Proceedings of **Z** Users Meeting
1 Wellington Square, Oxford

Jonathan Bowen

8th December, 1987

Programme Committee

John Nicholls
Jonathan Bowen
Jim Woodcock

Oxford University Computing Laboratory
Programming Research Group
8-11 Keble Road
Oxford OX1 3QD

Contents

1 Introduction

On 8th December 1987, the second annual **Z** Users Meeting was held at 1 Wellington Square, Oxford. John Nicholls, a Research Officer in the Programming Research Group (PRG) at Oxford University, chaired the meeting, taking over from Ib Sørensen (a lecturer at the PRG) who organized the meeting last year. 74 delegates from industry, academia and other research establishments were present representing a good cross-section of those who are interested in **Z**.

2 Annual Report

After an initial welcome, John Nicholls presented a summary of the work undertaken on **Z** at the PRG since the last meeting. The presentation was split up into a number of topics:

Standards. At last year's meeting, the general consensus was (at least from people outside the PRG!) that **Z** needed to be standardized to make it more acceptable and useful to industrial users and to enable tools to be produced. This has been attacked on two fronts at the PRG in the last year. A draft document entitled *Z: Grammar and Abstract and Concrete Syntaxes* by Steve King et al. has been produced, giving a BNF style description of a standard syntax. This was distributed at the meeting, and a final version is due to appear as a PRG monograph shortly. Additionally, Mike Spivey has produced a draft of a *Z Reference Manual*, a replacement for Bernard Sufrin's *Z Handbook*, of which more later. These documents could be merged. Additionally a less formal *Z User Manual* including more examples could be useful. Such a manual has been produced by IBM although the present version is not up to date with the latest **Z** syntax. Users are invited to express their views on the need for an informal manual.

Standards are important for stability. They should be clear and give a mathematical underpinning to the notation. If desirable, a Standards Group could be set up with the help of the PRG. The PRG will concentrate on basic theoretical research as well as joint work with industry to investigate the practical use of formal methods. Additional work is being done to provide guidelines for proof methods using **Z**.

Tools. It was suggested that these may be split into basic tools (e.g. producing **Z** documents on a PC) and intermediate/advanced tools (e.g.

FORSITE on a Sun workstation). More information is included on both these areas later.

Refinement & Concurrency. These are vital aspects of a general development method. Linking **Z** and CSP would be useful to cover concurrency. Talks on these two areas were presented later.

Communication. A number of methods of communication within the **Z** community were discussed.

Z forum. Ruaridh Macdonald of the Royal Signals and Radar Establishment (RSRE) at Malvern edits an electronic newsletter called *Z forum*. This is made up of short articles, requests and so forth contributed by readers. Anyone with an e-mail address on PSS, the academic network JANET, the UUCP network UKnet or many other networks connected to these may receive issues as they are produced and submit entries by contacting Ruaridh on `rm@uk.mod.rsre` or `rm%uk.mod.rsre@uk.ac.ucl.cs.nss` on JANET.

Z bibliography. A selected bibliography by Jonathan Bowen was distributed at the meeting. This document gives a current list of **Z** references available either from the librarian at the PRG or as published papers or technical reports from other institutions. A master list which also includes unpublished work has been compiled by the editor of the *Z forum*, Ruaridh Macdonald. The list is maintained in electronic form (in UNIX *refer* bibliography format). If you have any new references for this list or want a copy of it, please contact Ruaridh, via e-mail if possible at the same address given under **Z forum** above.

Posters. Posters describing research work, etc., were invited from the participants of the meeting. These were displayed at the meeting and are also attached to these Proceedings.

The idea of a **Z** Users Group was mooted for discussion (see section under **Future Work** later).

3 Work at the PRG

A number of presentations on work on **Z** at Oxford were given by members of the Programming Research Group.

3.1 Z Standards: Syntax, Ib Sørensen

A history of the evolution of **Z** in terms of key work was presented.

1979	The Specification Language Z (Abrial et al.)
	(ADA-like)
1980/81	The Basic Library (Abrial et al.)
	(Cliff Jones influence, VDM)
1982	A Theoretical Foundation to Formal Programming (Abrial)
	(Burstall/Scott influence, Set Theory)
1983	**Z** Handbook (Sufrin)
	Z Schema Notation (Morgan)
	Z Reference Card (Hayes)
	(Semantics — DPhil thesis — Spivey)
1986/87	Structuring Specifications — Schemas (Woodcock)
	Z: Grammar and Concrete and Abstract Syntax (King)
	Z Reference Manual (Spivey)

Throughout this period, case studies in the use of **Z** were also being undertaken.

The *Z: Grammar and Concrete and Abstract Syntax* was requested at last year's **Z** User Meeting. This work was supported by IBM. The contents include:

- Grammar in BNF.
- Standard terminology.
- Appearance and layout.
- Scope and type rules (informally).
- Abstract Grammar of **Z** (in **Z**).

Details of the grammar cover:

- Document structure.
- Language of *Definitions*.
- Language of *Theorems*. This is not yet stable since no extensive case studies have been undertaken yet. Theorems are currently not included in the *Z Reference Manual*.

- Language of *Predicates*. This has been stable since the early days of Abrial.
- Language of *Terms*. (Abrial's name — Mike Spivey uses the name *Expressions* in the *Z Reference Manual*.)
- Language of *Schemas*.

The concrete and abstract syntaxes are not split in this document, although they could be in a course on **Z**. Details of terminal symbols (e.g. `Z` to start a **Z** section and `EZ` to end a **Z** section) are also covered.

The target readers for this document include:

- Tool writers
- Educators
- Manual writers
- Expert users — not *naive* users

This has been used as a working document within the PRG. It is due for publication as a PRG monograph early in 1988.

3.2 Z Reference Manual, Mike Spivey

In starting, Mike noted that a LaTeX style file (`zed.sty`)[1] for printing **Z** specifications is available from him if you can contact him by electronic mail. His address is `mike@uk.ac.oxford.prg` on JANET.

Mike then continued his talk with a quotation:

> *Jack*: You're quite perfect, Miss Fairfax.
>
> *Gwendolen*: Oh! I hope I am not that. It would leave no room for developments, and I intend to develop in many directions.
>
> Oscar Wilde, *The Importance of Being Earnest*

If possible, a standard should leave room for future developments. The *Z Reference Manual* contains a minimal language for **Z** specifications, a subset of the language described in the *Z: Grammar and Concrete and Abstract Syntax* document. For example, theorems and the importing of one document into another are not included. However, this minimal language may

[1] This document was produced using this style file.

be added to, if necessary for a particular specification, but on the same mathematical basis.

At present the *Z Reference Manual* is in draft form and will be submitted for publication as a book; Mike proposed the following time-scale of events:

- December 1987. Draft available for review.
- 1st April, 1988. End of review period.
- 1st November, 1988. Proposed date for publication.

A copy will be sent to all participants of the **Z** User Meeting. People are invited to review the draft and send comments and reactions back to Mike at the PRG. The objective is to make the document understandable and readable, unlike many other standards. It uses examples sometimes, where appropriate. Algebraic laws are also included.

Mike then gave a short introduction to the difference between the *generic definition* and the *axiomatic description* in **Z**. A *generic definition* may have generic parameters and must make a unique definition:

X declaration
predicate

An *axiomatic description* has no generic parameters and may make a loose (i.e. a possibly non-unique) definition:

declaration
predicate

Looseness or liberality may be useful in the the following cases. An informal explanation of which is meant should also be included.

- To give freedom for the implementor. E.g., the coding of directories as data blocks.
- To allow details to be fixed later. E.g., the maximum number of users — a system option.
- When the details are irrelevant. E.g., the coding of characters as natural numbers.

The question may arise, "Why not have loose generic definitions?" Consider the following example:

$$X left, right : X$$
$$left \neq right$$

How should the following points be resolved?

- What about $left[\{0\}]$ and $right[\{0\}]$? They are both forced to be 0, despite the axiom $left \neq right$.
- What about $left[\varnothing[\mathbb{N}]]$? It must be a member of $\varnothing[\mathbb{N}]$, because of the declaration $left : X$; but $\varnothing[\mathbb{N}]$ has no members!
- Is $left[X]$ always the same?
- Is $left[\mathbb{Z}]$ the same as $left[\mathbb{N}]$?

There is a lot of information which is curiously hidden. Therefore it makes sense to insist that there is exactly one model.

3.3 Z and Concurrency, Jim Woodcock

Jim Woodcock, a Research Fellow at the PRG, presented some work on adding the facilities of CSP to **Z**, without changing the current features of **Z**. A number of rules for splitting operations in parallel are needed. Jim gave an example of a telephone exchange. First he described the system from a subscriber's point of view.

A *T*elephone may be either not ringing or ringing:

$$\begin{aligned} T \mathrel{\widehat=} & (lift \rightarrow S) \\ & \sqcap (lift \rightarrow R) \end{aligned}$$

where $\sqcap$ denotes non-deterministic *internal* choice (i.e. internal action in the system, outside the control of the subscriber).

Once a subscriber has lifted the receiver and *S*eized the line, he may then put the telephone down or dial a digit:

$$\begin{aligned} S \mathrel{\widehat=} & (clear \rightarrow T) \\ & \mid (dial \rightarrow D) \end{aligned}$$

where $\mid$ denotes deterministic *external* choice (i.e. action by the subscriber).

Whilst *D*ialing, a subscriber may put the telephone down or dial a digit. The system can detect when enough digits have been dialled:

$$\begin{aligned} D \mathrel{\widehat{=}} & (clear \rightarrow T) \\ & \mid (dial \rightarrow D) \\ & \sqcap (clear \rightarrow T) \end{aligned}$$

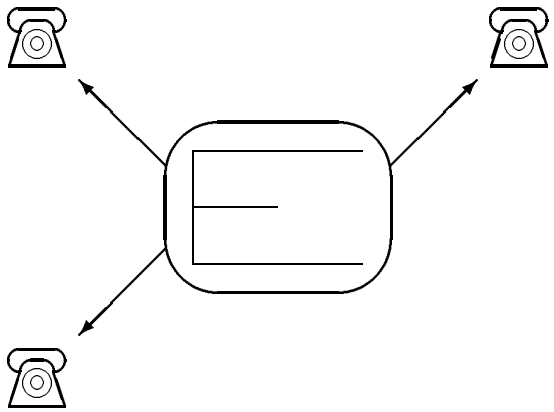
A *R*ecipient can hang up and then lift the receiver to reestablish the call. The other subscriber could hang up or the initiator could hang up.

$$\begin{aligned} R \mathrel{\widehat{=}} & (clear \rightarrow (lift \rightarrow T)) \\ & \qquad \sqcap T) \\ & \sqcap S \end{aligned}$$

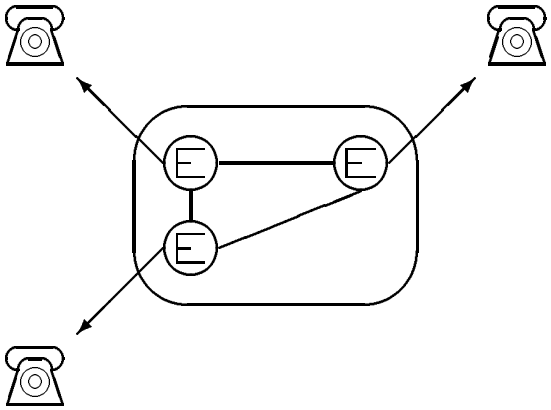
(Note that the description above has abstracted away from speech!)

An abstract view of the state of the telephone exchange can be provided as a standard **Z** schema, and operations on this state may be defined in the normal way by relating before and after states. In addition to the standard subscriber operations, there is a demon *Connect* operation which allows the system to make connections between subscribers.

Theorems about the state and the operations are easier to prove because there is a global state in the abstract description.



However in practice the exchange will be implemented as a number of distributed interconnected subsystems with a relay for each subscriber.



A retrieval relation between the abstract and concrete descriptions must be provided.

The non-determinism in the formal abstract description (i.e. all occurrences of $\sqcap$) may be removed by providing more information about the implementation of the telephone exchange. The implementation can be formally proved to be a refinement of the abstract description using formal rules provided by Jifeng He of the PRG.

The question and answer session elucidated a number of points:

- Transient states are not retrieved — only stable states are considered.
- Deadlocks and live-locks must be checked. Jifeng's rules ensure that these are avoided.
- Divergence must also be avoided — an infinite sequence of events should never be hidden.
- The *Connect* demon can connect two or more requests in an arbitrary order.

3.4 Z Refinement, Carroll Morgan

Carroll Morgan, a lecturer at the PRG, described a method of refinement from an abstract specification (e.g. something akin to **Z**) into a programming language. The notation used is not in the framework of the current **Z** notation.

What is needed is a single notation which at its most abstract allows the use of schemas[1] and at its most concrete allows executable programs[1].

[1] Almost

A calculus of *refinement* is introduced which operates across this notation uniformly.

At the abstract level, the *domain* of each operation is calculated and written explicitly, so that it need not be calculated again. The *changing variables* are indicated explicitly, so that unchanging variables need not be indicated. E.g.

EXAMPLE

$x, y, x', y' : T$

$\varphi(x, y, x')$
$y' = y$

translates to

$$x : [(\exists\, x' \cdot \varphi(x, y, x')), \varphi(x, y, x')]$$

The initial x indicates the changing variable(s). The rest of the statement indicates the precondition (or domain) and postcondition: $[pre, post]$.

At the concrete level, Dijkstra's language of *weakest preconditions* is used to describe executable programs. This is a very simple language:

Do nothing	`skip`
Do anything	`abort`
Assign	$x := E$
Sequence	$P;\ Q$
Conditional	`if` $G_1 \to S_1 \ [\!]\ \ldots\ [\!]\ G_n \to S_n$ `fi`
Iteration	`do` $G_1 \to S_1 \ [\!]\ \ldots\ [\!]\ G_n \to S_n$ `do`

An important feature of the language is that it is non-deterministic. If two or more guards on a conditional statement are true then any one of the corresponding guarded statements could be executed. However the language is readily translated into imperative programming languages such as Pascal, FORTRAN, etc.

The refinement calculus $\sqsubseteq$ has a number of rules or laws; the following are some examples:

- The precondition may be weakened:

$$w : [\varphi, \psi] \ \sqsubseteq\ w : [\chi, \psi] \qquad \text{if } \varphi \to \chi$$

- The postcondition may be strengthened:

$$w : [\varphi, \psi] \ \sqsubseteq\ w : [\varphi, \chi] \qquad \text{if } \chi \wedge \varphi \to \psi$$

- An assignment statement may be introduced:

$$w : [\varphi[E/w'], \psi] \ \sqsubseteq \ w := E$$

- Sequential composition ($;$) provides ordering of statements:

$$w : [\varphi, \psi] \ \sqsubseteq \ w : [\varphi, \chi'] \ ; \ \ w : [\chi, \psi]$$

  χ is an intermediate state. An interesting point is that this refinement is possible even if $\chi = \mathit{false}$.

- If at least one guard is *true* then the `if` statement may be introduced:

$$w : [(\bigvee i.G_i) \wedge \varphi, \psi] \ \sqsubseteq$$
`if`
$$(\ [\!] \ i. \ G_i \rightarrow w : [G_i \wedge \varphi, \psi] \ )$$
`fi`

- Finally, the `do` statement can be introduced for iteration:

$$w : [I, I \wedge \neg \ G] \ \sqsubseteq$$
`do`
$$G \rightarrow w : [I \wedge G, I \wedge 0 \leq V' < V]$$
`od`

  I is the invariant and V the variant.

Here is a simple example which calculates the absolute value of a natural number. Note that parts within [...] are the parts which are left the be refined.

ABS

$x, x' : \mathbb{N}$
$x' = \mid x \mid$

transforms to

$$x : [\mathit{true}, x' = \mid x \mid]$$

First the precondition is weakened:

$$\sqsubseteq \ x : [(x \leq 0) \vee (x \geq 0), x' = \mid x \mid]$$

Next the `if` rule is applied:

$$\begin{array}{ll} \sqsubseteq & \texttt{if} \;\; x \leq 0 \rightarrow x : [x \leq 0, x' =\mid x \mid] \\ & \;\; [\!] \;\; x \geq 0 \rightarrow x : [x \geq 0, x' =\mid x \mid] \\ & \texttt{fi} \end{array}$$

The preconditions can be transformed:

$$\begin{array}{ll} \sqsubseteq & \texttt{if} \;\; x \leq 0 \rightarrow x : [-x =\mid x \mid, x' =\mid x \mid] \\ & \;\; [\!] \;\; x \geq 0 \rightarrow x : [x =\mid x \mid, x' =\mid x \mid] \\ & \texttt{fi} \end{array}$$

Now the specification can be made totally executable:

$$\begin{array}{ll} \sqsubseteq & \texttt{if} \;\; x \leq 0 \rightarrow x := -x \\ & \;\; [\!] \;\; x \geq 0 \rightarrow \texttt{skip} \\ & \texttt{fi} \end{array}$$

It can also be made deterministic:

$$\begin{array}{ll} \sqsubseteq & \texttt{if} \;\; x \leq 0 \rightarrow x := -x \\ & \;\; [\!] \;\; x > 0 \rightarrow \texttt{skip} \\ & \texttt{fi} \end{array}$$

Informally, this can be refined into a typical imperative programming language as:

$$\texttt{if}\ x \leq 0\ \texttt{then}\ x := -x\ \texttt{end}$$

The refinement process above is not mechanical in that invariants, etc., must be found. The designer must decide which development step to take at each stage. However the refinement rules give a rigid and formal framework for this.

Next Carroll talked about *data refinement*, a transformation which systematically replaces one data type (the abstract) by another (the concrete) throughout a program. The program structure is preserved. Given abstract variables a and concrete variables c, we can define an abstraction invariant $I(a, c, g)$. The data refinement $\leq$ can then be calculated as follows:

$$a, x : [\varphi, \psi] \;\leq\; c, x[(\exists\, a \cdot I \wedge \varphi), (\exists\, a \cdot I \wedge \psi)]$$

This is always true — there are no proof obligations. However some subsequent algorithmic refinement is usually required. It is possible to have intermediate steps. The refinement is transitive.

John Nicholls concluded the session by noting that there will be a workshop on refinement at York University on 7th/8th January 1988 at which several members of the PRG (and others working with **Z**) will be giving presentations.

4 Prototype FORSITE Tool Demonstration

About 30 people attended a demonstration by Joy Reed and Jane Sinclair of the FORSITE prototype syntax and type checking tool running on a Sun workstation at the PRG after lunch.

5 Z Education

The possibility of a course on "*Z for Specifications*" based on the MSc. and industry courses at the PRG was discussed by John Nicholls. The objective is to produce an industry course to introduce the use of **Z** to specify software systems. It will be fully documented, with foils, exercises, course notes, etc. At present, Anthony Hall, John Nicholls, Joy Reed and Jim Woodcock are involved in designing and writing the material for this course. The following schedule has been drawn up:

- January 1988 — all modules specified.
- Mid 1988 — draft course material complete (pilot courses).
- Late 1988 — publication (format to be decided).

The (one week) course structure will be as follows:

A. Motivation.
B. Discrete mathematics (optional, background information).
C. Basic **Z**.
D. Managing projects using formal methods.
E. Extended case study.
F. Workshop (1–1$^1/_2$ days writing **Z**).
G. **Z** proofs.
H. **Z** refinement (introduction to another course).

The course material will be prepared for class room use rather than computer based training.

6 Z Tools and Environments

This session included two presentations on machine assistance for **Z**. These contrasted the use of **Z** in a sophisticated and a simple environment.

6.1 The FORSITE Project, Dave Bosomworth

Dave Bosomworth of Racal gave an overview of the FORSITE project. This is an Alvey project running from August 1985 to March 1989. Racal Research Ltd, System Designers, Surrey University and the PRG are involved in the project.

The background to the project was an Alvey Demonstrator Project involved in producing **Z** specifications over 6–9 months. This found the need for tools to aid organization and to do syntax/type checking. Names could be remembered in the head for only around two pages whilst reading specifications.

The project objectives were to produce a **Z** toolset and handbook. In the course of doing this, the syntax of **Z** should be tied down. To date, an evaluation system including a parser, type checker, WYSIWYG editor (QED) and printing facilities has been produced. This has users at four industrial sites (Plessey/Praxis/ICL/IST) and three academic sites (Edinburgh/York Universities and Sheffield Polytechnic).

The real requirements for **Z** tools is still a subject of debate. The project has not answered any hard theoretical questions so far (e.g. refinement and proofs). It is hoped that these will be addressed in the future, particularly in regard to large examples.

The plans for the project include work on proof systems (at present the **b** tool is used), transformation systems, development methods (there are no set steps at present) and better tools.

The FORSITE system runs on Sun workstations. It is mostly written in C, although the type checker is in the functional programming language ML.

6.2 Z on an IBM PC — the Issues, Jonathan Moffett

Jonathan Moffett of Imperial College recently attended a **Z** course given by the PRG. Following the course, he raised the question of providing support for producing **Z** documents on an IBM PC.

He noted that **Z** is text rather than graphics. Therefore a word processing package rather than a graphics package is all that is needed. The special (non-ASCII) characters in **Z** can be reproduced using special fonts. Additionally, the special characters can be keyed in on a normal keyboard.

Z is designed for large specifications. Therefore the special characters need to be stored in an easily handled form. Jonathan made the following (not yet implemented) proposals.

- The special characters should be keyed using mnemonics (see *Z: Grammar and Abstract and Concrete Syntaxes* by Steve King et al.).
- These mnemonics should be post processed using a macro processor. This processing is not necessary to be able to read the document whilst working on it. It is simply desirable to produce a better end product.

The **Z** character set may be split up into a number of groups on the IBM PC. There are the standard alphanumeric characters together with characters such as

$$(\;) \; [\;] \; \ldots etc.$$

which may be keyed directly on the IBM keyboard. There are standard letters but in different fonts. E.g.

$$\aleph \; \Re \; \ldots etc.$$

Note that these are Gothic rather than baroque fonts! Unfortunately, the more normal $\mathbb{N}$, $\mathbb{R}$, etc. are not available on the IBM PC. The extended graphics character set can be used for lines, boxes and $\vdash$. The mathematical font characters (for the Apple Laserwriter) cover the Greek letters and

$$\forall \; \exists \; \widehat{=} \; \neq \; \neg \; \vee \; \wedge \; \Rightarrow \ldots etc.$$

Special characters are needed for "funny" arrows and

$$\lhd \; \rhd \; \trianglelefteq \; \trianglerighteq \; \restriction \ldots etc.$$

Programmers are used to mnemonics. However John Nicholls noted that most people prefer seeing mathematical symbols where these are appropriate. They are easier to read and more compact. An alternative solution is to use PC LaTeX. Lexx, a live parsing editor, has a dynamic parser and could provide another solution. There are also word processing packages to run on the PC for as low as £20 which provide mathematical fonts. Jonathan Moffett's system cost around £700 including a printer. With an Epson laser printer, the cost would rise to about £2000.

7 Z User Presentations

A number of short presentations by **Z** users (mainly from industry) were given in this session. These provided varied examples of the way **Z** can be and has been applied to different problems.

7.1 British Aerospace, Brian Hepworth

Brian Hepworth of the Software Reliability group in the Military Aircraft Division at Warton presented the use of **Z** in control systems. Currently there are three users of **Z** in the company. Most users of **Z** are likely to be "naive" users (systems engineers) using "simple" **Z**. The difficult parts will be undertaken by a small group of experts.

Brian gave an example of a control system, which he noted was quite similar to Jim Woodcock's telephone exchange. Since the system is distributed, it can be easily split up for different teams. Previously CORE has been used at the upper level of design. It is hoped that **Z** will replace this. Many of the operations are stateless, with only inputs and outputs — the state is held externally. Sometimes it is necessary to introduce state to model history sequences. Functions specified by the **Z** experts can be defined to handle these simply. In the future it may be convenient to use some of the features of CSP for this.

7.2 IBM, Peter Collins

Peter Collins of IBM, Hursley Park, started by saying he was going to tell a "story of unstable (i.e. changing) software rather than unstable aircraft!" He presented the experience of using **Z** on the IBM Customer Information Control System (CICS). This is a large transaction processing system which is an extension of the operating system. It has been developed at Hursley and has many thousands of users worldwide. It consists of well in excess of half a million lines of code, written in a mixture of assembler and high level language. It has had many years of continuous development using a well established development process.

The use of **Z** at Hursley started as a joint project between the PRG and the CICS development team in 1981. The objective was to study the applicability of formal specification techniques to the development of CICS. Initially case studies where undertaken using the **Z** specification language.

In 1984 it was decided to use **Z** in the CICS, first in pilot projects and then for mainline development. The alternative possibility (CDL) was dropped and the complete development process has been based on **Z**. Twenty developers have written about 2000 pages of **Z** specifications. Subsequently,

around 90,000 lines of code have been written. Coding is now complete and testing is well underway. This has shown that the quality is significantly higher than when using traditional development methods. About a third of a release of CICS has been produced using formal techniques.

Formal methods have been used for the specification and design of selected CICS components. These provide both new functions and reimplementation of existing functions. Standard **Z** has been used for component specification. An extended version of **Z** has been used for component design. This includes Dijkstra's guarded command language to allow the specification of procedural design. Its mathematical basis permits formal reasoning.

The development process has been changed to accommodate the use of formal methods. Design Review 0 (**DR0**) has been introduced to review the **Z** specifications. At this stage the interface is reviewed by potential users. Quite a few problems are removed at this point. The **I0** and **I1** design inspections have been modified to take account of **Z**. **I0** is an informal check that the design is a correct refinement of the **DR0** specification. There is no formal proof of refinement. This is very expensive without machine assistance and is not believed to be cost effective at present.

The work at the PRG has been mainly based on case studies of real examples from CICS. These have exposed new kinds of problems. They have demonstrated the feasibility of using formal methods to the potential users and their management. They also provide examples for others to follow.

There is a user resistance to the introduction of formal techniques. Using real examples is more impressive than the simple examples found in much of the literature. Even so, some management faith is required. The specification stage is longer and the coding stage much shorter when using formal techniques. It is possible for managers to lose their nerve and revert to traditional methods.

The case studies were followed by pilot projects. These involved small motivated groups, giving rise to local experts at IBM. As a result, rules and guidelines were generated for other users. Additionally, these projects helped identify other requirements.

A number of courses have been developed at IBM. A two week Software Engineering Workshop uses SEDL rather than **Z**. However there are now a number of **Z** courses provided by John Wordsworth and others of the Software Engineering group at Hursley:

- *Z Specification course* — originally provided by the PRG but revised by IBM.

- *Z Refinement course* — the refinement of **Z** specifications to design and code.
- *Z for Readers course* — understanding other people's specifications.
- *Z Overview* — brief overview of **Z** for managers.

A number of tools for **Z** have been developed at IBM Hursley. The initial tools were developed by the CICS developers. **Z** documents may be prepared and viewed on an IBM 3279 display. Special **Z** characters are entered using mnemonics and are then translated. Subsequently documents may be printed using a Document Composition Facility on an IBM 3800. The system consists of terminals and a mainframe computer. It would be expensive to convert to IBM PCs. There is also a powerful cross-reference program which is very useful for large specifications. Despite the limited machine assistance, there have not been a lot of complaints about the lack of more sophisticated tools.

It is important in a commercial environment to have a stable notation. Researchers naturally wish to improve and change the notation, so there is a potential conflict of interests. No agreed standard for **Z** existed at the start of the project; it is a good thing that the PRG is now standardizing **Z**. As part of the project a precise and stable base language has been defined. This includes a formal syntax to be used as the basis for tools such as type-checkers. A *Z User Manual* has also been produced. This gives an informal guide to the notation with examples of use. There is a need for a manual such as this. Both Mike Spivey's manual (the *Z Reference Manual*) and one like this are useful and complement each other. Another important consideration is a standard for representing **Z** documents; standards become more important as the number of **Z** users increases.

Users at IBM have been impressed by the simplicity and elegance of **Z**. Users especially like the ability to compose large specifications from small parts. Better interfaces have resulted and the specification of existing interfaces has showed up their weaknesses. **Z** has proved to be a tool for ideas. Better informal documentation has resulted. **Z** specification has been used in testing — pre/post-conditions and invariants can be checked. However this can use more code than is actually being checked. The formal mathematical nature of **Z** has not proved a major problem in acceptance by programmers. Indications are that more problems are found during early stages. Documentation is more complete and easier to inspect. The overall quality of software and documentation is much improved.

One minor problem has been that some users find it difficult to find the right level of abstraction. If a programmer knows the internal workings of a

system, he often finds it difficult not to think about these when producing an abstract specification.

7.3 Plessey, David Cooper

David Cooper of the Formal Methods Group at Plessey Research, Roke Manor gave a presentation of a short paper entitled *Some Notes on the Use of Z*. This three page paper is included with this Proceedings.

The paper presents some difficulties they experienced in the use of **Z** for large specifications but David is still committed to **Z**. Perhaps part of the problem may be that the team has been trying to come to terms with the use of **Z** whilst working on a very large project. It is possible that a more gentle introduction to the practical use of **Z** may be desirable. More contact with the PRG could also help to solve some of these problems. Several instruction sets have been specified at the PRG without many of the problems covered in the paper. (The Motorola 6800 8-bit microprocessor instruction set has been completely specified and is available as a PRG monograph. Partial specifications of the Motorola 16-bit 68000 and the INMOS Transputer have also been produced.)

The project has relied heavily on the FORSITE tool. This ensures a strictly type-checked **Z** specification but also enforces notational limitations since the tool does not support all the features of **Z** and will not allow non-standard notational extensions. The IBM CICS project has demonstrated that it is not necessary to have such a tool to successfully produce large specifications.

7.4 RSRE, Chris Sennett

Chris Sennett of the Royal Signals and Radar Establishment (RSRE) at Malvern is concerned with security problems in computer systems. He is involved in the Hindsight Project which is working on a syntax and type checker for **Z** at RSRE.

The main part of the talk was about the translation of the mathematical content of a paper on *separability* by John Rushby of Newcastle University into the **Z** notation. This process simplified formulae and aided understanding. The proof steps were laid out so that they were potentially machine checkable. Chris commented that it could be desirable to have a number of equivalent definitions in the **Z** mathematical toolkit. Different definitions could be aimed at ease of understanding on the one hand and ease of use in proofs on the other.

In summary, the use of the **Z** language help specify precisely what *separability* is about and increased understanding of the problem. Using **Z** and English in parallel is very useful — the **Z** can aid the understanding of the English! The **Z** specification produced during the exercise was very compact. The original paper was 17 pages long; the version translated into **Z** by Chris consisted of 25 pages, despite the fact that the details of several proofs were given in the **Z** version which were not given (or were given in much less detail) in the original.

8 Research Directions

Professor Tony Hoare of the PRG presented his view of research at the PRG in the future. He started his talk with an entertaining analogy of a geologist and a car driver. The geologist (researcher) searches for oil (knowledge) and the car driver (industrial user) needs petrol (methods). Even if a petrol pump (**Z** user group?) is available, this must still be supplied with petrol.

What is basic research? It is as far removed from industry as geology is from the car — but the latter still depends on the former. Basic research is like mountaineering; we do it "because it's there." Basic research clarifies the structure of a disipline, and permits collaboration of specialists working in adjacent areas. Some mathematical research foundations are:

- Logic
- Algebra
- Type theory
- Category theory

Some computing paradigms for which these can or could be useful include:

- Hardware
- Functional programming
- *Prolog*ical programming
- Object-oriented programming
- Distributed systems

Sometimes it is useful to mix paradigms (e.g. assembler within a high level program). How can we deal with this? How do we check interfaces between paradigms, where many bugs can lurk?

Currently some machine assistance for proofs is available. E.g.

- Boyer-Moore — 15 years work.
- LCF, Edinburgh — again, 15 years work.
- HOL, Cambridge.
- Veritas.
- LF, Edinburgh.

Each of these presents formidable problems and expense to the practical user. Basic research may reveal new directions for progress. There are also a number of methods other than **Z**:

- UNITY — Chandy and Misra at Austin.
- VDM — an acceptable alternative!
- NuPRL — Martin-Löf type theory.
- Projet Formel — 2nd order logic and category theory.

It would be a worthwhile research project to prove a complete system at a number of levels using such tools and methods:

- Compiler/loader
- Operating system
- Architecture
- Logic design

Human readable proofs would be acceptable. The interfaces between levels should also be proved. Computational Logic Inc., in Austin, Texas, are attempting to do something like this with 12 people and Symbolics machines. We should also try to do this in Europe as an ESPRIT project. There is a lot of skill in this area in Europe, but it needs coordinating.

A short discussion then followed. Question: "Are reusable libraries a suitable topic for basic research?" The term *reusable* has been a buzz-word for

25 years. Such libraries must be as carefully designed as a range of cars and hence need a great deal of effort to produce. Standard functions such as `sin` and `cos` are already well known and understood. What equivalents do we need for computer systems? **Z** has isolated a number of common concepts and is sufficiently abstract to describe similarities between systems. But suppose we had a reusable library. How would we use it, index it, and so on? This would have to be done by a very large company with a lot of resources.

Question: "How do we deal with probability and is this a good research topic?" Yes — Mike Reed at the PRG is working on a probabilistic version of CSP. However timed CSP took a long time to develop and a probabilistic version may take even longer.

9 Future Work

John Nicholls asked the audience if they would be interested in the formation of a **Z** User Group. This would place **Z** in a wider arena. John Wordsworth of IBM suggested that it might be formed as a specialist group under the auspices of the British Computer Society (BCS). Interest was expressed by the audience. Anyone interested should contact John Nicholls at the PRG.

John closed the meeting by thanking the speakers, especially those during the **Z** user presentations which reflected the wide applicability of **Z**. Finally he thanked Emma Sowton of the PRG for arranging the administrative side of the meeting so efficiently.

10 Meeting at the PRG

On December 9th, the day after the **Z** Users Meeting, another much smaller meeting was held in the Conference Room at the PRG in Keble Road to discuss a number of topics related to the future of **Z**. Those present were Jonathan Bowen, John Nicholls, Joy Reed and Mike Spivey of the PRG, Brian Hepworth and Bruce Manley of British Aerospace, and Anthony Hall of Praxis. The conclusions of this meeting are set out below.

10.1 Tools

Basic tools (editors, formatters, etc.) are needed first. For wide use in industry and universities, it is important for tools to be able to accept a standard form of text which can be stored in ASCII or EBCDIC. This *Z In-*

terchange Form will be defined as a standard, and will incorporate standard **Z** mnemonics (see **Standards** subsection below).

In principle, it should be possible to develop intermediate tools, such as type checkers, etc., fairly rapidly, since much of the necessary work has been done in building the FORSITE prototype.

10.2 Education

The course development plan outlined at the **Z** Users Meeting was discussed and several suggestions were made. Details of the plans for this work will be circulated in the **Z** Forum electronic newsletter.

There is concern that in some courses **Z** is being presented as if it were a programming language. One way of avoiding this will be to show how **Z** fits into a typical development process.

10.3 Standards

It was agreed that the mnemonics in the document *Z: Grammar and Abstract and Concrete Syntaxes* by Steve King et al. should be used by the **Z** community when mnemonics are required by tools.

11 Acknowledgements

Thank you to all the speakers for making the 1987 **Z** Users Meeting a worthwhile day for all the participants. A special thank you to all those who gave copies of their foils and notes to aid the production of this record. Please accept the author's apologies for any mistakes in the transcription from verbal to written form. John Nicholls and other speakers from the PRG gave valuable comments on the first draft of the Proceedings. A copy has also been circulated on the **Z** Forum electronic newsletter and a number of minor corrections and additions have been made since then arising from comments by readers.

Finally, thank you to the audience for making the meeting a success, and hopefully a continuing annual event.