examples. The base of the formalism is a composition of data models with timed and coloured Petri nets, which are well known as modelling tools today. So, one should not be afraid of the subtitle. The author states: 'The chosen combination of formalisms is useful'. I personally share his opinion. Moreover there is not any better reference for such a statement, than to use an operational tool based on this formalism. Unfortunately, this tool is not described and a demonstration version is not attached, so it cannot be a subject of this review.

We find in this book models of transport systems, models of assembling processes and many others. A coherent and compact presentation makes this book a good academic manual. Its value is due to the many exercises attached to the end of each part. A glossary, enclosed at the end of this book, helps with the first reading and self-study. This glossary includes interpretation of new concepts.

The educational advantages make the proposed formalism accessible not only for students, but also for engineers who want to intensify their knowledge about information systems engineering and understand the theoretical background of new system modelling methods. Although it is a difficult book, I recommend it to this sort of reader.

R. SOBCZAK
TU, Gdansk, Poland

KATHY SPURR, PAUL LAYZELL, LESLIE JENNISON AND NEIL RICHARDS (eds)
Business Objects: Software Solutions. John Wiley. 1994. ISBN 0-471-95187-0. £29.95. 233 pp. hardbound.

Object-oriented (OO) programming is characterized by (i) multi-function routines that retain state data between invocations; (ii) all data being the state data of such routines, so that an 'object' is effectively a (perhaps complex) variable, along with such a routine, often globally accessible; and (iii) asymmetry of CALLs, which are sent, as 'messages', to one of their parameter objects. Other parameters are passed as parameters proper and may themselves then be 'sent a message' by the invoked 'method', i.e. the function named in the original message and being 'the responsibility of' the receiving object. The origins of the approach lie in simulation and it serves very well for the simulation of (actual or conceivable) hardware devices, e.g. to implement the virtual printers used in spooling systems, or windows and other graphical interface components. For general purpose programming, however, it presents serious problems: processing two complex objects together is peculiarly difficult; automatic optimisation is almost impossible because the CALL has to be implemented 'in' the object to which the message is sent (no matter where it might be on a distributed network—in effect, OO systems are data access dependent); and, as we would expect of a simulation or of any system with a lot of global data and processing, verification after any change is an exceedingly complex task.

While the programmers struggle with these (probably insuperable) problems of their latest fad, its principles are being applied at earlier stages in the software development lifecycle: witness this latest (fifth) publication in the series of BCS CASE group seminar proceedings, whose price, incidentally, seems to be inflating at about 10% per annum. If you are looking for a much-needed, hard, critical—indeed radical—appraisal of OO analysis: look elsewhere.

The selling point of OO systems is *reuse*, although we still spot, here and there, the shadows of dubious claims of improved quality and productivity. In an interesting paper on Hewlett Packard's 'Fusion' method, Howard Ricketts recommends 'defer[ring] the assignment of responsibilities', i.e. enforcing asymmetry on CALLs, 'as late as possible'. (Perhaps *sine die* would be best.) Fusion also delays 'establishing inheritance', i.e. copying methods from one sort of object to another, 'until well into design'. This is excellent: let us always postpone complexity. But it is hardly OO.

Ricketts' paper is one of four discussing methods and tools: all informative, but—sorry to be so sceptical—do we want OO analysis at all? In the section on Architectures for Reuse, Stuart Frost tells us that '[a] rapid convergence on object technology is occurring . . . We will not be able to take advantage of the latest computing technology . . . unless our software uses an [OO] architecture'. It may not be that bad: Tim Boreham's paper, 'Re-use in OO Analysis', is quite an elegant exercise in data analysis; but it could be rephrased entirely in terms of old-fashioned entity analysis. Perhaps we could get by if we treated 'OO' merely as the latest general adjective of approbation. Only such a semantic shift could justify the wild optimism of the two papers on 'Managing the Transition'.

The editors have, wisely, decided to include some explanatory front matter. Unfortunately, the tutorial on OO analysis is error-ridden (even ignoring the macabre J Smith of 10 Downing Street); and the preface and introduction are ill-considered. Is it true that the 'approaches [that] separated data from procedure suffered from an inability to deal with . . . change'? What then of data independence? Is 'one order processing system . . . much like another'? Even if the one is for pre-invoicing and the other for post-invoicing? Is analysis merely 'the process of obtaining and clarifying our understanding of the problem'? Whither the functional specification of our solution? Is '[t]he concept of an Object . . . intuitive'? If so, why did we not intuit it long ago? (Actually it is very like Chen's concept of an entity, and almost as limpid.)

Amazingly, the editors compare the OO approach with the old 'craft culture', which they say is 'reaching

the end of its days'. But presumably it is still a good enough beast to be beaten with the very latest stick. The dear old craft culture surely now works only in reviewing. We just pick up the books on the latest fashion, shake our hoary heads and utter the words our nannies taught us: you're getting over-excited, there'll be tears before the day's out.

ADRIAN LARNER
de Montfort University

WILHELM SCHÄFER, RUBÉN PRIETO-DÍAZ AND MASAO MATSUMOTO
Software Reusability. Ellis Horwood. 1994. ISBN 0-13-063918-4. £29.95. 160 pp. hardbound.

This is the book, but not exactly the proceedings, of the *1st International Workshop on Software Reusability*, held in Dortmund in 1991. 'Instead of just publishing the accepted . . . papers, the editors . . . . decided to try to produce a more coherent result . . . [to] present the major stream[s] of discussions and . . . [to] cover particular aspect[s] . . . in more technical depth or give clarifying examples . . . [T]his . . . produced a[n] . . . exhaustive coverage of . . . research in software reusability'.

There are six chapters (numbers of references in parentheses): An historical overview, with a convenient research framework (39); domain analysis methods, of which eight are described, compared, contrasted, and evaluated (36); managerial and organizational issues, with a brief survey of international practice and less brief consideration of many issues (66); formal methods, with three formalisms described (62); tools and environments, with three specific environments described (41); and empirical studies—a brief framework proposal (11).

Most welcome, perhaps because unexpected, is the chapter on formal methods. What, after all, are more reused than the standard solutions and methods of logic and mathematics? '[R]euse provides the economic foundation for the use of formal methods; the iterated use of knowledge amortizes the higher development costs raised by its formal description'.

'This book', says the blurb, 'will be appropriate for researchers, postgraduates . . .' And so it will. If you are researching in software reuse, this is the place to start. '. . . final year undergraduates in software engineering and computer science'. Well, perhaps just a teeny bit turgid for some of them. But add to that list of readers: the purveyors of facile arguments that the latest silver bullet will deliver reuse (I name no names). Reusability is not merely, not even principally, 'of components in composite structures': it is 'of resources in performing a task'; indeed 'of everything associated with a software project including knowledge'. And the 'managerial, economic, social, cultural, and legal . . . problems are as important [as], and more difficult to solve than, the technical problems'.

Expensive for a slim volume? But iterated reference to it will amortize the acquisition cost.

ADRIAN LARNER
de Montfort University

ALAN SOUTHERTON AND EDWIN C. PERKINS, JR
The UNIX and X Command Compendium: A Dictionary for High-Level Computing. John Wiley. 1994; ISBN 0-471-30982-5. £17.95, 640 pp. softbound.

The format of this book is very simple. It contains a list of over 2000 UNIX commands. Each entry displays the command, a short paragraph describing what it does, relevant keywords (which are cross-referenced in the index at the end of the book), a list of files which the command uses and a 'see also' list of related commands. For each command, indications are given as to which UNIX shells it can be used with (ksh, csh, sh, etc.), which versions of UNIX support it (BSD, SVR4, SCO, AIX, SunOS, etc.) and which type of user would typically need it (system administrator, end user, shell script writer, or experienced UNIX user). The sorts of commands listed are not just 'simple' commands (such as `date`), but also moderately complex pipes and other commands understood by a shell. These are presented as examples of those commands, such as

```
nroff -Tepson book.nr
```

where actual options and arguments are included, instead of the perhaps more common

`nroff -T` *printer filename*

I found many useful commands in the book with which I was unfamiliar; for instance the use of `dirs`, `pushd` and `popd` for manipulating a stack of directories. This is a neat feature of several shells which makes it easy to memorise directories you have visited and need to `cd` to again later. Unfortunately there are omissions and (a very few) errors. I found no mention of my favourite newsreaders (`nn` and `trn`)—the one which is discussed is `news`, a relatively primitive newsreader. The entry for `mesg`, though correct for most systems was wrong for mine.

At the end of the book are appendices serving as quick reference for the Vi and Emacs editors and shell special characters, plus the keyword index.

A most unusual book. I found it difficult to use as a reference work, but I thoroughly enjoyed browsing through it. The most useful sections of the book were the appendices. Although you normally use your favourite editor and shell, sometimes you need to use others (when presented with a script written for another