Book Reviews

L. M. G. Feijs & H. B. M. Jonkers
Formal Specification and Design. Cambridge University Press, 1992, £29.95, 335pp hardbound, ISBN 0 521 43457 2

Gradually 'Formal Methods' are being accepted as a 'good thing' but as yet they have not made a significant impact on the production of software. In principle the use of formal methods means that 'correct' programs can be shown to be correct (with respect to an agreed specification), or alternatively that programs can be derived in such a way that they are guaranteed to be consistent with a given formal specification. For a program/system to be correct all its necessary parts also have to be correct. Ideally an initial specification should only indicate what is to be done, not how it is to be achieved (i.e. it should not suggest a particular design) and hence is, of necessity, abstract. Consequently, the gap between an initial specification and a final implementation is conceptually BIG. This is often cited as the reason why formal methods have not been adopted as readily as some of us would have hoped.

Various attempts have been made to reduce this gap and thus make it easier to bridge. The use of VHLL's is one approach that has been tried. Another, is to require that the specifications are written in such a way as to make the extraction of a correct design (for an implementation) relatively automatic. This latter approach can be criticised—it injects a considerable 'how' component into what the purists argue should be a 'what' specification. Nevertheless, any piece of research that helps to 'get it right' more often should be welcomed as should any book which makes this work accessible to a wider audience. (Although it is nowhere clearly stated, I presume that the target readership is not confined to the computer science research community who, via journal publications, technical reports and conference papers, have already had sight of much of this material—and, indeed, more up-to-date variants—albeit in a more condensed and less discursive form.)

Essentially the book is about the language COLD. *Common Object-oriented Language for Design.* a language developed, as part of an ESPRIT project called METEOR, by Philips in Eindhoven. The language incorporates both the algebraic and state-based styles of specification as well as facilities to assist in the description of designs. But beware, the algebraic specifications, though superficially similar to those of OBJ etc., do not give rise to *initial* algebras. Instead, they require inductive or algorithmic definitions and axioms which disallow certain interpretations rather than rely on (implicit) minimality. Similarly, the state-based specifications do not look at all like Z or VDM, but include axioms that indicate the required properties of the operations being specified.

The book is appropriately organised in three parts; the first being devoted to algebraic specifications, the second to state-based specifications, and the third to more advanced features of COLD and associated theory. It is based on courses given to graduate students in Holland and assumes some familiarity with the mathematical notations and terminology used in formal computer science. This may 'throw' the ill-informed reader; this is a specialist book, but very readable by anyone familiar with the subject area and who wishes to know about COLD.

Some aspects could be explained more succinctly—BNF could be usefully employed in several places—and elsewhere there is an unfortunate overloading of notation (here *p*! does not mean *p* factorial, or that *p* is unique, or that *p* is an output variable), but these are minor criticims and do not detract from the flow of the presentation. The style of the COLD definitions and axioms provides a nice link with computability theory and, swimming against the tide, reinforces the case for retaining the more theoretical aspects of computing in our courses. A graduate computer scientist, or someone of an equivalent professional standing, who is involved with software production should find this book both readable and interesting.

D. J. Cooke
Loughborough

Steve Scrivener
Computer-Supported Cooperative Work. Ashgate Publishing. 1994. ISBN 0-291-39812 X. £45.00. 286 pp. hardbound.

Recently, a number of computer products and services have been announced that claim to offer support for collaborative and group work. These range from systems for structuring and coordinating formal work processes through to video facilities for enhancing informal communication between personnel. The design and evaluation of such technology has been the focus of much academic research in the last few years, generally collected together under the term Computer-Supported Cooperative Work (CSCW). With the imminent possibility of deploying collaborative technologies within organizations, it appears timely to bring together the relevant experiences of CSCW researchers and report these to a more general audience. This was the intention of a seminar held in London in 1992. Following from this meeting, Steve Scrivener has compiled a varied and lively collection of contributions on CSCW.

CSCW is a very broad topic that includes research on work practices by sociologists and anthropologists, new computer architectures and network applications by computer scientists, and teamwork by social psychologists. Therefore, drawing together a general collection of papers on the subject presents considerable difficulties