deterministic choice between x and y, subsequently *skip.x* + *skip.y* is used for that (as might be expected from other notations). The inexperienced reader is left with the wrong impression for at least four pages and, if he believed the earlier explanation, possibly for the rest of the book.]

Chapter 3 contains PSF descriptions of three simple protocols:

- An alternating-bit protocol in which the channels cannot lose or invent data but can, like a one-place buffer, either transmit each datum exactly or corrupt it (in which case that fact is recorded with the datum).
- A positive-acknowledgement-with-retransmission protocol in which channels can now lose data, so a timer is required to avoid deadlock; only a single acknowledgement value is used.
- A concurrent alternating-bit protocol whose channels can lose data and in which transmission and receipt are loosely coupled.

Chapter 4 contains PSF descriptions of three sliding-window protocols from Tanenbaum's *Computer Networks*:

- A sliding-window protocol with both sending and receiving windows of unit size.
- A sliding-window protocol with sending window of arbitrary size but receiving window of unit size.
- A sliding-window protocol with arbitrary window sizes.

Chapter 5 contains a PSF description of the Amoeba Transaction Protocol, developed by Mullender in his 1985 PhD thesis in Amsterdam. It is designed for quick interactions between a client and a server and incorporates features that make it efficient when bursts of transactions occur. It is of interest partly because it does not conform to the ISO model.

Chapter 6 contains PSF descriptions of two simple protocols for local-area networks which are chosen to be simple but representative:

- A simple token-ring protocol (unidirectional with a single token).
- A simple ethernet protocol (with carrier sensing and collision detection).

Chapter 7 contains PSF description of the IEEE token-ring protocol, with time abstracted (since PSF abstracts real time). It makes no reference to the previous chapter, but is included as an example of how a realistic case study can be coded in PSF.

Finally appendices contain a library of PSF data modules (based on the types introduced in Chapter 2) and the syntax of PSF.

Each chapter starts with a clear introduction and ends with very brief summary and bibliographical notes.

This book, then, contains a collection of programs with brief explanations of the systems they simulate. At whom is such a collection aimed? Not at the expert in communications protocols; nor the novice, for whom little discussion of design properties and trade-offs appears. Not at the reader interested in the application of formal methods to communications protocols. For such a reader would presumably be interested in: the structuring of complex protocols in terms of simpler ones; correctness of the designs; the stepwise development of a protocol from its specification; and in a discussion of modelling, assumptions and alternative approaches. None of those appears in this book.

The fact that the alternating-bit protocol forms a buffer, for example, is just subtle enough to require proof. Here is one way to proceed. Define the simplest flow-control protocol with a single value for acknowledgement; it clearly forms a one-place buffer. Now refine that so the single acknowledgement value is replaced by natural numbers (initially 0 then incremented on each successive acknowledgement); by the preceding design, that too forms a one-place buffer. Again refine, this time by reducing the index modulo 2, to gain the alternating-bit protocol; again a simple argument yields correctness from that of the preceding design. Each stage in that hierarchical development isolates a feature of the final design. Yet because the concerns are separated each may be verified, and understood, independently and simply.

Such conceptual expression of complex systems in terms of simpler ones is not used at all in this book, though in almost every case some such consideration is possible. Thus the most powerful weapon in the armoury of formal methods, the control of abstraction, is ignored.

It thus appears that the authors have traded all for executability of their descriptions. The outcome of executing a PSF program is a trace, or history, of the system being simulated. It must be left for the reader to decide how helpful is a listing of traces of a complex (usually non-deterministic) system in understanding its behaviour.

J. W. Sanders
University of Oxford

Derrick Morris and Boris Tamm (EDs)
Concise Encyclopaedia of Software Engineering. Pergamon Press. 1993, £150, xiv+400 pp. hardbound, ISBN 0 08 036214 1

There is a popular myth that encyclopaedias exist so that people can look things up. In fact their prime purpose is to provide browsing fodder for ruminants like me. The *Concise Encyclopaedia of Software Engineering* consists of almost a hundred articles, averaging four pages apiece. It certainly passed the browsing test—I spent several happy hours dipping in here and there, reading sequences of unrelated entries, following the 'See also ...' pointers to related articles, making occasional serendipitous discoveries and, from time to time, feeling irked at omissions or misrepresentations of pet topics.

What about the other tests? Is it concise? Is it encyclopaedic? Is it software engineering? It is certainly concise—for an encyclopaedia. The articles are generally pithy, well organized and well written. They are reasonably consistent in style. Given the large number of contributors (almost 80), credit is due to the editors, Derrick Morris and Boris Tamm, for bringing them all into line.

It is also encyclopaedic—for a concise work. Sometimes I wish the articles were not so concise. For instance the five pages on *Graphics* conjure up a picture of the late 1970s and early 1980s; there is no hint of ray tracing, which has been in routine use for decades by the US Army or even solid modelling, which is a necessary complement to three-dimensional graphics. If graphics is a proper subject why are animation and virtual reality not treated? However, perhaps a better question to ask is whether graphics is a proper subject for discussion in an encyclopaedia on software engineering. Is it because the editors wanted at least one entry under the letter G? Hardly, because there is nothing under B, J, Q (Query Languages feature under *Databases*; perhaps there is a place here for Queuing Theory?), U (Unix makes an appearance in several articles), W (no Windows?), X (no XWindows?), Y and Z (Abrial's Z notation is treated in the article on *Specification and Verification of Software*).

No, if G is to have an article, it should be on Graphical User Interfaces (GUIs): despite their disadvantages (expense, in space and time, and inadequate standards), GUIs and GUI builders play a substantial part in the development of big software systems—their use in prototyping and building user interfaces is important because the interface, being the visible part of a software system, is often the deciding factor in whether or not the system is acceptable. At least GUI makes it into the list of acronyms at the back of the book; HCI (Human–Computer Interaction) and MMI (Man–Machine Interface) do not.

The article on *Graphics* and the lack of an article on GUIs exemplify one of my concerns: the encyclopaedia looks dated. Yes, I know encyclopaedias should record eternal truths rather than fashionable ephemera, but ... . Cynics would say that software engineering has precious few of the former but plenty of the latter. Each article ends with a bibliography which, rightly, includes early seminal publications but often gives the impression that nothing much has happened for many years. Perhaps it hasn't. Or perhaps the book was a long time in the making.

The last of my tests asked whether the subject was indeed software engineering. Many of the articles clearly are: *Software Engineering Environments*, *Lifecycles*, *Software Project Management*, *Specification and Verification of Software*, *Prototyping*, *Requirements Capture*, *Portability of Software* and many others. Some are on the borders of the subject or are concerned with specific application areas that present unusual or unusually difficult software engineering problems: the main example is real-time software which merits several articles. Some are, to my mind, out of place: I have already mentioned *Graphics* and there are others including *Code Generation* and *Systolic Algorithms for VLSI*.

Each reader will have some pet topics that are missing; one of mine is GUIs. Another is the costing of big software projects: estimating costs and performing cost/benefit analyses are notoriously difficult. A little is said about the former in the article on *Software Project Management* but I could find nothing on the latter; an article on each seems sensible as these are central to whether a project goes ahead. The article on *Design Methodologies* makes very brief mention of SSADM, Jackson's System Development (JSD) and several other design methods; again, these are central to a software engineer's life and each, while not perfect, at least contributes some useful ideas and thus merits rather more space in this work. JSD also receives an honourable mention in *Specification Languages* and a less than honourable one in *Maintenance of Software*. In the latter it is dismissed as contributing little to the problem of software maintenance—this is curious as ease of maintenance is one of the principal features of Jackson's method.

Despite this carping, the reader will have noticed that, by and large, the encyclopaedia has passed my four tests. The price, though, is horrific.

Perhaps the most illuminating article is Cronhjort's on *Validation and Verification of Software*—after giving a good overview of the problem of how to do software engineering it shows that we still don't know the answer.

R. M. McKEAG
The Queen's University of Belfast

ALISON CAWSEY
Explanation and Interaction: The Computer Generation of Explanatory Dialogues. The MIT Press. 1993, £24.95, 232 pp. hardbound, ISBN 0 262 03202 3

Explanation and Interaction presents a system for analysing and creating human–human and human–computer dialogues. The dialogues discussed are those of a student–expert type, with the user able to ask questions, prompt further explanations and ask for more detail in the areas and the expert responding with appropriate levels of information. The system used throughout the book is called EDGE.

The book begins by exploring the various possibilities involved in human–human interactions and attempts to describe them in a standardized form that can be applied to modelling them on a computer. The chosen domain is that of an expert explaining 'something' to a novice. In the examples in the book these are 'simple' electronic circuits involving transducers, resistors, etc. The fact that these circuits appear so simple belies the complexity of designing a conversational system to explain them.