B.W.BOEHM SOFTWARE ENGINEERING ECONOMICS

A REVIEW ESSAY

A.Bryant* and J.A.Kirkham+

The current tendency for software costs to exceed hardware costs is the effect of both an absolute increase in the former and decrease in the latter. Software development is now the major cost factor for any project concerned with the introduction or enhancement of electronic data-processing (EDP) equipment: it is also the most difficult for which to produce acceptably accurate estimates - both in terms of monetary cost and development schedule. In response to the growing dissatisfaction caused by the all-too-frequent tendency for final software development costs and schedules grossly to exceed initial estimates, attempts have been made to perfect estimating methods and models which can claim at least a reasonable degree of accuracy, providing a realistic basis for budget planning and labour allocation.

If the prime impulse behind these efforts can be termed that of 'budgetary credibility', there is also another, broader but less precise, impulse which stems from the non-budgetary effects of the adoption of EDP techniques. Some cost models ignore this second aspect, others seek to account for it through a wider concept of 'cost' derived from the field of 'cost-benefit analysis'. An example from the latter category is provided by the Constructive Cost Model developed by Barry W. Boehm and explicated in his recent book "Software Engineering Economics", Prentice Hall, 1981.

The reception accorded to Boehm's work has been little short of rapturous, some have described it as a sort of data processing managers' bible which should be on hand whenever software budgets are being decided and allocated.

The current authors, however, must demur from any such approval in the light of a careful study of Boehm's text and an implementation of the first two versions of his model. This is not to undermine the importance of the questions to which Boehm's argument is addressed, but instead to temper the praise which it has evoked.

Boehm introduces his argument by stating that the objective of the book is to 'equip you to deal with software engineering problems from the perspective of human economics as well as from the perspective of programming' (p.1). The intention then is to claim a wider perspective for Software Engineering Economics (SEE), which will be implemented in the COCOMO approach.

Initially Boehm illustrates his concept of this new approach with two examples. In the first, a case study of the development of an information processing system for Scientific American, the point is made that what at first sight appears to be a perfectly acceptable and well-programmed system may, once implemented, lead to disastrous results due to neglect of certain key operational elements. This can be avoided by considering not merely the immediately obvious 'programming' aspects of the implementation, but other facets such as those concerned with 'operational problems; budget problems; schedule problems; problems in determining the relative priorities of users' needs' (p.8).

---

* MFT Computers Ltd., Charles House, Low Lane, Horsforth, Leeds LS18 5DE.
+ Postgraduate School of Studies in Computing, University of Bradford,Bradford West Yorkshire, BD7 1DP.

The second case study further stresses the necessity of expanding the purported limits of orthodox perspectives in the light of the experience of the effects of the introduction of electronic data processing (EDP) to an urban school attendance system in USA. In this instance the programming and economic goals were both achieved, but to the disadvantage of the attendance clerks who would be made redundant upon implementation of the system. These clerks were for the most part women, with school-age children, from the poorer sections of the city, often relying upon such employment both in terms of the income and the advantage that the situation and hours of work coincided with their children's school hours. Once denied this opportunity, they would either have to seek work which afforded neither of the latter expediencies or claim welfare.

Boehm, having himself been part of the team involved in the second example, urges that not only must software development encompass non-programming aspects of system implementation, but that this wider perspective must not be restricted to what he terms the 'material-economics approach'; instead the 'human-economics approach' must be considered (p.12). Unfortunately Boehm gives neither an indication of how this objective is to be achieved, nor what it would entail. He somehow believes that the development away from 'a primarily production-oriented economy toward a primarily service-oriented economy' (p.12) represents the substance of such a transition. Yet this seems more than questionable given the following observations. The growth of the tertiary (service) sector and the decline of primary (agriculture) and secondary (manufacture) sectors is not particularly straight-forward in advanced economies, and is not applicable to third-world and many non-capitalist ones. Moreover, within economies which do exhibit this tendency in some respects, the concept of a commodity has expanded to include an ever wider range of goods and services. Consequently the constraints of 'material-economics' (whether Keynesian, monetarist, or other) continue to apply. Finally, given the large, and increasing, proportion of computer personnel - of all levels and abilities - engaged in military projects - offering neither service nor product in the usual sense - Boehm's suggestions seem misdirected and incomplete. His grasp of economics is clearly deficient, what then remains to be seen is whether or not the concept of software engineering economics and the cost model offered are of relevance and use in a more limited sense.

The approach to software engineering is termed the GOALS approach: 'Goal-Oriented Approach to Life-cycle Software'. Fundamentally this embodies a 'control loop which involves periodic review and iteration of the programming products with respect to a more general goal structure' (p.15). At the general level this includes the recommendation that all projects should establish a full range of objectives and thus of constraints involved. Each objective then forms all or part of a subgoal, the solutions for which must be developed in the light of the constraints imposed by all other goals; the process to include iteration where necessary. Summarizing his argument to this point, Boehm offers the following definition of software engineering: 'the application of science and mathematics by which the capabilities of computer equipment are made useful to man (sic) via computer programs, procedures and associated documentation' (p.16).

Boehm's widening of perspective does not then in detail amount to a qualitative advance upon those more restricted ones concerned solely with computer implementation; the predominant focus rests on programs, procedures and documentation. This assessment is further supported by the statement that the major challenges for the software engineering profession are those of increased software development productivity and increased efficiency of software maintenance (p.18). However much he may appear to wish to include 'humanitarian' aspects in the field of software engineering economics, it is clear that the basis of Boehm's position, and of COCOMO itself, emanates from a more limited set of concerns, more immediately connected with software development in the orthodox sense of the term. The discussion of 'conflicting objectives' and of Weinberg's experiments (1) give exclusive consideration to orthodox software factors such as development effort, program clarity and memory requirements. This is not to deny that COCOMO may well prove to be of use, but only in a far more localized sense than that promised in Boehm's introduction. (Indicative of this is that none of the factors incorporated in COCOMO relates to the 'humanitarian' aspect seemingly of such central concern to the author.)

In turning to consider Boehm's model in this more limited light the initial problem concerns the accuracy and reliability which is claimed The rationale behind any software estimating model derives from the indeterminate nature of software development and maintenance, together with the demonstrable tendency for actual expenditure to outstrip initial estimates, often by a factor of ten or more. Given the well established observation that software costs are increasing both in terms of overall proportion of total EDP costs, and as an overall proportion of gross domestic product (see Boehm, pp17-18), it is understandable that there is a good deal of interest in the perfection of methods for producing initial cost-estimates that prove to have a high degree of reliability. No cost estimating model, in whatever field, can ever guarantee complete accuracy, but merely claim a degree high enough to ensure that initial figures bear some close relationship to final expenditure. The current norm for software models is, according to Boehm, to be within 20% of actual costs, 70% of the time (p32). Often even this degree of accuracy is only achieved with regard to a specific category of projects. Boehm, however, claims that the 'Intermediate' and 'Detailed' versions of COCOMO in achieving this level of reliability for a wide range of applications 'provide a good deal of help in software engineering economic analysis and decisionmaking' (p32). A claim seemingly accepted unquestioningly by reviewers of the book. The assumptions and limitations behind the claim, however, need to be clarified before any conclusion can be reached.

---

1. This experiment consisted of giving different directions to five programming teams regarding the optimum aspects of the same task. For instance one team was required to complete the project with a minimum amount of effort, another was told to optimize output clarity. Four of the five achieved top rating for their particular primary objective - the other achieved second from top - but all failed to perform consistently well with regard to other teams' primary objectives. From this Boehm concludes that successful software development needs to encompass a range of possibly conflicting goals. (See pp.20ff and references listed there). G.M.Weinberg and E.L.Schulman, "Goals and Performance in Computer Programming" Human Factors, 1974, 16(1), 70-77.

COCOMO consists of three upwardly compatible versions of the model: 'Basic', 'Intermediate', and 'Detailed'. The basic level is designed to do no more than return initial order of magnitude estimates derived solely from project size and development mode (see below). The other two models are more complex in introducing a range of 'cost-drivers' (i.e. factors which influence development and maintenance effort and duration) which, depending upon the way they are set, increase or decrease the nominal values for the number of labour units, development schedule and so on. The nominal values are merely dependent upon project size.

For a project to be amenable to COCOMO cost-estimating its size must be calculated in terms of 'delivered source instructions' (hereafter DSI), expressed in units of 1000 (hereafter KDSI). This forms the basis for all further results and analyses.

The concept of 'delivered source instructions' is defined as follows:-

Delivered. This term is generally meant to exclude nondelivered support software such as test drivers. However, if these are developed with the same care as delivered software, with their own reviews, test plans, documentation, etc., then they should be counted.

Source Instructions. This term includes all program instructions created by project personnel and processed into machine code by some combination of preprocessors, compilers, and assemblers. It excludes comment cards and unmodified utility software. It includes job control language, format statements, and data declarations. Instructions are defined as lines of code or card images. Thus, a line containing two or more source statements counts as one instruction; a five-line data declaration counts as five instructions (pp 58-9).

As it stands this definition and the context in which it is meant to apply are strangely contradictory. For the Basic version of the model the project size, in KDSI, is the prime variable, and is meant to provide the basis for early order of magnitude estimates. Yet the calculation of KDSI would appear to rely upon fairly detailed knowledge of the project which at that stage would not necessarily be easily and readily available. Boehm fails to consider this matter, and neglects to explain how at the very least some estimate of project size in these terms can be ascertained from an initial outline of the form and nature of system implementation and objectives. Even within the more limited sense within which COCOMO is being considered this is a major flaw. In none of the examples of COCOMO calculations does the author indicate any method for calculation of project size in KDSI, the number is simply stated or introduced as the result of 'an initial study' (eg see p63). If the alleged reliability of the model is reassessed in the light of a fairly optimistic estimate of the inexactitude of the initial figures for project size (20% in either direction) this must severely undermine any claims to 'reasonable accuracy'. The following

figures illustrate this:-

Basic Estimates for Project of 32.0 KDSI accuracy of 20%

Organic Development Mode

Size	Labour Units	Schedule	Staffing Levels
KDSI	Lab Months	Months	Personnel
25.6	72.24	12.71	5.69
32.0	91.34	13.90	6.57
38.4	110.56	14.95	7.40

On the face of it this may be an acceptable range of values, although since Boehm himself later costs a labour month at $5000-$6000 at 1980 labour prices this would produce estimates ranging from $361,200-$552,800 at $5K per labour month and $433,400-$666,400 at $6K per labour month.

This range is even larger for the other development modes.

Embedded Mode

Size	Labour Units	Schedule	Staffing Levels
KDSI	Lab Months	Months	Personnel
25.6	176.26	13.08	13.48
32.0	230.54	14.26	16.17
38.4	286.74	14.84	19.32

At $5K per labour month the estimate ranges from $880,000-$1,444,000; at $6K from $1,050,000-$1,720,000.

It might, however, be argued that despite appearances this is still a substantial improvement upon previous estimates, and that, in common with all economic models, a certain - and appreciable - degree of inaccuracy is inevitable. Since Intermediate and Detailed models are constructed to produce multipliers which alter nominal values by factors as high as 11, such early initial inaccuracies can only be further exacerbated: on the other hand, if the effort multipliers only alter nominal values by a fraction then any effect would fail to balance initial inaccuracies. All the 'sophistication' outlined in the second and third levels must be seen to rest upon possibly intolerably crude foundations.

All this is not to deny that the concept of delivered source instructions may well be useful, particularly when considering adaptation or maintenance of existing software (although even here application is not straightforward - see below); or perhaps when the size of the project to be developed can be estimated

from a large store of information concerning existing similar projects. As presented by Boehm, however, there is no obvious method by which it can be rendered applicable to the early stages of development of an entirely new project. The COCOMO approach to software engineering economics must then be judged further deficient both in its own terms and those of what would generally be required of a cost-estimating model. Claims for the novelty of the approach, together with its applicability, are rendered severely questionable.

It is, however, worth persevering with the discussion of COCOMO for at least two reasons. The primary one derives from software engineering economic models in general, the other - of far more limited import - relates to the possible utility of the specific model itself. Currently in almost all advanced socio-economic formations software development is taking place in a context characterized by general economic contraction and the expansion of applications of EDP techniques. Consequently there is great interest in developing the ability to predict and control software expenditure. Crises in profitability, inflation, and the current attempts to manage fiscal matters by application of 'supply-side' economics preclude many previously possible options open to seller or purchaser to counteract discrepancies between initial and final budget figures. Assuredly, whatever the failings and lacunae of COCOMO and other cost models, there will be an increasing number of attempts to produce satisfactory ones. In spite of the severe and disabling criticisms made of COCOMO, Boehm's book does seek to analyse and clarify many aspects of the software development and maintenance processes; possibly a contribution in itself to the development of the field of software engineering economics. The major part of what follows is devoted to this matter. It need merely be added, as the second reason alluded to above, that, if Boehm's discussion is found to offer a useful mode of analysis, it may be worthwhile devising some means of overcoming the problem of producing an initial estimate for project size in terms of DSI.

As was mentioned above, COCOMO is divided into three upwardly compatible versions; the distinctions between the three can be described as follows. Basic COCOMO returns initial estimates based solely on project size - in KDSI - and on development mode (see below). These items of data then form the basis for calculating the number of labour months; development schedule; number of full-time equivalent software personnel; productivity: the equations used being derived from the COCOMO database. In addition figures are presented in terms of 'effort' and 'schedule such that the duration and number of personnel required for each phase can be estimated. These figures are calculated via interpolation from sets of results for standard-sized projects. Intermediate COCOMO builds upon this. Project size and development mode are presented as before; initial equations are of the same form, albeit using different factors. Since effort and schedule figures are solely dependent upon project size these are no different from those produced using the simpler model for the same project size. The values for labour months, development schedule, and so on, are, however, nominal; later adjusted to final values according to the resulting figure for the overall 'effort adjustment factor' (EAF). This factor is the product of fifteen 'effort multipliers' each representing a 'cost-driver' which, initially set to a nominal value of 1.0, can be altered by the user depending upon desired enhancements or tolerable deficiencies in specific aspects of the software product. Finally Detailed COCOMO takes into account these effort multipliers not simply as overall effects on the project, as in the Intermediate form, but as producing specific and varying results on each individual phase for both development and maintenance. (The Detailed model will not be considered further in this discussion.)

All the COCOMO estimating equations are derived from analysis of the COCOMO database. This consists of details for 63 projects covering a wide variety of different applications and development contexts. Nearly 75% of the projects were developed in the period 1975-79; 50% were implemented on mainframes (see Boehm, p83 for a summary of project characteristics, pp494-99 for a more detailed report).

Plotting the actual figures for each of the projects against the estimates produced by each version of COCOMO provides the basis for the claimed level of accuracy. (The present authors are unable to comment on the methodological intricacies involved in using the same set of data both for deriving the set of estimating equations and establishing their degree of accuracy. Boehm includes no justification for this step.)

The model itself derives from a specific account of 'the software life-cycle' which consists of nine subgoals each linked to a particular phase of development. The inter-relationships between the phases are characterized by the term 'waterfall model' in which 'as much as possible iterations of earlier phase products are performed in the next succeeding phase' (p36). The ninth phase 'phaseout', concerns the 'clean transition of the functions performed by the product to its successors (if any)' (p37). In addition, two further subgoals are sought during each phase of the software life-cycle: 'verification and validation', and 'configuration management'.

Quoting several studies in support, Boehm argues that the subgoals are both necessary and sufficient for developing any software product; furthermore the proposed sequence is the most effective and successful (pp38-41). COCOMO is then designed to enable estimates to be obtained of the proportion of effort spent in each phase of the life-cycle. In addition each phase can be further analysed in terms of eight major project activities as follows:-

1 Requirement Analysis
2 Product Design
3 Programming
4 Test Planning
5 Verification & Validation
6 Project Office Functions
7 Configuration Management/Quality Assurance
8 Manuals

The reader must be warned at this juncture that the 'Plans and Requirements' phase has a somewhat peculiar status. For all levels of COCOMO, 'P&R' is assumed to form an independent stage of production prior to development. The values for this phase are not included in total product figures: their process of derivation is not elucidated, but it can be inferred that they are attained by extrapolation backwards from figures for completed projects (pp46-52).

Apart from dealing with the development of software products, COCOMO also covers the ensuing maintenance. Maintenance is defined as 'the process of modifying existing operational software while leaving its primary functions intact' (p54). This includes both updating and repairing activities, but specifically excludes major redesign and redevelopment (ie more than 50% of new software product performing functions; design and development of sizeable software interfaces; 'data processing system operations, data entry, and modification of values in the data base' (p54)).

An additional factor in the development of a software project is its mode of development. Boehm offers three; organic, semi-detached, embedded. The organic mode is characterized by 'relatively small software teams' developing 'software in a highly familiar, in-house environment' (p78). This results in lower communications overheads for the duration of the project, and also only a comparatively small loss in productivity due to these overheads as project size increases.

With regard to embedded mode 'the major distinguishing feature ... is a need to operate within tight constraints' (p79). Such projects need constant and extensive verification and validation, as well as rigorous configuration management and quality assurance, to ensure that development maintains compatibility with the existing 'strongly coupled complex of hardware, software, regulations, and operational procedures' (p79). Being concerned to a far greater extent with entirely new projects than development in organic mode, communications overheads are proportionately greater at every stage. The labour distribution curve is more pronounced than that for organic mode. In the early stages a small team of analysts would be used to complete product design, followed by introduction of a far larger team of programmers to carry out detailed design, coding, and unit testing.

Semi-detached mode is an intermediate one between organic and embedded. A project may be intermediate in one of two respects. Either it falls between the other two modes in terms of its overall characteristics, or it may be a mixture of components of different mode.

Boehm's model also attempts to account for the diseconomies of scale inherent in software development. Taking the foregoing discussion as whole it would be expected that Basic COCOMO would be designed to produce the following values:-

1 Total number of labour units
2 Expected development schedule
3 Number of personnel required
4 Phase distribution in terms of
    a labour units
    b development schedule
    c effort
    d personnel required

The basic unit of labour expended is the 'labour month' (hereafter LM) consisting of 152 hours of working time (Boehm, p59). The Basic COCOMO equations for number of labour months are as follows:-

Organic	LM=2.4(KDSI)**1.05
Semidetached	LM=3.0(KDSI)**1.12
Embedded	LM=3.6(KDSI)**1.20

The development period (TDEV) is then given in terms of the number of labour units as follows:-

Organic	TDEV=2.5(LM)**0.38
Semidetached	TDEV=2.5(LM)**0.35
Embedded	TDEV=2.5(LM)**0.32

(For derivation of these equations see Boehm, chapter 29.)

The number of personnel is then obtained from the ratio of number of labour units: development schedule - (LM/TDEV). This produces a value termed 'full-time equivalent software personnel' (hereafter FSP).

Productivity can then be estimated from the ratio of project size: number of labour units - (KDSI/LM).

For each development mode Boehm then produces values for standard product sizes:-

Basic Project Profiles
Medium-Size Projects

Quantity	Mode		
	Org	Sem	Emb
Total Effort (LM)	91	146	230
P&R	5	10	18
Design	15	25	42
Programming	56	85	124
Detailed Design	22	37	60
Code/Unit Test	34	48	64
I&T	20	36	64
Total Schedule (Months)	14	14	14
P&R	1.7	2.8	4.5
Design	2.7	3.6	4.8
Programming	7.7	6.8	5.6
I&T	3.6	3.6	3.6
Average Personnel (FSP)	6.5	10.4	16.4
P&R	2.9	3.6	4.0
Design	5.6	6.9	8.8
Programming	7.3	12.5	22.1
I&T	5.6	10.0	17.8
Percent of Average Personnel			
P&R	45	35	24
Design	84	66	54
Programming	113	120	135
I&T	85	96	108
Productivity (DSI/LM)	352	219	139
Code/Unit Test	941	667	500

(Figures taken from Boehm, p92)

The following observations are pertinent. The estimated effort (1) increases between organic and semidetached modes, and between semidetached and embedded modes. Moreover, as project size increases there is an accompanying 'diseconomy of scale' (p76); the rate of decline in productivity increases. Development schedule varies with product size independent of development mode, hence development in embedded mode requires most effort for a given project size; organic mode requires least. Again there is a diseconomy of scale, the rate of increase in FSP increases with product size.

Analyzing these figures with respect to the software life-cycle produces a further set of standard tables. Boehm does not seek to offer separate values for each of the nine phases mentioned above, instead he amalgamates them under four headings; 'Plans and Requirements'; 'Product Design'; 'Programming'; 'Integration and Test' - the third is subdivided into 'Detailed Design' and 'Code and Unit Test'.

For each standard sized project there is provided a breakdown in terms of these categories for both effort and schedule values. These reflect the basic labour distribution for the specific mode. For all three modes there is an increase in effort between the start of product development and completion of 50-60% of development schedule, after which there is a corresponding decrease. For any given product size a set of values for all these features can be obtained by interpolation based on relevant standard values. In practice these also include the proportion of each of the eight major activities for each phase.

Maintenance values are calculated on the basis of 'annual change traffic': 'the fraction of the software product's source instructions which undergo (sic) change during a typical year, either through addition or modification ' (p71). The maintenance effort is then given by the product of this fraction and the number of labour units for development: the annual requirement for maintenance staff will then be this figure divided by twelve.

---

(1) With regard to effort values, the embedded mode requires 'considerably greater effort' in terms of the proportion devoted to 'Integration and Test' phase (p89). This is a result of the greater care demanded in adhering to and rectifying software requirements. It should be noted that because development schedule increases markedly between organic and embedded modes, the proportion of effort spent in any particular phase of the latter may be less than in the corresponding phase of the former: the actual effort, expressed in labour months, will, however, be greater.

Similar remarks apply to different phases for schedule values. For embedded projects a larger proportion of time will be spent in 'Plans and Requirements' and 'Design' phases, since they require high levels of validation and specification, but use a smaller number of personnel than later phases. (If the number were to be increased, the communications overheads would become a major factor - see Boehm,p80.) Since the number of personnel increases for the 'programming' phase, a lower part of the estimated development schedule will be spent at this juncture.

The major limitation of Basic COCOMO, to which Boehm admits, is that it only takes account of DSI - and annual change traffic for maintenance. Other factors relating to hardware constraints, personnel, and so on are neglected. In addition, the estimates for full-time software personnel are for average staff levels for each phase, giving rise to discontinuities at boundaries. Intermediate COCOMO is designed to deal with most of these aspects.

Intermediate COCOMO provides a 'compatible extension' (p114) of the Basic model, providing greater accuracy and more detailed control by the interested user. This facility arises as a result of the incorporation of '15 predictor variables' (p114) or cost drivers. Boehm claims that this effects a level of accuracy of 'within 20% of the project actuals 68% of the time' (p115).

Previous models, such as those based on the System Development Corporation studies in the 1960s (see p155 and references there cited), sought to encompass as many as 104 different factors including type of application, expertise level of analysts and programmers, program complexity, language used and amount of travelling required. Boehm offers a set of 15 cost-drivers which he considers

RELY	Required software reliability
DATA	Database size
CPLX	Product Complexity

Computer Attributes

TIME	Execution time constraint
STOR	Main storeage constraint
TURN	Computer turnaround time

Personnel Attributes

ACAP	Analyst capability
AEXP	Applications experience
PCAP	Programmer capability
VEXP	Virtual machine experience
LEXP	Programming language experience

Project Attributes

MODP	Modern programming practices
TOOL	Use of software tools
SCED	Required development schedule

After due consideration of the nature of a specific project, the user can set each cost-driver to a desired rating - lower or higher than an initial nominal rating. In total there are six possible ratings - very low, low, nominal, high, very high, extra high - although in practice no more than five are applicable to any single cost-driver: for example the cost-driver 'Analyst Capability' cannot be set to a rating of 'extra high'. The effect of altering a cost-driver rating

from nominal is to alter its corresponding 'effort multiplier' from a value of 1.0. For example the effort multiplier for a 'high' rating of required software reliability is given as 1.15; for a 'low' rating it is 0.88. The implication is that to develop a software product of high reliability will take longer and require more labour units than will the same product if developed with a nominal level of reliability.

Initially, in similar fashion to that for Basic COCOMO, project size and mode are ascertained and then used to produce values for effort (LM) and schedule (TDEV), etc. In this case, however, the estimating equations are not exactly the same for calculating LM.

Organic	LM=3.2(KDSI)**1.05
Semidetached	LM=3.0(KDSI)**1.12
Embedded	LM=2.8(KDSI)**1.20

The equations for calculating TDFV are the same as before. These values are merely nominal ones; the final ones being dependent upon the product of the effort multipliers for each of the 15 cost-drivers - the 'effort adjustment factor' or EAF.

For a project of 128 KDSI, to be developed in embedded mode, the nominal effort value will be 945.8 LM. If all cost-drivers are left at nominal ratings, then this would be the final estimate for number of labour units; development schedule would be 22.4 months; average full-time software personnel 42.2; productivity 135 DSI per month. Should it be decided to develop the same product with high reliability, then the effort multiplier of the cost-driver 'RELY' would be set to 1.15 and the effort adjustment factor ('EAF') would be 1.15. As a result the value for LM will be altered from 945.8 to 1087.7, giving adjusted values for TDEV (23.4), FSP (46.5), and PROD (118).

Intermediate COCOMO is designed to allow users to specify the nature of the project in more detail, and further to perform sensitivity analyses by altering cost-driver ratings and noting the resulting effect on LM,TDEV, etc. A brief glance at the values relating to individual cost-drivers shows that for the first two categories - Product and Computer attributes - setting the cost-driver to a rating above nominal increases the total effort required (effort multiplier is greater than 1.0); while setting it to a lower rating has the opposite effect. Two of the Computer attributes, however, - 'TIME' and 'STOR' - cannot be rated lower than nominal. For the last two categories - Personnel and Project attributes - for all but the last cost-driver the converse applies: increasing the rating lowers the total effort required, and vice-versa. For 'SCED' any alteration from nominal will increase the total effort required.

Apart from presenting a table of what each rating for each cost-driver entails, Boehm for the most part only deals specifically with 'RELY' and some of the personnel attributes. The ramifications of altering some of the others are left wholly or partially unexamined.

The 'RELY' cost-driver is set according to the nature of the project. It will be set to 'very high' if system failure would present a risk to human life - eg a nuclear reactor control system (p121), and to 'high' if heavy financial loss would be incurred. Lower ratings would depend upon acceptable levels of failure

and the benefit obtained from decreased development costs as a result of the need for fewer labour units. Once developed, however, the system must be maintained, and it would seem obvious that if the software has been developed to a certain degree of reliability it would be maintained at a similar level. Moreover the effort to maintain a system developed with a 'low' or 'very low' level of reliability would be expected to be greater than if it had been developed at a nominal level; COCOMO allows for this by incorporating a different set of values for this cost-driver for the maintenance phase. From these it can be seen that Boehm adjudges maintenance effort for 'low' and 'very low' projects to exceed that even for 'very high' ones. Once recognized, this would seem to militate against any user deciding to lower the rating below the nominal level. (More details of the ramifications of altering this cost-driver are given by Boehm in table 8-5,p123.)

Database size and 'CPLX' cost-drivers are more straightforward, and the relevant ratings are easier to ascertain from the details given. Boehm fails to point out that 'TIME' and 'STOR' cannot be rated lower than nominal. If for instance 'STOR' has been set at nominal this would imply that any increase in memory would have no effect on development or maintenance effort. This may well be true, and the limit of $\frac{1}{4}$=50% use may accurately reflect this; but Boehm neglects to justify this constraint.

Personnel attributes are the most directly cost-sensitive category, and the book covers some examples of the ways in which sensitivity analyses can be used (ppl25ff). Thus the decrease in EAF resulting from a 'high' or 'very high' rating for 'ACAP','AEXP', or 'PCAP' will in turn lead to an increase in unit labour costs; an analyst with six or twelve years' experience being paid more than one with only three years' experience. This increase may not, however, lead to an increase in total developrnent costs: on the contrary, since the number of LMs will decrease, the total labour costs may decrease. From the scale of values given, employing analysts and programmers with the highest possible levels of expertise would reduce the number of labour units to approximately 35% of nominal values. Whether or not this leads to decreased costs overall would depend upon prevailing pay and overhead differentials, and availability.

Finally to turn to the category of Project attributes. MODP refers to such factors as structured code and top-down design (p130). If applied to the entire project for routine use, these practices are estimated to reduce development labour units by 18%. Once developed, the effect on maintenance becomes dependent upon project size; so, similar to RELY, a separate set of values is required. This arises for two reasons; the greater extent of application of modern programming practices (MPPs) the less maintenance will be required, and the lower are the effects of the diseconomies of scale for larger projects. Cost-driver TOOL refers to the range of compilers, assemblers and loaders - as well as diagnostic aids - available. Clearly if these are primitive and unreliable they will increase nominal values; if they are highly developed and efficient they will decrease the values.

SCED only applies to development phase. Any alteration from nominal will increase the final EAF. None of the examples given deal with ratings other than nominal, and it is unclear how these would operate in practice. For instance if SCED is to be 75% of nominal this will increase nominal values by 23%. So a

project requiring 10 LM would then need 12.3: the value of TDEV being reduced from 5.22 to 3.92 months. Using this value of TDEV gives 3.14 FSP as opposed to the initial value of 1.92. If, however, SCED were set at 'very high', this would produce a 10% increase, giving TDEV of 8.35 and 1.32 FSP. It is not clear why any alteration of this cost-driver will increase EAF. Obviously if the project is to be completed within a shorter time, an increased effort will be required. But if the project can be planned to take longer then there seems to be no obvious reason why overall effort should increase. In his later discussion of Detailed COCOMO, Boehm mentions that in 'schedule-stretchout situations (sic!)... there is more time for thorough planning, specification, and validation' in the earlier phases (pp466ff). This, however, seems to have no in-built positive effect on, for instance, maintainability: so it would appear to be a wasted expenditure of effort - at any rate Boehm takes the matter no further.

In implementing Intermediate COCOMO, it became clear that although there may well be a strong case for considering all 15 cost-drivers as independent factors, some attention should have been given to specific combinations and their ramifications. Setting MODP to 'very high', TOOL to 'very low', and PCAP to 'very low' would seem rather impractical; and although the corresponding EAF of 1.44 ought to underline this, it would disguise the fact that had MODP been left at nominal the EAF would have been 1.76. It does not seem clear why routine use of MPPs reduces the EAF even when programmers with less than one month's experience are used. It might be suggested that, on the contrary, the EAF should increase as a result of such a combination. It might be contended that such amalgams would be ruled out as a matter of course by knowledgeable project planners, but this would fail to satisfy the criticism that perhaps the 15 factors are not entirely independent variables.

So far little has been said about adaptation. In the discussion of the estimation of DSI it was observed that with respect to the adaptation of existing software the concept may be more readily applicable. Adaptation, however, is not without specific problems itself. As Boehm notes 'no quantities are as easy to under-estimate as are the estimates of how much one will have to change an existing piece of software to set it to work successfully in a new product environment' (p138). A good deal of effort will need to be expended on redesign, reworking of sections of code, and integration and testing. To this end Intermediate COCOMO incorporates a technique for quantifying these tasks and thereby arriving at an 'Adaptation Adjustment Factor' (AAF) which, normally, scales down the project size of the existing software. AAF is dependent upon 'percent design modified' (DM), 'percent code modified' (CM) 'percent of integration required for modified software' (IM) as follows:-

$$AAF = 0.4(DM) + 0.3(CM) + 0.3(IM)$$

Equivalent project size is then given as

$$EDSI = (ADSI)AAF/100$$

where EDSI is the equivalent size in DSI and ADSI is the size of the software to be adapted.

For a simple conversion, such as that required when adapting a program to run on a different installation, typical figures might be

DM = 0 no design change
CM = 15 15% code change
IM = 5 small effort needed
AAF = 0.4(0) + 0.3(15) + 0.3(5) = 6
If ADSI = 50,000
EDSI = 50,000(0.06) = 3000 DSI

(Boehm,pp134-5)

The coefficients for the three factors are derived from the 'general average fractions of effort devoted to design, code, and integration and testing' given by COCOMO values for standard size projects (p137). In practice it may be desirable to alter them slightly to reflect changing COCOMO values across project size and development mode, but since the calculation of AAF is not very sensitive to such differences, Boehm recommends adhering to average values.

The second form of Intermediate COCOMO is concerned with a more detailed level of estimates pertaining to individual components of a software product. Boehm considers that the macro model will be used in earlier stages of a project, and the micro model applied to the more detailed stages later in development.

Although not expressly described as such, the reader is lead to assume that each component forms a clearly demarcated subsection of the overall product, and attains an independent existence in terms of development. This must be the case given that any component may be a piece of adapted software or a newly developed program. Also the purpose of applying Intermediate COCOMO to individual components is to aid in assessing the effect of distinguishing between different groups of people with differing levels of capability to various components depending upon their complexity. In the light of this, a major aspect of this version of Intermediate COCOMO seems questionable as presented by Boehm. In producing figures for component level estimates, Boehm adds all component sizes together and produces an overall nominal set of figures for number of labour months, schedule and productivity. The number of labour months for any individual component is then calculated on the basis that the ratio of component size to total project size is identical with that between number of labour units. Such an assumption overrides two other, related assumptions made with some force by Boehm. The first concerns the diseconomy of scale inherent in all software projects. The second derives from the functional modularity which allows components to be distinguished and treated as separate entities. In other words splitting a project into discrete parts should lead to a decrease in the total effort required for two reasons: each component will be smaller than the total, and hence diseconomies of scale will be less marked; functional modularity will further reduce this. Boehm gives as an example a project of three components, respectively 7000, 5000, 10000 DSI (pp146ff - NB for no apparent reason Boehm uses DSI rather than KDSI in this chapter). The total size is then 22000 EDSI - no component being adapted from existing software. For a project of this total size, developed in organic mode, the effort will be 82 LM. Using Boehm's method, the effort for each component will be given by (component size/total size)*82: Producing values of 26, 19, 37 respectively. If instead of this the effort for each component is calculated separately such that

effort is given by 3.2(size)**1.05 the figures are 24.69, 17.34, 35.90 respectively. This yields a total of 77.93 LM, 5% lower than the value of a project of total size 22000 DSI. Whether or not this is in any way an accurate reflection of the effects of reduced diseconomies of scale and modularity is both beyond the scope of this review as well as the ability of the present authors. At first glance it seems no more or less questionable than do other aspects of COCOMO as presented by Boehm, and the justification behind its derivation certainly seems more firmly grounded than does Boehm's unexplained use of straightforward proportions. For the remainder of this discussion, and in the implementation of component level estimating, the 'revised' derivation of individual component effort value has been substituted for that offered in Software Engineering Economics (pp146ff).

Boehm suggests that 'it is highly useful to collect and record intermediate level software cost estimating information on a standard form organized for the purpose' (p146). This is called a 'Component Level Estimating Form' (hereafter CLEF). A CLEF allows any software product to be divided into a maximum of ten components each of which can be analyzed and adjudged in terms of the 15 cost-drivers of Intermediate COCOMO - for both development and maintenance phases. (Should details for more than 10 components be required, the present authors would suggest that users develop a hierarchy of components and produce CLEFs for each structured subsection).

For instance a system may consist of a control component which must be developed and maintained at a very high level of reliability, and to which are to be allocated programmers and analysts of a corresponding high level of expertise. Another component may be concerned with I/O, adapted from existing software and not requiring ratings above nominal for any cost-drivers. Implementation of Intermediate COCOMO-micro would then result in each component having all 15 cost-drivers set, then producing 15 individual EAFs. The nominal number of labour units for each would then be altered accordingly, and the total would be the sum of all adjusted values. For maintenance each component would be allocated a value for ACT, and adjusted values would be similarly calculated.

The above discussion has sought to demonstrate the major deficiencies in Boehm's approach, without simultaneously undermining or undervaluing the concept of software engineering economics or the development of cost-estimating models. On the contrary, by pointing out the flaws in COCOMO the present authors hope to contribute to a developing area of work.

With respect to COCOMO itself the main criticisms can be considered in three groups. First there are those concerned with Boehm's intention of incorporating a 'wider' perspective in his approach. As was pointed out, the assumptions behind this 'humanitarian' framework are highly questionable, and Boehm spends no time explicating them in any satisfactory way. Furthermore they do not appear to play any significant part in the development of the model itself. It is interesting to note that reviewers talked of the book as a data processing managers' bible; indicating a far narrower readership and range of application than might have been expected given the intention of a wider approach. As it stands the reader can merely fantasize upon what sort of model it would be that could in some way take account of Boehm's intentions.

If the first group of criticisms concerns what is to be encompassed in the 'economics' of software engineering economics, the second refers to the model of software engineering or software development itself. Here the fundamental failing, possibly a fatal one, is the neglect of any discussion of the method for producing the initial figures for project size - the basis for all other estimates. If Boehm wishes to argue that this represents a specific problem prior to application of the model itself, then he must supply the reader with the rationale and guidelines for this phase. As presented in his account project sizes seem to appear from nowhere in particular: the 'Plans and Requirements' phase of software development is granted an ex post facto existence.

Finally there are a number of points which apply to the presentation of Intermediate COCOMO. These concern the 'independent' status of the 15 cost-drivers, and the ramifications of their different ratings. In addition there is the peculiar way in which the micro version makes use of proportions, ignoring the diseconomies of scale and functional modularity stressed by Boehm himself. Taken together these criticisms would seem to put in doubt much of the sophistication, accuracy and usefulness claimed for COCOMO. If they can be remedied then perhaps other aspects of the approach - the concept and description of the software life-cycle, the analysis of the phases and activities emanating from this; the sensitivity analyses and component level estimations - can be more fully utilised and their value estimated. In the meantime healthy scepticism rather than religious fervour would appear to be more suitable.

[I made a call to Barry Boehm concerning the foregoing review, and he offered the following comment. PGN]

I think the authors have indeed made a reasonably careful study of the first 9 of the book's 33 chapters, but their review does not seem to cover any material in the rest of the book -- which discusses many of the questions that they raise. B.W. Boehm