Experience Teaching Z with Tool and Web Support

Jonathan P. Bowen

South Bank University, SCISM, Borough Road, London SE1 0AA, UK
Email: jonathan.bowen@sbu.ac.uk
URL: http://www.jpbowen.com/

17 July 2000

Abstract. This short paper describes experiences of presenting the formal Z notation on one and later two course modules to computer science undergraduates, especially with respect to providing supporting web-based resources and using tool support. The modules were part of a more general course unit on formal methods.

1 Introduction

The underpinning of computer science with mathematics is important to ensure that theory and practice are not separated [20]. The teaching of formal methods in particular has become of increasing interest [7], partly because of the difficulties involved, especially in teaching the subject to poorer students [8]. The United Kingdom has traditionally had many courses on the formal Z notation within its computer science undergraduate degrees [17]. Ideally the use of formal methods, including model-based approaches as Z is normally used, should be integrated into software engineering courses. Certainly an overview of Z is included in a number of software engineering textbooks (e.g., see [22]). In some cases, formal methods (including Z) has been well integrated into software engineering programmes [9].

Experiences teaching the formal Z notation within a more general course unit on formal methods are given over the four year period of 1996 to 1999. The Z component of the course was an early adopter of the provision of web-based supporting material. It also expanded from a purely "paper and pencil" based approach to a tool-based approach during the course of its development. The unit was delivered to students of widely varying mathematical ability which can be problematic. It is difficult to keep the better students interested and not to leave the weaker students behind.

Until recently, formal methods has been taught as a core subject at the Part II level of the Computer Science undergraduate degree course at the University of Reading in the UK. Part II constitutes terms 3 to 5 of the nine term (three year) course due to the non-standard structuring of degree courses at Reading. Students were taught three different aspects of formal methods in each of the termly modules, one of formal specification (using the Z notation), one on refinement which leads reasonably naturally from the Z module, and one on analysis of algorithms, especially with respect to efficiency, that was somewhat less well integrated. For a "unit description" briefly describing these three aspects, see an edited version of the description issued to students in Figure 1.

2/CS/3T – Formal Methods

Omissions and inaccuracies in a requirements definition can have serious repercussions in later stages of system development. Specifications written in natural language, accompanied by semi-formal diagrams and illustrations, may be ambiguous or even inconsistent. Mathematical techniques, or formal methods, can help us to reason about requirements and to detect inconsistencies, investigate the consequences of design decisions, and test the suitability of an implementation, before building any part of the system. These techniques are applicable throughout computer science, from requirements definition to hardware design and are a complement to current design practice.

The course introduces methods for specifying software systems, refining and transforming specifications into abstract programs and analyzing the efficiency of the final implementation.

The specification part of this course begins with a review of the mathematical basis of formal methods, which is followed by an introduction to Z, a widely-used formal notation for describing sequential systems.

The refinement part of the course examines the mathematical aspects of program correctness and deals with the rigorous development of imperative programs. It introduces a theory of program refinement, and shows how it is possible to proceed from a specification to executable code without introducing new, possibly undesirable, aspects of behaviour.

The analysis part of the course introduces mathematical methods for designing algorithms and estimating their efficiencies on sequential machines. The emphasis is on applying useful programming paradigms and sound design principles in the construction of efficient algorithms.
Terms: Summer, Autumn and Lent

Pre-requisites: 1/CS/C, 1/CS/D, 1/MA/3

Lecturers: A.E. Abdallah, J.P. Bowen, P.N. Nissanke

Teaching and Learning Methods: The course is primarily lecture-based, and is supported by weekly tutorials/practicals.

Assessment Coursework: There are two components to the coursework:

1. Reading and writing specifications;
2. Refining specifications formally to code.

Weight: 25%

Examination: One three-hour written paper
Weight: 75%

Fig. 1. Unit description.

During the period 1996 to 1999, student numbers for undergraduate computer science courses have increased substantially, partly due to central government pressures and partly due to demand because of the high expectations for employment at the end of such a course. Many computer science undergraduate courses have insisted on A-level mathematics or equivalent in the past in the UK. However, this is increasingly less so with the larger numbers of students. Thus there is a widening range of mathematical ability in students pursuing computer science degrees. This is problematic in course units depending on mathematical abilities, such as those involving formal methods.

Because of this situation, and the increasing "tail" of students attaining poor marks for the formal methods unit, it was decided to concentrate on two rather than three aspects of formal methods. This reduces the number of notations that a student encounters and allows more in-depth study. The analysis aspect of the unit was removed and the specification aspect increased to incorporate experience of tool-based support for formatting, type-checking and animation of a Z specification. Students did significantly better with the experience of producing a more realistic Z specification, both in the assessment of the work itself and in the subsequent examination.

The prerequisites for the course unit were mathematical units in Part I (terms 1 to 2). This included a unit specifically on discrete mathematics (logic and set theory). Two versions were available. One was given by the Mathematics Department, and was more suitable for the abler students with a mathematics A-level. The other was given by the Computer Science Department, tailored to students with less mathematical ability. This was a self-paced course unit with no formal lectures, but was instead based around exercises with supervised sessions. The material for this course unit has recently been published as a book [18]. The notation used in this book largely adheres to the Z style of discrete mathematics, so there is a natural transition to a Z course unit. The associated unit was successful in raising the level of poorer students. However, a problem that was not resolved for a while was that it was possible for students to concentrate on logic or set theory to the detriment of the other and still pass the unit. This many students entered the formal methods unit without the full mathematical grounding ideally required for such a unit.

2 Teaching

The formal methods course unit was lecture-based (two hour-long lectures per week), with the support of weekly tutorials in the first term and (latterly) weekly practical sessions in the second term. The lectures introduced the following broad topics:

- Industrial use of formal methods [12];
- Mathematical notation (logic, sets, relations, functions, sequences, ...);
- Schema notation for structuring;
- Case studies;
- Z type-checking / animation.

Refinement towards a program was covered in a separate (one term) module of the same (three term) unit.

In the first term of the unit (Term 3 of the degree course), the students were introduced to the most widely used parts of the mathematical and schema (for structuring) notations of Z. The notation for logic, sets, relations, functions and sequences was built up using a series of simple examples, normally developed on the board rather than being presented on slides. Writing a formal specification is far more difficult that reading a specification but it is important for students aiming to be software engineers to be able to do both. The examples were chosen to be relevant to other aspects of the degree course; e.g., a (simplified version of) the World Wide Web was used as a running example. The process of producing a formal specification [4, 5] is as important as the final specification itself [10], so a major aspect of the unit was to show how a specification can be developed, in the framework of the Z notation.

Four exercises, supported by tutorials, were used to help develop student's understanding of Z and their specification skills:

1. Logic and Quantifiers / Sets and Set Comprehension.
2. Relations and Functions / Numbers and Sequences.
3. Normalization and Schemas.
4. Z Specification.

An aspect emphasized in the unit was the normalization of expressions to their most basic types using just given sets, Cartesian products ("$\times$") and power sets ("$\mathbb{P}$"). Understanding this process greatly aids the use of a Z typing-checking tool (needed in the second term). This is especially helpful in the analysis of error messages produced by such type-checkers that can sometimes be confusing without a basic appreciation of the Z type-checking mechanism.

In the original structure of the unit with one term allocated to Z, students were given an assessed exercise where they were required to develop a simple Z specification by answering a number of directed questions. The final question was optional, aimed at the better students, and required the use of and reasoning about the composition of two operation schemas. Most undertook this work using paper and pencil, although a few managed to use a word processor, despite the difficulty of formatting Z schemas and some of the Z operators.

After the Z component of the unit was extended to two terms, the original written assessed work was dropped. Instead, at the end of the first term, an assessed test was held during one of the lecture slots. This provided useful and fast feedback on the progress and ability of the students at a stage where it was still possible to do something about it if required. The tutors of students who failed the test were informed so that they could discuss the situation with the students concerned. In the second term (Term 4 of the degree course), remedial tutorials were provided for those who did poorly in the assessed test. Feedback from the students indicated that this was a helpful aspect of the course unit and certainly results improved at the bottom end of the scale after its introduction.

Lectures in the second term concentrated on larger more realistic examples, emphasizing the process of developing a Z specification and the typical structure of such a specification when Z is used in a model-based style with an abstract state, as is commonly the case in practical use. This style comes more easily to engineers than some of the even more abstract algebraic approaches where no state is modelled explicitly. Second term lectures also provided support for the tools to used in the practical sessions for the development of a Z specification by the students themselves.

The practicals in the second term involved producing a Z specification starting with a template for some of the initial schemas. The suggested specifications included an invoicing case study [2, 4] and a system allowing files to be printed. Students had the option of developing a formal specification of there own for some system they knew well, but few students attempted this. Figure 2 shows some of the general guidance given to students.

3 Recommended books

Various books were recommended on the unit. No one book was followed exactly. It was recommended that a Z textbook be obtained by students and used to complement the course unit with additional reading outside lectures. The following books were recommended:

Formal Specification and Documentation Using Z (Bowen) [1]:
This book (by the lecturer) is more suitable for advanced students and is probably better for an MSc course. It provides a brief introduction to formal methods in general and Z in particular, followed by a number of mainly real-life case studies.

The Way of Z (Jacky) [13]:
This was especially recommended for the last year of delivery of this course unit when references to specific chapters to be read in conjunction with the lectures were given. It is written by a practitioner and a very clear down-to-earth style making it very approachable and convincing for many students. Uniquely, it has actually been recommended to me by students. It is suitable for less able students as well as more able ones since the book is split into short chapters ranging from basic concepts to more advanced features of Z such as the technique of promotion for larger specifications.

An Introduction to Formal Specification and Z (Potter, Sinclair & Till) [19]:
This book, in its second edition, was recommended as an alternative book for students to consider if they found the other recommended books unsatisfactory. It was not explicitly followed in the delivery of the unit.

Using Z: Specification, Refinement, and Proof (Woodcock & Davies) [26]:
This is an excellent book for advanced students, but not so good for the less mathematically able. Its scope is also wider than just formal specification using Z since it also covers proof in the context of Z and refinement from a Z specification towards a program.

The Z Notation: A Reference Manual (Spivey) [24]:
This excellent reference book, aimed at the serious Z specifier, has been a de facto standard for Z since its original publication in 1989, with a second edition in 1992. It was not recommended for purchase by students, but was recommended for consultation by more advanced students using copies available in the main library.

Formal Methods (2/CS/3T) – Z notation: Practical assessment

The assessment will involve writing a formal specification using the Z notation and then applying the ZTC type-checker. Once type-checked, you should attempt to use the ZANS animator on your specification. You should either use the example in Exercise 4 as a starting point or specify any system of your own choice. In the latter case, it is recommended that you choose a system that you already know well. The specification should consist of the following as a minimum:

1. Definition of basic types.
2. Optional: axiomatic and/or generic definitions for use in the subsequent specification.
3. An abstract state schema.
4. An initial state.
5. Change of state schema if required.
6. At least three operations where success is assumed. Success and error schemas providing suitable reports.
7. Definitions of total operations.
8. Optional: One or more validation hypotheses or theorems with proofs.

You are expected to do the following during the course of the term:

1. Produce a Z specification along the lines above, written in LaTeX source form, and formatted using the LaTeX document preparation system. Your formal specification should be accompanied by an informal explanation, of approximately the same size as the formal part of the description.
2. Successfully type-check the entire specification using the ZTC type-checker.
3. Attempt to animate part (at least one operation) of the specification using the ZANS animator. Note: make a separate copy of your specification for this since you will probably need to modify it to make animation possible. Any problems should be noted in your report.

You may discuss your specification with the demonstrator during practical sessions, or with the lecturer after the Monday lectures, as you develop it. A report on your practical work during the term should be handed in at the end of the assessment period, covering the following:

1. Introduction to the problem which you decide to tackle.
2. Presentation of the Z specification, including matching informal English explanation.
3. Discussion of problems encountered in the development your specification, particularly with respect to ensuring its type correctness using ZTC.
4. Presentation of animation runs using ZANS for one or more operations in your specification.
5. Conclusions, including general problems encountered and any changes of approach you would use if you were to undertake the exercise again.

You may use the above items as the basis for the sections in your report. Feedback on your assessed work will be provided during the practical sessions and if required during lectures as well. You are expected to discuss your assessment as it progresses, as if you are a member of a design team. It is highly recommended that you write up your report in draft form as you proceed, and polish it at the end, rather than leaving it all till the end. Extra marks can be gained for the following:

– Elegance of specification and approach.
– Clarity and organization of your report.
– Simplicity (if correct), rather than verbosity.

Guidance on the above can be given at the practicals. In particular, you should have your Z specification checked by the demonstrator and successfully type-checked by ZTC before proceeding to attempt to animate it.

Fig. 2. Practical assessment.

4 Tool support

Students used the following tools in the development of their Z specification for the assessed work in the second term:

LaTeX [15, 16]: This is a document preparation system widely used in academia and with good support for mathematics in general and the Z notation in particular. The `oz` Z style file [14] was used on the course unit for the formatting of Z within a LaTeX document [25]. LaTeX is not a WSIWYG system, so students were able to use any standard ASCII text editor with which they felt comfortable. Most used the `textedit` editor available on Sun workstations.

LaTeX uses a macro-based markup system to control formatting. To avoid students having to learn too much about LaTeX, a template file was made available that provided a basic document structure with some initial incomplete Z schemas as examples. One lecture was allocated to teaching minimal general LaTeX markup and some specific Z markup. This proved sufficient and there were few problems in practice.

LaTeX is freely available on-line [16]; an excellent free Windows version is also available [21].

ZTC [30, 31]: This is Z type-checker that accepts LaTeX documents as input. The error messages can sometimes be rather obscure, but an understanding of the normalization of Z expressions taught as part of the course unit helps in interpreting the messages. In general, only the first error message can be trusted since it is easy for an error to have severe subsequent knock-on effects. ZTC can produce a nicely formatted text listing of the types found during its checking that can be a useful aid. An advantage of ZTC (especially in an academic environment) is that it is free. A disadvantage is that it is not quite as reliable as some commercial Z type-checking tools such as *f*UZZ [23] but works well with most normal Z specifications. Thus for didactic purposes it is more than adequate and causes few problems. Another advantage is that the ZTC manual is very comprehensive and is available in electronic form (POSTSCRIPT format). This was issued to students in printed form for convenience. The appendix giving the Z markup is an especially useful reference section.

ZANS [27–29]: This is a prototype Z animator designed for use with and produced by the same person as ZTC. As a prototype system it is less robust than ZTC, but is still a useful tool. In particular, probably its most helpful feature is not animation but the ability to determine if an operation schema is "explicit" or "implicit."

An explicit operation determines all its after-state components and outputs with postconditions of the form $x' = \ldots$ and $y! = \ldots$. Such explicitly deterministic operation schemas can often be animated relatively easily. An implicit schema has at least one after-state component or output that is not explicitly determined. Thus the operation is non-deterministic and it is more difficult to animate it effectively. The specification may be perfectly reasonable if the non-determinism is a deliberate feature of the formalization where certain aspects have been loosely specified and left for the implementor to determine. However, in the case of novice specifiers

such as students, the reason is, far more often than not, that a predicate for a state component or output have been omitted. Often in the case of after-state components, what is actually required is that the state component keeps its value before the operation (e.g., $x' = x$). Thus ZANS is a useful aid for checking and correcting specifications even if no actual animation is done.
ZANS comes with a helpful tutorial [28] that was issued to students in printed form. This was presented in a lecture and it was recommended that students follow the tutorial before attempting animation of their own specification. The tutorial includes a specification with a deliberate error that can be detected through animation of the various error cases for one of the operations.
Often specifications need to be changed to be more deterministic to allow animation. Students were told that this was allowable, but that the original specification is what was required in the write-up of the practical, with an explanation of what needed changing if necessary.

Practicals were undertaken on Sun workstations, although PC versions of ZTC and ZANS were available for any students who wished to use these tools on their own PCs. Although students were less experienced with Sun workstations than PCs, this did not cause any significant problems in practice.

Student understanding of producing a Z specification increased significantly after the practical sessions. Many seemed to appreciate using tools far more than just pencil and paper. However, a danger with the use of an animator is the possibility of confusion between a (possibly non-executable) formal specification and an executable program [11]. A few always seem to stubbornly fail to recognize the difference even after this is emphasized repeatedly in lectures.

5 Web support

The course unit was supported by on-line web resources that were developed and improved each time the unit was delivered. These are still available[1] under:

`http://www.cs.reading.ac.uk/cs/people/jpb/teaching/z.html`

This resource includes the following:

– Overview of topics covered on the course unit.
– Recommended books with annotations,
– A two-page *Z Glossary* in POSTSCRIPT format, together with the matching LaTeX source.
– Lecture foils (in POSTSCRIPT format):
  - Industrial use of formal methods;
  - Introduction to Z – part I (mathematical notation);
  - Introduction to Z – part II (schemas);

[1] Please bear in mine that this resource is no longer maintained by the author due to lack of access since moving to South Bank University, so some external links may not work.

  - Case study – A file storage service;
  - Case study – A text formatting tool (“`pos`”).

  These are based on chapters of the Z textbook by the lecturer [1] and are freely available on the web for educational purposes, especially for those following the book.
- Four exercises (in POSTSCRIPT format) as previously mentioned. Note that model answers are also available, but are not on-line. These were given to students in printed form once they had attempted the questions.
- Assessment information, especially concerning the practicals.
- Links to external information concerning Z, including the WWW Virtual Library Z page (maintained by the lecturer).

Note that students were given much of this material in printed form. Otherwise having on-line material encourages students to simply print it out anyway. This is more expensive than mass photocopying and clogs the departments printers when they could be used for more productive purposes, such as printing out individual student’s personal work. However, having the material on-line is useful in those cases were the students lose their printed notes since it is readily accessible if needed quickly (e.g., near deadlines!). It also makes the material easily and accessible and updatable by the lecturer. This is especially helpful in revising the material each year for a course unit delivered and developed over a number of years.

6 Tips

Here we give some examples of issues in writing simple Z specifications that are often not very explicitly covered in Z textbooks. However, when students first start writing Z specifications, it is helpful to give a rather rigid process to doing this to ensure that parts of the specification are not omitted.

Z can be an overwhelming notation for students to learn if it is presented as a mass of notational detail in the form of the many different operators available in Z. Instead it is better to present a little notation and then an example that demonstrates why and how that notation is useful. It is helpful to introduce the basic schema notation early for structuring the examples and then to build on this as the course unit progresses. For most undergraduate courses, it is best to stick to a reasonably sized subset of the notation in any case. Some features of Z are much more widely used and are more useful than others. Much of Z is based on standard mathematical notation but some additional operators that have been found to be useful in the specification of computer-based systems have been added.

When an operator is introduced, as well as explaining its formal meaning, it is helpful to say when and how it is typically used in specifications. Z schema operations, at least in simple specifications, often divide into four basic styles:

- Adding a small amount of state to the full abstract state of the system;
- Deleting some state from the system;
- Updating part of the state (which can be considered as a combination of deleting some state and then adding some new state to replace it in some way);

- Status operations where the abstract state does not change during the operation but some part of the state is returned to the user as an output.

Given these four styles of operation, various Z operators turn out to be most useful in each situation.

- Set union ("$\cup$") is useful for adding to a set (or relation, a special sort of set in Z). For example, the predicate $x' = x \cup \{i?\}$ adds the input element $i?$ to the (state component) set x before an operation and constrains this to be the after-state x'. It may be desirable to ensure that $i?$ is not already in the set x since if it is the operation has no effect on x (i.e., $x' = x$). A *precondition* $i? \notin x$ could be added to ensure this if required.
- Set difference ("$\setminus$") is useful for deleting from a set. For example, the predicate $x' = x \setminus \{i?\}$ removes the input element $i?$ from the set x before an operation and again constrains this to be the after-state x'. It may be desirable to ensure that $i?$ *is* in the set x this time since otherwise the operation does not change x. In this case, a precondition of $i? \in x$ could be added to check this.
  When removing "maplet" elements from a (binary) relation (or function), pairs of (sub-)elements are involved. Often it is easiest to identify this using just using one of these elements, typically although not necessarily the one in the domain. In this case Z's domain anti-restriction operator ("$\ntriangleleft$") is useful. A predicate such as $r' = \{i?\} \ntriangleleft r$ removes any elements in r when the first of the pair is $i?$. A precondition of $i? \in \mathrm{dom}\, r$ would check that the input $i?$ is in the domain of r if desired. A similar range anti-restriction operator ("$\ntriangleright$") is also available. Unlike the standard set operators such as union and difference, these relational anti-restriction operations are rather more unique to Z and are not used in mathematics in general.
- For updating relations and functions, the overriding operator ("$\oplus$") is helpful. The predicate $r' = r \oplus \{i1? \mapsto i2?\}$ removes any maplets with input $i1?$ as the first element and replaces these with a new maplet $i1? \mapsto i2?$, equivalent to the pair $(i1?, i2?)$. Again overriding is an operator that is fairly unique to Z, but has been found to be very useful in practice. It can be found in many real Z specifications. In fact it is normally defined in terms of moving part of a relation and then adding to it: $r1 \oplus r2 = (\mathrm{dom}\, r2 \ntriangleleft r1) \cup r2$.

Z schema boxes have two areas, one for the declarations and one for any extra constraining predicates (defaulting to $true$ or no constraints if it is not present). In practice, the predicate part of an operation schema specifying the change of abstract state is normally conceptually split into two parts, the precondition (involving before-state components only) and the postcondition (specifying how the after-state relates to the before-state).

When considering an operation, an abstract state may be change from a before-state (packaged in a schema named $State$ for example) and a matching state schema named $State'$ where all the components have the same name but with the $'$ prime added. An unconstrained change of state consisting of both the before and after-state would be denoted as $\Delta State$ but convention and is useful as a starting point for operations that change the state in some way (e.g., add to, delete from or update it). $\Xi State$ is conventionally use to indicate the all the matching state components remain the same (e.g., $x' = x$) and it useful in the case of status operations.

In an operation schema, a precondition of $true$ means that the operation is "total" and can be applied in any situation. If a precondition is required, it often involves checking if a particular element (typically an input) is or not in some set forming part of the abstract state, using the $\in$ or $\notin$ operators, as illustrated above. In a precondition of the form $x \in S$ it is a good idea to ensure that S is not the empty set ($S \neq \varnothing$) since if it is, this reduces to $false$, something to be avoided in specification because this predicate can never be satisfied by any implementation of the operation.

Postconditions are often explicitly formulated in the form $x' = \ldots x \ldots$ for each after-state component (and output) in simple Z specifications. If a state component is not to change, this must be explicitly stated (e.g., $x' = x$). Often such predicates are omitted from Z specification by novices and the ZANS animation tool is excellent for detecting this. An implicitly specified operation is not necessarily wrong of course; in fact this is how non-deterministic specifications are normally achieved in Z. However, a student finding an implicit operation using ZANS can check whether a non-deterministic specification really is required; if it is, extra informal explanation can be added as to why it is non-deterministic.

It can be helpful to explain that $x' = f(x)$, some function of x, is the same as $x' \in \{f(x)\}$. This is a special form of $x' \in \{\ldots\}$ with a deterministic singleton set. In the same vein, $x' \in \{\}$ (the empty set with no elements in it), or equivalently $x' \in \varnothing$, is something to be avoided as a postcondition since it is the same as $false$, resulting in a specification that cannot be implemented. The predicate $true$ is an almost equally useless postcondition in an operation since, with no constraints, any implementation will do.

Typically all the before and after-state components, inputs and outputs are going to be mentioned in the predicate part of an operation schema one way or another, possibly hidden through schema inclusion of some form. If they are not, the student should check why this is so, and explain this especially carefully in the associated informal documentation.

There are further issues involving the initial state, error schemas, combining schemas to produces total operations with a precondition of $true$, etc. In each case, a process to help in determining what is required can reduce the likelihood of errors in formulating a simple specification, and thus aid in increasing the understanding of students in producing a basic Z specification that does not include errors that are obvious to any experienced Z specifier.

7 Conclusion

In the experience of the author, using tools in supporting formal methods course units helps the students in their understanding and increases their appreciation of the usefulness (or at least decreases their negativity) of formal methods.

By insisting that students type-check their Z specifications (using the ZTC tool on the course unit described here) and check for explicitness (or otherwise) at least using the related ZANS animation tool, many errors in Z specifications can be discovered and eliminated by the students themselves, sometimes with no help from demonstrators in the case of bright students. This allows demonstrators (and markers) to concentrate on the more interesting and difficult aspects of formulating a Z specification that require human inspection.

Web support for formal methods and other course units is a useful adjunct. A benefit is the accessibility of material by students, the staff involved, other colleagues, internal and external examiners, etc. It also helps in the maintenance of course unit material as a unit develops since this can be easily added and information can quickly be corrected in the case of errors. However, it is recommended that all essential material is still given to students in paper form, even if it is available on the web, since students will tend to print this anyway, which is still relatively expensive compared to photocopying. Of course the web resource can contain considerable extra supplementary material if desired at very little cost once it is installed.

The use of tools and web support are helpful for formal methods courses. However, a major problem in the UK at least is increasing students numbers and decreasing mathematical ability of the extra students. This can result in a very wide range of mathematical backgrounds and ability on a course unit such as that described here. This makes teaching the course difficult since good students may become bored whereas poor students can easily fall behind. In the latter case, it is particularly hard to catch up since much of the later material on this Z course unit depends on previously taught material.

With larger numbers of less able students on computer science degree courses, it is likely that formal methods will increasingly move out of mainstream core course units into more specialist optional units. With the increasing popularity of four year MEng courses for brighter students in the UK, this is the type of computer science course, where the author believes that formal methods should remain a core subject. Such students may end up programming the most critical applications [6], and should be educated to the highest level possible to help ensure their competence as future software engineers [3].

References

1. J. P. Bowen. *Formal Specification and Documentation Using Z: A Case Study Approach*. International Thomson Computer Press, 1996.
2. J. P. Bowen. An invoicing case study in Z. In M. Allemand, C. Attiogbé, and H. Habrias, editors, *Comparing Systems Specification Techniques*, pages 461–471, IRIN, 2 Rue de la Houssinière – 44322, Nantes Cedex 3, France, 26–27 March 1998.
3. J. P. Bowen. The ethics of safety-critical systems. *Communications of the ACM*, 43(4):91–97, April 2000.
4. J. P. Bowen. Z: A formal specification notation. In M. Frappier and H. Habrias, editors, *Software Specification Methods: An Overview Using a Case Study*. Springer-Verlag, 2000. To appear.
5. J. P. Bowen and M. G. Hinchey. Formal models and the specification process. In A. B. Tucker, Jr., editor, *The Computer Science and Engineering Handbook*, chapter 107, pages 2302–2322. CRC Press, 1997. Section X, Software Engineering.
6. J. P. Bowen and M. G. Hinchey, editors. *High-Integrity System Specification and Design*. FACIT series. Springer-Verlag, 1999.

7. C. N. Dean and M. G. Hinchey, editors. *Teaching and Learning Formal Methods*. Academic Press, 1996.
8. K. Finney. Mathematical notation in formal specification: Too difficult for the masses? *IEEE Transactions on Software Engineering*, 22(2):158–159, February 1996.
9. D. Garlan. Integrating formal methods into a professional master of software engineering program. In J. P. Bowen and J. A. Hall, editors, *Z User Workshop, Cambridge 1994*, Workshops in Computing, pages 71–85. Springer-Verlag, 1994.
10. J. A. Hall. Seven myths of formal methods. *IEEE Software*, 7(5):11–19, September 1990.
11. I. J. Hayes and C. B. Jones. Specifications are not (necessarily) executable. *IEE/BCS Software Engineering Journal*, 4(6):330–338, November 1989.
12. M. G. Hinchey and J. P. Bowen, editors. *Industrial-Strength Formal Methods in Practice*. FACIT series. Springer-Verlag, 1999.
13. J. Jacky. *The Way of Z: Practical Programming with Formal Methods*. Cambridge University Press, 1997.
14. P. King. Printing Z and Object-Z LaTeX documents. Department of Computer Science, University of Queensland, St. Lucia 4072, Australia, May 1990.
15. L. Lamport. *LaTeX User's Guide & Reference Manual: A document preparation system*. Addison-Wesley Publishing Company, 2nd edition, 1993.
16. LaTeX3 Project. LaTeX: A document preparation system. URL: *http://www.latex-project.org/*, 8 November 1999.
17. J. E. Nicholls. A survey of Z courses in the UK. In J. E. Nicholls, editor, *Z User Workshop, Oxford 1990*, Workshops in Computing, pages 343–350. Springer-Verlag, 1991.
18. N. Nissanke. *Introductory Logic and Sets for Computer Scientists*. Addison Wesley Longman, 1999.
19. B. F. Potter, J. E. Sinclair, and D. Till. *An Introduction to Formal Specification and Z*. Prentice Hall International Series in Computer Science, 2nd edition, 1996.
20. H. Saiedian. Mathematics of computing. *Journal of Computer Science Education*, 3(3):203–221, 1992.
21. C. Schenk. MiKTeX: Free TeX for Windows users. URL: *http://www.miktex.org/*, 8 July 2000.
22. I. Sommerville. Model-based specification. In *Software Engineering*, chapter 9, pages 189–205. Addison-Wesley Publishing Company, 5th edition, 1995.
23. J. M. Spivey. *The ƒUZZ Manual*. Computing Science Consultancy, 34 Westlands Grove, Stockton Lane, York YO3 0EF, UK, 2nd edition, July 1992.
24. J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall International Series in Computer Science, 2nd edition, 1992.
25. SVRC. Object-Z LaTeX macros. URL: *http://svrc.it.uq.edu.au/Object-Z/pages/latex.html*, 25 February 1998. The University of Queensland, Australia.
26. J. C. P. Woodcock and J. Davies. *Using Z: Specification, Refinement, and Proof*. Prentice Hall International Series in Computer Science, 1996.
27. Xiaoping Jia. An approach to animating Z specifications. In *Proc. 19th Annual IEEE International Computer Software and Applications Conference (COMPSAC'95)*, pages 108–113, Dallas, Texas, USA, August 1995.
28. Xiaoping Jia. A tutorial of ZANS – a Z animation tool. Release 0.31, July 1998.
29. Xiaoping Jia. Z animations. URL: *http://sotiris.cs.depaul.edu/fm/zans.html*, 25 February 1998. DePaul University, USA.
30. Xiaoping Jia. Z type checker. URL: *http://sotiris.cs.depaul.edu/fm/ztc.html*, 12 August 1998. DePaul University, USA.
31. Xiaoping Jia. *ZTC: A Type Checker for Z Notation – User's Guide*. DePaul University, Institute for Software Engineering, Department of Computer Science and Information Systems, Chicago, Illinois, USA, August 1998. Version 2.03.