile, complaints started to be raised that the pseudo-code was not at a high enough abstraction. Lubkowski claimed that the original SC 147 was satisfied with the pseudo-code (for the most part) because they had a design specification that, in their minds, was already "validated" because of the link to the MOPS. No explicit validation procedures were required.

The pseudo-code approach held for almost ten years. The catalyst for a change to a formal approach was Boeing's discovery of the "descent inhibit altitude change" error on their simulator. Boeing recommended that Independent Verification and Validation be performed on the requirements and, as a result of the ensuing discussions, SC 147 concluded that an understandable formalism was required. The "transition table" approach was finally chosen.

Lubkowski feels that the "transition table" approach is liked by all project participants, even though it is still too detailed. Lubkowski is concerned whether the formalism will help with testing and simulation of requirements.

Lubkowski also noted that, for safety reasons, a requirements simulator is needed which would be used to test TCAS induced errors. Currently, an "encounter model generator" is used, but this can not be used to determine the risk with or without TCAS for a given set of circumstances that might result in a near midair collision.

10.2.9 Interview with a pilot

On a flight from Dallas-Fort Worth to Toronto, Dan Craigen took the opportunity of sending a note to the pilot, asking whether he had any observations on TCAS, in particular, and on the ongoing evolution of the computerisation of the cockpit. His written response follows:

> ``Yes, I am aware of TCAS. Most of our fleet has been outfitted but this older 727-100 doesn't have it as it is due to be retired soon. It is generally thought to be a system with potential, although there are still too many false alarms in the terminal areas of high density airports (i.e., Dallas, Chicago). High rate of climb stage 3 aircraft such as 757s and S80s will trigger TCAS of arrival aircraft at a constant cruise altitude. More software development and more control parameters might be the answer.
>
> I am an electrical engineer by my background although I have never used my undergraduate training in the field. I, of course, see the advantages of more electronic aids to the pilot but am

still somewhat cautious about back-up/fail-safe measures.

I flew the 767/757 for several years and loved the avionics but grew to be very careful about complacency of the automatic systems. In general, the state-of-the-art electronics are very good but human pilot nature is slow to change to accept it. I think electronic modernization of the cockpit is inevitable and good but am very concerned about what happens to pilot skills and ability to continue when multiple failures occur after years of problem-free flying."

10.3 Evaluation

10.3.1 Background

Product:

The purpose of the TCAS family of instruments (of which TCAS II is a member) is to reduce the risk of midair collisions between aircraft. TCAS functions separately from the ground-based air traffic control system. Formal methods were used on the requirements for the CAS Logic and are being used on the surveillance system.

Industrial Process:

Use of a formal notation, based on state machines, to reverse engineer the requirements for the CAS Logic. Review by members of the SC 147 committee and Independent V & V of formalism and pseudo-code currently underway. Earlier development of TCAS made extensive use of simulation.

Scale:

7,000 lines of pseudo-code for the CAS logic. Formal specification of CAS logic is about the same size as the pseudo-code.

Formal Methods Process:

Use of Statecharts-like language with both notational and notional changes. Extensive review of specification.

Major Results:

Development of formal specification of the CAS Logic that is reviewable and under intellectual control.

Interview Profile:

Half-day interview with Leveson, shorter interviews with Nivert (FAA rep.), Lubkowski (Mitre; developer of pseudo-code specification) and White (pilot).

10.3.2 Product Features

Client satisfaction	+	(1)
Cost	n/a	(2)
Impact of product	n/a	(3)
Quality	n/a	(4)
Time to market	n/a	(5)

(1) Identifying the clients as the FAA and the SC 147 committee, there appears to be general happiness with the formal specification. It appears to be sound and reviewable.

(2) No information.

(3) Insufficient information.

(4) Insufficient information. Currently in Independent Verification and Validation. There are general impressions of improved quality, but this is too subjective at this point.

(5) Insufficient information.

10.3.3 Process Features

General process effects

Cost	n/a	(6)
Impact of process	+	(7)
Pedagogical	+	(8)
Tools	n/a	(9)

Specific process effects

Design	+	(10)
Reusable components	n/a	(11)
Maintainability	n/a	(12)
Requirements capture	+	(13)
V & V	n/a	(14)

(6) Insufficient information.

(7) FAA and SC 147 found the formalism easier to comprehend and closer to their engineering backgrounds. As a result, the parallel English specification project was terminated. It is expected that the TCAS III development will use formal methods in the same form.

(8) FAA and SC 147 seem to have readily adopted the ideas and are considering the use of formal techniques in other projects (notably, TCAS III).

(9) No formal methods tools were used or available.

(10) The formalism was reviewable and led to intellectual control of the specification. The formal description is generally conceded to be more tractable as it excludes design information that is incorporated into the pseudo-code.

(11) Not applicable.

(12) Insufficient information.

(13) The formalism helped to clarify various TCAS requirements by being more reviewable and tractable than the pseudo-code.

(14) Results from Independent Verification and Validation not yet available.

10.3.4 Observations

In the interview notes, Lubkowski mentions a concern on whether the formalism will help with testing and simulation of requirements. Leveson reports that Professor Elaine Weyuker, of New York University, is working on techniques to generate test data directly from the requirements specification.

Lubkowski also noted a complaint that the formal specification was too detailed. However, according to

Leveson, it is no more detailed than the English language specification that was being developed by Mitre.

Leveson has also indicated some concerns about the "lack of abstraction" comments that have been made about the requirements, since it suggests that the formal description includes detailed design information. In fact, the formalism is a black-box specification that abstracts the design details of the pseudo-code representation and leaves a description in terms of externally visible behaviour.

This project has been an interesting exercise in the transition of formal methods into a community that had no familiarity with formal methods, and little familiarity with software engineering. Presenting the methods in terms that were familiar to the largely engineering audience appears to have been fundamental to the apparent success of the project. Our main caveat is that this project has not been completed and the results of the Independent Verification and Validation are pending.

10.3.5 Formal Methods R & D Summary

10.3.5.1 Methods

A rigorous graphical and tabular notation was used to specify the CAS Logic. Communication of the formal ideas to the engineers was a success, with a small learning curve.

10.3.5.2 Tools

No formal methods tools were used, though Leveson's group has now started developing and selecting formal methods tools for performing analysis on the specification.

10.3.5.3 Recommendations to the FM Research Community

This case study suggests the importance of couching formal methods concepts in terms of the cultural milieux to which these concepts are to be transitioned. There are important pragmatic lessons here in that the notations of interest to computing scientists may not be as acceptable to other disciplines.

10.4 Conclusions

This project appears to have been a successful application of formal methods through the development of a reviewable and intellectually manageable formal requirements specification. The general consensus of all participants is that the formalism is better than the pseudo-code and certainly better than the English language specification that was being developed in parallel. Even without using analysis tools, there appears to have been general increase in confidence and understanding of the CAS Logic requirements.

From a transition perspective, this has been a successful transition of formal methods ideas to the aviation community. An important aspect of this success has been the use of notations and notions to which the engineers can relate.

11 INMOS TRANSPUTER
USE OF FORMAL METHODS IN HARDWARE VERIFICATION

11.1 Case Description

In 1985, a small group of designers at INMOS Ltd. in Bristol, England, began exploring the use of formal program specification, transformation and proof techniques in designing microprocessors. INMOS manufactures advanced microprocessors, and their best known product is the Transputer family of 32-bit VLSI circuits with a unique architecture designed for concurrent multiprocessor applications (processor, memory and communication channels are self-contained on each Transputer chip).

This case consists of three inter-related projects, all of which use formal methods in some aspect of the design or development of components of three generations of the INMOS Transputer:

1. The "floating point" project: the use of Z to specify the IEEE Floating Point Standard which was applied to two successive generations of Transputer (a software implementation, and a hardware implementation).
2. The use of Z and Occam to design a scheduler for the T 800 Transputer.
3. The use of Communicating Sequential Processes and Calculus of Communicating Systems plus a "refinement checker" in the design and verification of a new feature of the T9000 Transputer, the Virtual Channel Processor (VCP).

The formal methods work began with several experiments that culminated in the application of Z and the Occam Transformation System [7] to specify the IEEE floating point standard. The results of this formal specification were then applied to a package of subroutines that had been written to provide floating point arithmetic for the T414, since the Transputers did not have floating point hardware at the time. This program was specified in Z and implemented in Occam, which is a multi-purpose language derived from CSP that is used by INMOS both as a programming language for the Transputer and as a hardware description language [5] This package was then used in the design of the successor Transputer's hardware floating point unit, using a similar approach. A scheduler for the T800 was also formally specified. Lastly, formal methods were used in design and verification of a new feature on the third generation Transputer (T9000), the Virtual Channel Processor. The VCP is a device which allows any number of logical connections between two processors to be implemented by a single physical connection.

11.1.1 Product and Process Profile

Size:	T414 FPU:	10 pages of Z specification of IEEE 754 standard for floating point arithmetic; a few hundred lines of Occam code as a software subroutine implementing the IEEE 754 standard
	T800 FPU:	microcode controlled hardware
	T9000 VCP:	10*6 states; unknown number of Occam transformations

Stages:

1985	initiation of research project on applying formal methods to hardware design
1986	Z specification of IEEE 754 standard for floating point arithmetic
1986-87	applying the formalization of IEEE standard to T414 floating point unit design by a software subroutine
1987-88	formal specification and verification of T800 FPU
1988-89	initiation of T9000 design; formal design and verification of Virtual Channel Processor

	for T9000
1990-91	completion of formal verification of VCP

11.1.2 Issues of Interest

The INMOS case is included in the study because it is one of the few examples of the use of formal methods in a commercial setting for the specification and verification of a commercial hardware product. As well, productivity improvement claims have been made anecdotally by INMOS personnel and outsiders who are familiar with this work to the effect the formalization effort reduced testing time over conventional methods.

11.1.3 References

1. May, David; Barrett, Geoff; and Shepherd, David. "Designing Chips that Work." *Philosophical Transactions of the Royal Society of London*, A 339, pp.3-19, 1992.

2. Barrett, G. "Formal Methods Applied to a Floating Point Number System." *IEEE Transactions on Software Engineering*, 15, pp. 611-621, 1989.

3. Griffiths, V.A. "Specification, Refinement and Verification of the VCP." Internal INMOS paper, January 24, 1992.

4. Three papers by David May in *The Year of Programming*. University of Texas at Austin, 1988, pp. 65-129.

5. INMOS Ltd. *The Occam Programming Manual*. Prentice-Hall International, Hemel Hempstead, UK, 1984.

6. Good, D.I. "Mechanical proofs about computer programs." *Mathematical Logic and Programming Languages*, C.A.R. Hoare and J.C. Shepherdson, eds., pp.55-74, Prentice-Hall International, Englewood Cliffs, N.J., 1985.

7. Goldsmith, M.H. "The Oxford Occam transformation system (version 0.1)." draft user documentation, Oxford University Programming Research Group.

11.2 Interview Summary

Date of Interview:	May 14, 1992
Organization:	INMOS Ltd.(a subsidiary of SGS-Thomson), Almondsbury, UK
Respondents:	David May, Geoff Barrett, Victoria Griffiths and Peter Thompson.
Interviewers:	Dan Craigen and Ted Ralston (author)

11.2.1 Organizational Context

INMOS Ltd. is a manufacturer of microprocessors and microcomputer products. The company was founded in 1982 to develop, manufacture and sell a microprocessor called the Transputer. The Transputer has a novel architecture that combines processor, memory and communication channels all on one integrated circuit. INMOS has had its commercial problems over the last decade, first with respect to delays in production of the first generation of Transputer (the T400 series), followed by sluggish sales of succeeding generations (the T800 and the recently introduced T9000). Starting as an independent company in 1982, INMOS has been purchased twice in the intervening decade, and is currently a part of the SGS-Thomson Microelectronics Group.

11.2.2 Project Context and History

During the period 1980-85 there was a general increase in awareness of formal methods and the putative benefits of using the technology, on the part of David May. However, there was no serious use of the technology. The possible use of formal methods was triggered by the Royal Society meeting at which Don Good, on the invitation of Tony Hoare, gave a presentation on the Computational Logic, Inc., use of formal methods up to that point. (Primarily, on the Gypsy Verification Environment.) The work started with a software implementation of the floating point unit and was followed by the move to hardware. Questions arose as to how you would verify that the unit was in consonance with the IEEE standard (ANSI/IEEE Std. 754-1985) and, in fact, what the standard was precisely. Certainly, there was a large test suite associated with the standard, but this seemed to be conceptually inadequate.

A problem had arisen during testing of the original software implementation of the floating point arithmetic. This program was not large but had some complexities that were producing errors during testing, the root cause of which was proving difficult to find and resolve. The program was tested in a relatively conventional way, by executing all the arithmetic operations, with four rounding modes, on a large number of operand values (50,000 per operand), and then comparing results to a reference implementation used widely in CAD. The rounding algorithm turned out to be the source of the errors (double rounding), which was discovered through the formal specification.

This experience led to the use of a similar approach in verifying the floating point hardware for the next generation Transputer, the T800. The FPU in the T800 is a datapath controlled by microcode. The formally specified Z-to-Occam floating point program developed for the T414 was used as a model for designing the T800 FPU. A lower level implementation, also in Occam, was used to describe the detailed operations of the FPU hardware. The high-level Occam program and the lower level Occam description of the hardware were then compared using Occam transformations (using the Occam Transformation System which was developed at Oxford contemporaneously).

11.2.3 Application Goals

From both a software and hardware perspective, there were concerns about how to test a floating point unit: conventional testing by simulation would take a long time and, consequently, negatively effect the

time to market. Prototypes are not used at INMOS for testing complex hardware. INMOS wanted to avoid a large amount of testing and to reduce design time. They were also looking for increased confidence in the design.

11.2.4 Formal Methods Factors

At this time, there was some contact with Oxford for the implementation of transformation tools related to Occam. Hoare and May discussed the possible use of formal methods on the FPU and felt that it was a sufficiently tractable example upon which to experiment with the technology.

The team consisted of seven people, broken down into the following tasks:

1 computer architect for product specification
1 system designer for microcode, logic, and datapath
1 mathematician for formal specification and proof
2 electronic engineers for circuit design and layout
1 programmer for compiler and instruction set tuning
1 mathematician for scientific function library

Geoff Barrett (one of the mathematicians) was key to the success of the FPU effort. He is a powerful theoretician but is also interested in trying his techniques, or developing new techniques, on real applications. Barrett is a Ph.D. mathematician without any particular training in hardware. Barrett used Z and reworked the IEEE standard in Z, using no mechanized techniques to refine the Z specification into Occam.

11.2.5 Formal Methods and Tools Usage

The basic INMOS philosophy concerning formal methods has been to use them on tractable subproblems, where otherwise the amount of testing would explode. In the case of the T9000 VCP, it was not known how to test it adequately. There is a large degree of nondeterminism in the VCP. There is a lack of adequacy in the test suites from CAD vendors and the INMOS group focuses its testing in the areas where they lack confidence.

11.2.6 T800 Floating Point Unit

The Floating Point Unit (FPU) of the T800 was designed as a separate microcoded processor, which enabled major parts of the floating point logic from the earlier T414 to be reused. To check the microcode algorithms, they used transformational schemes and used an embryonic transformation tool being developed by Oxford. The microcode was being related to the high level software algorithm.

```
Z ----------------> Occam  <------------------> Occam
    Geoff                  David
    Barrett                Shepherd
```

The low level Occam was complicated and convoluted and, ultimately, was made to look very much like the microcode. Shepherd worked closely with the engineer working on the project and his presence was useful in flushing out errors.

They were ultimately able to transform all the floating point instructions. The engineer for the T800 would still claim that the "conventional approaches" (i.e., testing) would have uncovered the various problems. However, there would have been questions as to whether the design worked, which would have involved an additional three months of coverage testing to gain the right degree of confidence (in the absence of the use of formal methods). The three months of coverage testing was dropped from the plan. Shepherd was also able to design an economical test suite (effectively, white box testing).

The FPU work was invisible to the organization outside of the project. Pursuing formal methods was deliberately kept as a low profile by David May; he didn't want to do anything big and was experimenting to see how the technology worked. Until recently, this approach continued to be followed at INMOS and it is only within the last 6-12 months that it has started becoming more public. May views formal methods as an experimental technology and looks for opportunities to put people in touch with each other. Essentially, wait until a project is in need of help where he believes that formal methods could make a difference.

11.2.7 T800 Scheduler

The Scheduler was interesting because it is primarily a concurrent system, and May, et al., wanted to have some experience using their "bridging semantics" (as they call the transformations between the various Occam levels) on a concurrency problem. CSP could not be used because the Scheduler had "priorities" (which involved refusal sets) for which a CSP model for Occam implementation could not be done. Barrett used a technique of upwards and downwards simulation together with a slightly weakened proof approach. This developed an analog in the operational semantics which is quite close to Morgan's refinement rules. This approach became the basis for the work on the T9000's VCP.

Barrett's Ph.D. thesis was the scheduler of the T800, but was rather generic, hence some of the work carries over to the T9000. To perform the work, Barrett produced an operational semantics for Occam and the microcode as described above. Barrett noted Z was certainly not good for reactive/event based systems. CSP is closer to what was needed but does not handle priorities, for example. It was also not obvious how to develop a denotational semantics for the Transputer instruction set. The following diagram depicts the approach used.

```
CSP w/ priorities ----> generic scheduler-----> transputer instruction set
  |                                  |
  |                                  |
```

operational semantics <-----ad hoc refinement---> operational semantics

This work was not product oriented; it was a Ph.D. thesis and an idealized version of the transputer instruction set was used.

11.2.8 T9000 Virtual Channel Processor

Verification of the VCP in the T9000 was a big concern since the processor is much more complex than the T800 and they had had difficulties with the T800. The VCP multiplexes channels over a single communications link and uses a packetized protocol. Thompson noted that this is a distributed problem with an arbitrary number of channels and packets could be routed through a packet switching system, hence an arbitrary number of processors. Therefore, there are serious testing and simulation problems; though engineers would prefer to use simulation (because they understand it better).

There was a time gap between the T800 and T9000 projects. Few of the individuals who had been around for the T800 project were still around for the T9000 project and there were no initial plans for the use of formal methods on the design or validation of the T9000. Most of the initial version of the T9000 was developed without formal methods. However, the T9000 is undergoing a redesign effort which is using formal methods, especially with respect to the pipelining and the communications processor.

Initially, they started constructing large state machines, but real questions arose about whether these state machines were correct. Hence, at Barrett's suggestion, they broke the state machines into much smaller pieces and combined them with the CSP parallel operator, renaming, etc. A second more concrete representation was also developed. This more concrete representation was used to derive a simulator model of the VCP and has been used by the software team for implementing software for the T9000. VHDL simulation vectors were also derived from the concrete representation. In parallel with the above, there was an attempt to develop a highly abstract description of channels: this was successful and became part of the refinement proofs.

A number of inconsistencies and errors with the VCP design were found through the above process, which overall took four person-years of effort.

11.2.9 Tools

Barrett did not want to become involved in doing the proofs for the T9000 work, so he ended up developing a tool (for Vicky Griffiths) which did much of the refinement. This turned out to be a useful way of transferring the knowledge to Griffiths, as well. She was involved in the incremental development and use of the tool. To understand the tool, one needs to understand traces and refusal sets. The tool compares finite state machines for refinement. Their approach is a few orders of magnitude (as measured by states) from the capabilities of the model checking algorithms (used by Clark and others). They might need to change their technology and use model checking techniques.

From a tool perspective, INMOS has primarily been working with refinement checkers. Barrett did not use any tools on the floating point unit. There were no Z type checkers at that time. Shepherd did use an early version of the Occam Transformation System from Oxford. The transformation tool is based on the algebraic laws of Occam. Some aspects of the transformation had to be performed by hand. There was also a separate tool that transformed parts of the Occam programs into VHDL. No tools were used with the Scheduler.

For the T9000 a refinement checker was written (part-time over a month), along with transformation

programs and they used Synopsis' CAD tool for VHDL simulation. "Formal Systems Ltd" (a spin-off company from Oxford University) has been contracted to improve the refinement checker. Effectively, their tools were stand-alone tools.

INMOS would like a tool that would generate VHDL from the state machines.

11.2.10 Results

INMOS was awarded the Queen's award and the T800 had its development time shortened by three months (according to May), thereby saving some three million pounds Sterling (US$ 4.5 million roughly). They achieved time-to-market and quality goals. Note, May observed that unlike software products, hardware vendors do not have the option of shipping faulty products, since the development costs are higher and profit margins narrower and a poor yield and/or error-based recall can wipe out any profits.

May noted that, from a manager's perspective, there is a delicate point of contact between the formal methodists and the engineers. It needs to be carefully handled. Barrett claims that he would not do anything particularly different.

11.3 Evaluation

11.3.1 Background

Product:

Components of three generations of commercial microprocessors (the floating point arithmetic units of the T414 and T800 Transputers, the T800 Scheduler, and three interdependent parts of the T9000: the Virtual Channel Processor, the control link, and the C104 switching circuit).

Industrial Process:

A conventional design and development process for silicon integrated circuits with formal specification and verification playing the role of design verification.

Scale:

The Transputer is a 32-bit VLSI circuit with a special architecture and microcode for concurrent processing. The FPU specification involved one large predicate (order 100 pages of Z) for the microcode ranging from 4 microwords to 90 microwords in size; the Occam software implementation of the floating point was a few thousand lines of Occam. The VCP involves more than 10*6 states.

Formal Methods Process:

The FPU involved specifying the original floating point software package in Z, then implementing in Occam, followed by a comparison of high-level Occam with lower level Occam implementation of the hardware behaviour by means of transformations using the Occam Transformation System tool. The VCP involved a formal specification using state tables, which were refined to generate Occam models using a refinement checking tool, which was then converted to VHDL representations for simulation using test vectors partially derived from the formal specification.

Major Results:

The "design verification" approach has been successfully integrated into the design process at INMOS as an accepted adjunct to traditional testing, simulation, and validation activities; reduction in testing time and improvement in confidence in the quality of the Transputers.

Interview Profile:

One day of separate interviews with individuals in the formal methods group, and one engineer.

11.3.2 Product Features

Client satisfaction	+	(1)
Cost	+	(2)
Impact of product	n/a	(3)
Quality	+	(4)
Time to market	n/a	(5)

(1) In this case, the client is the Transputer design group who have benefited from the formal methods group's efforts in resolving a few tough problems.

(2) Three-million pound saving on testing T800 FPU plus a three-month reduction of testing time helped keep overall T800 price competitive.

(3) Not enough information to evaluate, although the Transputer family is the main strategic product of INMOS (but not necessarily SGS-Thomson).

(4) Reliability (in terms of correctness) of the FPU and the VCP has been enhanced by the formal methods.

(5) While the three-month reduction in testing time for the T800 FPU impacted cost, production factors retarded the delivery of the T800. The T9000 has been delayed slightly by the need to re-design some components; the formal verification, while not initially included in the design plan, has neither retarded nor improved progress on the re-design. However, we did not collect enough information on delivery schedules and market requirements to determine definitively any positive or negative impact as a result of using formal methods.

11.3.3 Process Features

General Process Features

Cost	+	(6)
Impact of Process	+	(7)
Pedagogical	+	(8)
Tools	+	(9)

Specific Process Features

Design	+	(10)
Reusable Components	+	(11)
Maintainability	n/a	(12)
Requirements Capture	0	(13)
V & V	+	(14)

(6) The cost of introducing the formal methods has been minimal, given the relatively small number of people involved compared against the overall cost of the design, development and manufacturing of a major silicon fabrication effort. Coupled with the positive cost savings resulting from a three-month reduction in time by not having to perform additional coverage testing, the net impact of the process is positive.

(7) The formalization has improved confidence in the quality of the components, and the interaction between the engineers and the formal methods team has improved the understanding of the behaviour of the product, both intellectually and through the improvement in the testing stage. The fact the formal methods group is now more closely integrated into the critical design/development path also indicates a positive impact.

(8) The insight into the source of errors (and how to avoid or fix them) through the use of the formal methods has improved the engineering of the Transputer. Personnel in both the formal methods group and on the engineering staff have learned important lessons through the cross-fertilization.

(9) Tools (with the exception of the Occam Transformation System) were developed as needed in-house. The Occam Transformation System was used opportunistically as it was already under development and met the requirements INMOS had. The in-house tools were adequate in terms of robustness, although the refinement checker did not have enough memory or computation power for all the states of the VCP.

(10) The hardware design for the T800 FPU was verified more rapidly through the use of the previous floating point arithmetic software that was specified in Z and implemented in Occam than would have been the case by simulation and testing alone. The VCP re-design process used formal specification and verification to verify the design of the VCP. The inclusion of formal methods into the design process at INMOS also reflects a positive impact on design.

(11) The Z-to-Occam model for the FPU was developed with reusability in mind when it came to verifying the T800 FPU hardware. Also, the formal specification of the T800 Scheduler uncovered design features that were reused in the VCP design.

(12) No information on maintenance on shipped Transputers was collected.

(13) The formal specification of the IEEE floating point standard helped to clarify several ambiguities in the standard, which were used in eliciting and negotiating requirements for the T800 FPU later.

(14) No information on an Independent V & V on any of the products was collected. The formal methods were used to sharpen the functional testing by simulation on the T800 FPU. They also were used to generate what INMOS calls "intelligent test vectors" for the VHDL simulations run on the T9000 VCP.

11.3.4 Formal Methods R & D Analysis

11.3.4.1 Method

Specification:

Z and CSP were used as the specification languages. The inability of Z and CSP to handle timing constraints effectively indicates an area that needs further investigation.

Design and Implementation:

CSP and CCS models were used in the design of the VCP. The attempt to use a formal specification of the block structure of the VCP as a stepping-stone to checking the VHDL models points toward a research need to investigate ways to bridge the gap between these design

representations (possibly through application of other model checking approaches).

Testing:

Simulation and conventional testing methods were used in addition to white box testing derived from the formal specification.

Just as automated checking of silicon design rules and circuit layouts are a standard part of present commercial CAD systems, it seems from this case that larger designs (e.g., greater than 10*7 transistors) will be constructed from independent modules and that automated checking that these modules conform to a specification will become essential for future CAD systems.

Deriving simulation vectors from the concrete specification was successful and suggests further research on deriving simulation cases from the higher-level formal specification might prove valuable.

Tools:

The Occam Transformation System, a Z editor and type checker, and a in-house refinement checker (a proof correctness checker) were used.

This case suggests that future silicon design (greater than 10*7 device density) will require more tools that allow for the formal design of the system at the microcode level. This means a proof system for formal specifications written at a high-level of abstraction in languages such as Z or VDM, silicon compilers to compile intermediate representations into HDL implementations, a proof system to check HDL designs, and a layout system to generate and check layout from HDL. For example, INMOS is experimenting with defining axioms for the behaviour of low level modules in the HDL library which can be used by the HOL theorem prover to verify HDL designs.

Based on the VCP effort, INMOS would like a tool that can generate VHDL from the state machines.

The commercial silicon compiler was assumed to be correct (in part "verified" by simulations), but in order to carry on this work in a future with greater density and complexity demands these compilers may have to be verified formally (as in the work of Claesen at IMEC in Leuven, Belgium).

11.3.4.2 Recommendations to FM Research Community

The INMOS use of formal methods illustrates two primary needs that should be addressed by the research community. First, continued work is necessary to understand the relationship between formal proving techniques, such as those used in the VCP project, to testing strategies; and, second, how formal specification and verification techniques can fit into advanced CAD systems as new circuit designs move into the VLSI regime.

With respect to testing, the INMOS case appears to show some potential for ways to speed-up coverage testing, as well as suggesting an approach to specification-based testing.

With respect to coverage testing, formal objects such as logical formulae seem to be amenable to case analysis, as are recursive expressions and algebraic equations such as those produced by Occam. In addition to case analysis, proofs such as those conducted in the verification of the VCP can help assure

all parts of the proposition. (In the INMOS case, this proposition is whether all the states of the VCP were covered and satisfied the specification.) It must be noted that this is highly dependent on theory being well enough developed, and more research on theory is needed before this can be achieved.

Specification-based testing, in the context of the INMOS experience, is the analysis of formal specifications of complex hardware modules to generate tests to uncover errors either in the specification itself or in characteristics of the implementation that might not be tested otherwise. It would seem to be a potentially fruitful area of research to explore the use of formal specifications as a basis for systematic generation of tests for both the specification and from the specification for code which purports to be its implementation.

With respect to advanced CAD, the emergence over the last few years of automated checking of silicon design rules and circuit layout and connectivity as standard capabilities points to the potential for further advances in design techniques. The INMOS experience in designing the VCP illustrates an emerging need for better techniques for understanding the behavior of circuit designs which involve device densities of 10*7 or greater. Two areas in which formalization appears to hold potential is in the formal specification of the behavior of modules and their interfaces, and the use of formal specification, transformation and proof in design synthesis.

With respect to the behaviour of modules and their interfaces, automated checking that the composition of these modules corresponds to an overall specification is one possible avenue (as was performed on the VCP, although it was unable to reach all the possible states).

With respect to synthesis, the control logic of the VCP was produced by synthesis from VHDL descriptions. This reveals the importance of the input to synthesis tools requiring extensive use of behavioural description, for which a sound understanding of the semantics of the behavioral description language could help in reducing the simulation needed.

11.4 Conclusions

The most striking feature of the INMOS case is the effectiveness of what David May calls the "softly, softly" approach. This approach is the opposite of the approach promulgated in U.K. Ministry of Defence Interim Standard 00-55 which mandates the use of formal methods for certain classes of safety critical systems procured by the Ministry of Defence. The INMOS approach is a combination of opportunistic problem identification (to match the capabilities of the formal methods) with a small hybrid team of good engineers and mathematicians. This approach appears to offer a successful route for introducing formal techniques into a dedicated hardware engineering environment.

The strategy on tool development also reflects a needs-driven approach (as opposed to the "build the tools and they will come" approach evident in other projects). The tools that were developed were driven by the needs of the application developers, not the reverse. An example is the awareness of, and eventual integration with, commercial CAD tools (especially through the VHDL route) is an interesting convergence with another "environment"

The so-called "state tables" which were used in the VCP specification and verification represented another tabular form of intermediate representation, similar though different than the tables used in the Darlington and TCAS projects. This intermediate representation comes in two forms, both of which carry through concepts from the formal specification. The first is in the form of a state machine diagram and the second is in the form of tables which show the state transition name and behaviour.

As in several other cases in the UK, the presence of a group of experts relatively close at hand (in Oxford) made a difference in the uptake and success of the experience. A larger question is whether this presence remains as important as the use of the method in-house matures over time.

12 HEWLETT-PACKARD MEDICAL INSTRUMENTS ANALYTICAL INFORMATION BASE (AIB) COMPONENT MONITORING SYSTEM

12.1 Case Description

The Analytical Information Base (AIB) is part of an application running within a patient monitor called the Component Monitoring System (CMS). The CMS is used in intensive care units and operating rooms. A bedside unit monitors a patient and transmits data and alarms to a nursing station. Displays are available at both sites. The CMS allows for the easy insertion and removal of modules that monitor specific patient characteristics.

The AIB is used by the ST Measurement application to maintain information on the patient being monitored. The information is then available for batch review. Accountability is involved in that alarms also yield audit trails; that is, database capabilities provide both patient vital sign records and analysis.

Although the AIB is not viewed as being life-critical, the quality of embedded software is of increasing concern in the medical instruments area. In particular, the evolution of medical instruments has been such that there is an increasing software component and, as perceived by some of the Hewlett-Packard representatives we interviewed, there is a strong possibility that regulators, such as the U.S. Food and Drug Administration (FDA), will impose strict standards on software. For example, in 1990 the U.S. Congress passed the "Safe Medical Devices Act." As a consequence, the FDA wants to see links between product testing and product claims.

12.1.1 Product and Process Profile

Size: 55 pages of HP-SL, of which 39 pages specify database actions and 16 pages pertain to interface issues. 1290 lines of specification and design, and 4390 lines of delivered code (plus 4580 lines of support code).

Stages:

January 1991	Agreement to launch project.
March	Detailed planning meeting.
May	Specification work starts at Bristol.
June	Requirements workshop, at Bristol.
July	Specification review.
October	Design review, part 1.
November	Design review, part 2.
December	First code review.
January 1992	Integration testing complete.
February	Other code reviews.
March	Acceptance by Waltham starts.

12.1.2 References

1. Bear, Stephen. ``An Overview of HP-SL." In VDM 91 *Proceedings of the 4th International Symposium of VDM Europe*, October 1991, Springer-Verlag, Lecture Notes in Computer Science 551, pages 571-587.

2. Goldsack, P., Rush, T. "Specifying an Electronic Mail System with HP-SL." *Hewlett Packard Journal*, December 1991, pages 32--39.

3. Harry, P.; Rush, T. "Specifying Real-Time Behaviour in HP-SL." *Hewlett Packard Journal*, December 1991, pages 40--45.

4. Dollin, C. "The HP-ST Toolset." In VDM91 *Proceedings of the 4th International Symposium of VDM Europe*, October 1991, Springer-Verlag, Lecture Notes in Computer Science 551, pages 687--688.

5. Ladeau, B.; Freeman C. "Using Formal Specification for Product Development." *Hewlett Packard Journal*, December 1991, pages 46-50.

6. Cyrus, J., et al. "Formal Specification and Structured Design in Software Development." *Hewlett Packard Journal*, December 1991, pages 51-58.

7. "Applying HP-SL to the Prometheus Defibrillator." Video, Hewlett Packard Laboratories, Bristol, 1992.

12.2 Interview Summary

1. Date: 12 May 1992 (half day)
   Respondents: Stephen Bear and Paul Harry
   Organization: Applied Methods Group, Hewlett-Packard Laboratory, Bristol, England
   Description: Interviewed by Dan Craigen (author) and Ted Ralston

2. Date: 4 June 1992 (half day)
   Respondents: Rob Ladeau and Brian Connolly
   Organization: Hewlett-Packard Medical Products Division, Waltham, MA
   Description: Interviewed by Dan Craigen (author) and Susan Gerhart

Note 1: We have merged the information from the two interviews into a single summary.

12.2.1 Organizational Context

Two divisions of Hewlett-Packard were involved in the development of the Analytical Information Base (AIB): the Hewlett-Packard Research and Development laboratory at Bristol, England, and the Medical Products Division located at Waltham, Massachusetts, U.S.A. At Bristol, the Applied Methods Group and the Specification Technology Group were involved. At Waltham, a small project team (the ST project team) within the Monitoring Systems lab, along with the Software Quality Engineering group, were involved.

Between 1989 and 1992 the Applied Methods Group worked on a series of collaborative projects including the AIB project. The objectives of this group were to:

- Demonstrate that formal specification methods are applicable to HP products, and provide significant benefits;
- Demonstrate that formal specification techniques can be used by HP engineers; and
- Transfer formal specification technology from HP Labs to a number of product divisions.

In parallel, the Specification Technology Group developed a formal specification language, the Hewlett-Packard Specification Language (which is a VDM derivative), and supporting tools, HP-ST. Towards the end of 1991 the two HP Labs projects were merged. Cardiac Card Systems produces medical monitoring products. The Waltham Software Quality Engineering group has a role of inspecting systems for quality and to improve developmental practices at Waltham.

From an educational perspective, HP management generally have degrees in Electrical Engineering. At the engineering level, Electrical Engineering and Computer Science degrees are predominant. At the research laboratories, Master's or Ph.Ds in Computer Science predominate, though there are also engineers, mathematicians and physicists.

At Waltham, there has been a substantial shift during the past ten years from hardware to software. Currently, 80% of the engineers are involved with software. A decade ago, HP was still fielding products that were solely hardware; no embedded software.

There is no HP-wide system development process except, perhaps, for the use of phase reviews. Product development groups have a large degree of independence in the choice of technologies they apply.

12.2.2 Project Context and History

As noted above, the Analytical Information Base (AIB) is a real-time database that is to be added to an application within the Component Monitoring System. Detailed work on the AIB started May 1991 and acceptance testing started in Waltham in March 1992. There are about 35 person years of effort involved with the development of the Component Monitoring System; of this, only nine person-months were directed at the AIB.

Educationally, the personnel involved with the AIB project had backgrounds consistent with their divisions at HP. Both Ladeau and Connolly have M.Sc. degrees (Computing Science and Computer Engineering, respectively). Bear has a Ph.D. in mathematics and Harry has degrees in mathematics and physics, along with an M.Sc.in Computing Science.

Waltham measures its processes and provides feedback, through the Software Quality Engineering group, to improve future processes. The Software Quality Engineering is instrumental in assessing and providing feedback. In addition, it has the power to mandate changes.

12.2.3 Application Goals

The driving application goal was to develop the AIB. However, there were additional goals in that Waltham wanted to have experience in applying formal methods to a product development and Bristol wanted to develop a light weight, pragmatic approach to using rigorous techniques for specification, design and coding.

The development of the AIB was very low on the Waltham priority list but still needed to be finished. Since it had a low priority of staffing, the availability of HP Bristol staff provided leverage and reduced costs. A previous patient monitor (older hardware) had an AIB-like database, and Component Monitoring System clients were asking for that functionality to reappear as it had been lost because of changes in hardware.

Time to market, economics, reliability and quality are all deemed important for the AIB. With respect to the AIB, time to market concerns were mitigated by the release of regular up-dates to the Patient Monitoring System.

There were two novelties associated with the AIB development. Firstly, there was the use of formal methods on a production development. Secondly, Waltham perceived the Bristol group as an offshore development team. Such a perception is of note since HP has been considering going offshore (for example, to India and Eastern Europe) to develop its software.

12.2.4 Formal Methods Factors

Formal methods were chosen because of their potential benefits during the early phases of development. The Hewlett-Packard Specification Language (PH-SL) was chosen because in-house expertise was available. Effectively, the investment to learn a different formal methods technology would only be considered if HP-SL was shown to be clearly inadequate.

12.2.5 Formal Methods and Tools Usage

According to Paul Harry, the Hewlett-Packard Specification Language (HP-SL) was used at two stages

of the AIB project: specification and design. Quoting from his reply to the first questionnaire:

> Specification. The AIB is a data intensive system, well suited to a "classic" VDM approach of state plus operations. We therefore specified the entire AIB system at a relatively abstract level. The message interface to the AIB was also specified. The main requirements not formalised were performance ones -- as a result these were probably somewhat neglected.
>
> Design. Individual components of the design, in particular Abstract Data Types, were identified from the formal specification. These were then individually formally specified. To complete the design, the composition of the Abstract Data Types was informally defined.

The use of formal methods did change the procedures that the ST project team had been following. Firstly, the HP-SL resulted in a specification that provided a description of behaviour in far greater detail than usually provided. Secondly, the number of reviews increased, they were more rigorous, and the available information more detailed. As a consequence, Software Quality Engineering concluded that they should be tougher with their reviews on other products and that requirements analysis should occur earlier in the development cycle. The specification was also used, by Harry, to aid in unit testing. In short, HP-SL is the formal specification language incorporated in the Rigorous Engineering Process, which was developed and promoted at Bristol. The Rigorous Engineering Process incorporates abstract machines, reviews, and testing. Waltham suggested that, while the rigorous engineering process might be beneficial to them, mastering a formal specification language, such as HP-SL, requires substantial effort. It was also suggested that the notations are not popular with engineers in Waltham. An additional problem was the lack of reviewers. It was impossible to find domain experts who were also familiar with the ideas of formal specification.

For reasons of continuity, the AIB used an earlier version of HP-SL. Tool support for this version was limited to a syntax checker and pretty printer. A well-formedness checker has been implemented for a later version of the language. The HP-SL tool was separate from the configuration management toolset used in Waltham. Cross-referencing and Emacs-like jumping to declarations were provided by the HP-SL tool.

It was noted that, when the HP-SL tool included its own version of configuration management, it was painful to use. Once some of the complex features were removed, and it became less ambitious, it was a better tool.

12.2.6 Results

The AIB was developed in a controlled fashion, to high quality (zero defects), and with traceability of requirements to code. The only problems found during integration of the AIB were in some supporting code that proved to be useful and became part of the AIB software without going through the rigorous development process; there were no errors detected in the AIB code that was rigorously developed.

Despite the high quality of the AIB, Waltham does not intend to adopt the formal methods component of HP-SL. The Software Quality Engineering group has shown interest in using the "rigorous engineering process," as it forces earlier decisions, found more errors, and provided more systematic and detailed reviews.

On a corporate level, the Applied Methods Group collaborative projects have shown that formal specification techniques provide benefits, but the learning costs are very high, so the perceived value is quite low.

HP Labs is not a long-term technology transfer organisation, the Labs responsibility is to develop technology which has high impact on important products. The Applied Methods Group and Specification Technology Group projects demonstrated that formal specifications were applicable and usable, but, this did not create significant demand for the technology. This stage in the project has coincided with a change of organisation and focus within HP Labs in Bristol, and as a result a decision has been made not to continue with the technology transfer work.

12.2.7 Defibrillator

We include here a brief discussion of an earlier project for background purposes.

Prior to the work with the AIB, the Applied Methods Group undertook similar work with a number of other HP divisions. In particular, in conjunction with a product development group in McMinnville, Oregon, they developed a specification for part of a defibrillator. The defibrillator was called Prometheus.

It was noted that HP pioneered the first defibrillator that included microprocessor control. However, that was two generations ago and there is now substantially more software being included in such machines. Given the criticality of the product, there was a perceived need that formal methods matched the needs of the defibrillator development. To transfer the HP-SL technology to Oregon, a project-centred approach was used. This included a one-week course, followed by a workshop and intensive consulting.

It took three weeks to specify the defibrillator software, and the specification was placed under revision control. Ultimately, there was an in-depth review of the specification at Bristol.

It was of some interest to those who had worked in the defibrillator domain for a long time that the specification effort uncovered flaws. Initially, the manager of the project (Dan Jordon) was concerned about the amount of time being spent on the specification and not on, for example, writing code.

Once the HP-SL specification had been developed, it was found that writing the C code was easy. The coding phase was efficient taking about 6-8 weeks. It was also noted that test code started showing up earlier in the development process.

HP-SL led to the results that Oregon were looking for, especially on quality. They are striving for error-free software. It was also noted that the dominant issues on Prometheus were those of hardware, rather than software. Consistent with the AIB project, the defibrillator product prior to Prometheus also had error-free software.

Safety criticality was a driving force for the development of Prometheus; but this is not generally the case at HP. There were, and are, concerns at HP about possible regulatory activity by FDA on software.

12.3 Evaluation

12.3.1 Background

Product:

The AIB is a real-time database that maintains information on the patient being monitored.

Industrial Process:

Use of Rigorous Engineering Process and the Hewlett-Package Specification Language. Implementation offshore at Bristol. Acceptance testing at Waltham. System testing not yet initiated.

Scale:

55 pages of HP-SL, of which 39 pages specify database actions and 16 pages pertain to interface issues. 1290 lines of specification and design, and 4390 lines of delivered code (plus 4580 lines of support code).

Formal Methods Process:

Use of HP-SL to specify AIB requirements and to trace the requirements to code. Used abstract machine style of specification. Had syntax checker for HP-SL; but well-formedness checker not available for early version of language.

Major Results:

Successful implementation of the AIB using HP-SL as the specification language. High quality code delivered (zero defects at integration). High productivity during coding phase. Failed to transfer formal methods technology into Waltham; though some software engineering methodological ideas are expected to be adopted.

Interview Profile:

Half day interview with Bear and Harry in Bristol.
Half day interview with Ladeau and Connolly in Waltham.

12.3.2 Product Features

Client satisfaction	0	(1)
Cost	0	(2)
Impact of product	n/a	(3)
Quality	+	(4)
Time to market	0	(5)

(1) The client, in this case, the ST measurement group, indicated satisfaction with the system produced and the process by which it was produced. There was little deviation from initial requirements, the code does follow from the requirements and design documentation, and there is traceability of requirements. The rating for this category reflects the satisfaction with the delivered AIB software. However, the use of HP-SL was not perceived to have added enough value to offset the steep learning curve required.

(2) The cost of the development of the AIB software was considered to be a negligible part of the product development when compared against the total manufacturing cost of a predominantly hardware product. Hence, the software development cost had no impact. Note that, during the

collaboration, the majority of the time and personnel costs associated with using the formal method were borne by HP Bristol Labs and not the product division.

(3) The overall integrated Component Monitoring System, of which the AIB is one component, is but one of many such products sold by HP, and is not considered by HP to be a strategic product. In addition, the Componenet Monitoring System application which uses the AIB has not been released.

(4) In a report from Waltham to Bristol, it was noted that "Of special note is the fact that zero defects were found during integration and testing of the approx. 4500 lines of database code. This extremely high level of quality made integration and testing a very straightforward task."

(5) The Component Monitoring System is updated periodically and the AIB development has not had an effect on any new release dates.

12.3.3 Process Features

General process effects

Cost	0	(6)
Impact of process	+	(7)
Pedagogical	+	(8)
Tools	0	(9)

Specific process effects

Design	+	(10)
Reusable components	0	(11)
Maintainability	n/a	(12)
Requirements capture	+	(13)
V&V	+	(14)

(6) The quality of the AIB code provides added value, but it is difficult to quantify. It was felt that the cost of mastering a formal specification language, such as HP-SL, is too high to justify the learning costs.

It was also felt that costs could be reduced by using selected aspects of the rigorous development process (eg, many, very structured reviews at all development stages) without the use of HP-SL.

It was noted that the productivity achieved during the implementation and test phases was high, (in excess of 150 lines of defect free code per day).

(7) The impact of the process was positive since the Waltham Software Quality Engineering Group intends to import certain aspects of the Rigorous Engineering Process. For example, to require earlier decisions during the requirements phase, to improve traceability, and to reduce testing. However, stated Waltham intention is that the Hewlett-Packard Specification Language will not be a part of the proposed new procedures.

(8) Successful since both Bristol and Waltham learned from the experiment. Bristol successfully

increased their understanding of how to carry the formal methods down to the code level. Waltham wanted greater exposure to formal methods so as to ascertain their utility.

(9) Except for a syntax checker, no formal methods tools were used.

(10) Designing the AIB component was helped by the ability to model the behaviour of the system using HP-SL in a way that was not possible using approaches which the product group had used previously. Various requirements were found to be superfluous.

(11) There was no apparent positive or negative effect on reuse.

(12) Insufficient information, although zero defects at integration suggests that maintenance costs will be low.

(13) Understanding the requirements and expressing the required system behaviour was improved. It helped to communicate ideas between Bristol and Waltham and led to the removal of four unnecessary requirements.

(14) At the very least, the specifications helped with unit testing. Testing also used executable pre- and post-conditions. Formalism was a benefit in the review process by defining common vocabulary.

12.3.4 Observations

A main driving force for HP's development of a wide-spectrum of medical instruments is the perception, within the medical community, that many of these tools will result in reduced labour costs.

In the opinions of the authors, it appears that one of the problems with the attempted transition was the lack of follow-through after the presentation of courses. In addition, there was not an explicit corporate commitment to formal methods technology.

12.3.5 Formal Methods R & D Summary

12.3.5.1 Methods

An HP derivative of VDM, called HP-SL, was used and the specifications took the form of abstract machines. Real-time was not part of the specification.

12.3.5.2 Tools

An HP-SL syntax checker and a pretty-printer were used. In addition, the tool supported cross-referencing and emacs-like jumping to declarations. A well-formedness checker has now been implemented.

It was noted that, initially, when the tool had its own configuration management capabilities, it was painful to use. Now that the tool is less ambitious it is a better tool.

12.3.5.3 Recommendations to the FM Research Community

While most of the methodological ideas embedded in HP-SL seem to have been viewed positively by Waltham, it is clear that the notational aspects of a formal specification language, such as HP-SL, were an inhibitor. We were explicitly informed that no-one could be found in Waltham who could review HP-SL specifications. Notional ideas of HP-SL were also a problem. For example, the key idea of developing a model of a system prior to its construction does not map to the way most of the engineers in Waltham perceive how they work.

It appears, therefore, that the formal methods community needs to take a closer look at how to fit their technology with existing engineering practices. It is important to ensure that the benefits of formal methods are not outweighed by the perceived costs of learning how to use the technology in a mature way. There is also a warning on the tool front; do not let the tools become overly ambitious or the perspective users may be swamped with too much functionality. In many instances, the learning curve for formal methods is already steep enough.

12.4 Conclusions

The AIB was successfully developed and achieved quality goals at no increase in cost. In addition, the engineers at Waltham were exposed further to formal methods ideas and have decided that the insertion of HP-SL methodological concepts would be useful. However, the collaboration emphasized that the learning costs are very high, too high for spontaneous adoption within Waltham. This experience has been repeated elsewhere within the company and as a result HP Labs Bristol is not continuing with the technology transfer work.

It is of interest that in this area where domain expertise is high, the quality of the AIB was no better than that of released software products developed by Waltham. It appears that in cases where the software systems are not particularly large, the in-depth knowledge of the domain is sufficient to maintain high quality.

Appendix A Acknowledgements

Support for this study was provided by organizations in Canada and the United States. The Atomic Energy Control Board of Canada provided support for Dan Craigen and for the technical editing provided by Karen Summerskill. The U.S. Naval Research Laboratories, Washington, D.C., provided support for all three authors. The U.S. National Institute of Standards and Technology provided support for Ted Ralston.

The authors particularly wish to acknowledge the numerous individuals who agreed to participate in the study. Many of these individuals were sidetracked from their busy schedules, yet made the time available to fill out questionnaires and to be the subject of interviews. This study could not have been possible without their participation.

We also wish to thank our review committee, which consisted of Lorraine Duvall, John Gannon, Morven Gentleman, Adele Goldberg, John Marciniak; and the reviewers from our sponsoring organizations, namely, Richard Taylor of AECB, Dolores Wallace and Rick Kuhn of NIST, and Connie Heitmeyer, Anne Rose, Alan Bull, Carolyn Gasarch, and Daniel Kiskis of NRL.

The authors wish to thank Karen Summerskill, ORA Canada, for her work on editing and typesetting this document.

Dan Craigen
ORA Canada
Ottawa, Ontario, Canada

Susan Gerhart
Applied Formal Methods
Austin, Texas, U.S.A.

Ted Ralston
Ralston Research Associates
Tacoma, Washington, U.S.A.