Discussion and correspondence
Measuring students' attitudes towards structured walk-throughs

R. S. Lemos

Department of Business Information Systems, California State University, Los Angeles, California 90032, USA

The effect of classroom team debugging activities (structured walk-throughs) on student's attitudes towards this process was empirically assessed with a sample of 87 undergraduate Business Administration majors enrolled in an introductory COBOL programming course. For each assigned program, students read and critiqued team members' program listings.
Attitudinal outcomes were measured by a 'perceived team effectiveness survey' constructed for the experiment. This survey consisted of positive and negative statements about programming which were measured on a Likert-type scale. Analysis of variance was used to check for the consistency of responses between classes and t-tests were used to test for differences from 'neutral' responses.
The significantly positive responses on 81.7% of the questionnaire items were found to be consistent between classes. These findings provide empirical support to the assertion that students perceive structured walk-throughs to be an effective pedagogical approach.

(Received January 1980)

With software costs continuing to increase, more attention is being focused upon formalised software design and development practices. One group of intuitively promising practices are the so-called 'group debugging' techniques. These techniques are based on the assumption that reviewing someone else's work provides the following benefits (Weinberg, 1971, pp. 54-60):

1. Error detection is improved since it is easier to detect errors (in both grammar and logic) in someone else's work than one's own. Weinberg describes this phenomenon as 'cognitive dissonance'.
2. Humans are not consistent with respect to the quality of their day-to-day work output. We all have 'bad days'. However, if our work is reviewed, there is a good chance that most errors will be detected by group members. This decreases the variability of work output.
3. Estimates of the progress that has been made will be more realistic since more than one person will be familiar with the work.
4. The maintainability of the software will be improved since more than one person must be able to understand how it works.
5. The most important alleged benefit is that reviewing someone else's work raises the proficiency level of the reviewer.

The last benefit is expected to be realised in one of two ways. First, if an 'expert' person is reviewing poor work, their debugging skills are going to be challenged since they must suggest the most appropriate modifications. This is difficult to do since the reviewer has to decide from several alternatives (including the recommendation to start again). Second, if an 'inexpert' person is reviewing quality work, their skills will be increased by being given the opportunity to study and be influenced by the work of their more proficient peers.

However, the formality of the group review process can vary quite a bit. At one end of the 'formality' continuum are the design or code inspections. As described by Fagan (1976), these inspections include a designated moderator with specialised training, definite participant roles, checklists of potential errors, process follow-up and detailed error feedback to the participants. The other end of the continuum is represented by informal walk-throughs that vary in terms of regularity and throughness. Yourdon (1975) and Hughes (1977) use the term 'structured walk-throughs' to describe a review procedure that is less formalised than inspections, but more formalised than walk-throughs. While similar to inspections in many respects, structured walk-throughs involve participant motivated review sessions and the team roles are less rigidly defined. One of the most well known advocates of the peer review process is Weinberg (1971) who supports the incorporation of these activities in a classroom environment that emphasises 'egoless programming'. Egoless programming encourages an open, shared programming approach that is essential for an effective group review process.

While incorporating group review procedures like structured walk-throughs into our instructional programs can be intuitively appealing, the following questions must be objectively answered:

1. Will instructor motivated structured walk-throughs increase the programming proficiency of students?
2. What will be the effect of structured walk-throughs on students' attitudes toward programming?
3. How will the implementation of group review procedures in the classroom affect students' ratings of the teaching effectiveness of the instructor?
4. How do students feel about the group review process itself?

This paper deals specifically with the last question and is based upon a research study that empirically addressed all of the above questions. The study involved the teaching of introductory COBOL to undergraduate business majors. Results showed that students participating in structured walk-throughs scored significantly higher on a programming examination (Lemos, 1979b). These students also exhibited more positive attitudes towards programming (Lemos, 1978). However, these students rated the instructor lower in selected indices of teaching effectiveness than did students who did not participate in team activities (Lemos, 1979a).

While these results are useful in evaluating the effectiveness of structured walk-throughs, it is also important to gauge student attitudes towards working in teams. Jones (1974) has discussed and emphasised the importance of developing team skills for entry level people. In addition, support for this position is definitely the current trend. Therefore, it is important that students perceive team activities in a positive manner. They will be experiencing such activities in their professional careers and will be more effective with positive attitudes.

This paper describes the effect that structured walk-throughs

had on students' attitudes towards the team review process, as measured by a post-experiment attitudinal survey. Specifically, student responses were analysed for significant differences from an 'undecided' response to the items on the questionnaire. Also, the three classes were checked for significant differences between sections on their responses to the questionnaire items.

Subjects

The subjects consisted of 87 undergraduate business majors who completed one of three sections of a required introductory COBOL course at California State University, Dominguez Hills. **Table 1** presents a general description of the sections. The three sections, taught by two instructors, had been randomly assigned to the treatment group that was to participate in the structured walk-throughs. Four other sections, totalling 128 students and taught by three instructors, did not participate in the team activities. All instructors had been teaching for about two years, had taught the course before, and had never used a team approach in their classes.

Procedure

Five programming problems were assigned during the quarter. The last program involved control break processing. For each of these assignments, students were required to bring to class (on designated days) a listing (not a run) of the current assignment. The students who complied with these instructions were randomly assigned to three-person teams. If participants did not number a multiple of three, then four-person teams were formed and worked in a 'round-robin' fashion. Each member of the team than proceeded to critique the listings of the other two members of the team formally. These written evaluations focused upon identifying all grammar and logic errors, and recommending the modifications necessary to correct the errors. The instructor's role was that of a consultant who could be called upon to answer any questions raised by the reviewers. At the end of the session, each participant used the two independent critiques of his/her listing to make any needed corrections. The students then ran their programs and reported back to the teams at the next session to discuss their results. Successive run attempts were the responsibility of the individual student.

The mean number of listings contributed by students to the team activities was 3·28 or 65·6%.

Table 1 The population for the study

Section	*Quarter*	*Class size*
A	Winter	34
A	Spring	27
B	Winter	26
		87

*Instructor A is the researcher.

The perceived team effectiveness survey

This questionnaire was developed by the researcher to obtain information on the attitudes of the students towards the team activities. The thirty items on this questionnaire surveyed attitudes towards: the teams in general; their fellow team members; the degree of team participation; the contribution of team activities towards learning; and the usefulness of the critiques. The questionnaire used a Likert-type scale to score the twelve positive and eight negative statements in the following way:

Positive statements	*Comment*	*Negative statements*
5 points	Strongly Agree (SA)	1 point
4 points	Agree (g)	2 points
3 points	Undecided (Und)	3 points
2 points	Disagree (Dis)	4 points
1 point	Strongly Disagree (SD)	5 points

Table 2 Descriptive and inferential statistics for experimental group responses on the 'attitudes toward team activities' questionnaire.

Item no.	*The experimental group*									*Test for differences between the sections*	
	Class 1 (n = 34)			*Class* 4 (n = 27)			*Class* 5 (n = 26)				
	Mean	*Std dev*	*Range*	*Mean*	*Std dev*	*Range*	*Mean*	*Std dev*	*Range*	*F*	*p*
1	3·62*	0·99	2-5	3·74*	0·98	2-5	3·54*	0·99	1-5	0·28	0·75
2	3·85*	0·96	2-5	3·74*	0.71	2-5	3·62*	0·94	2-5	0·53	0·59
3	2·71	1·14	1-5	2·85	0·99	1-4	2·73	0·87	1-4	0·17	0·85
4	3·44*	1·02	1-5	3·44*	0·97	1-5	3·54*	0·99	2-5	0·08	0·92
5	3·44*	0·89	1-5	3·30	1·03	1-5	3·08	0·94	1-4	1·08	0·34
6	3·82*	0·87	2-5	3·70*	0·82	2-5	3·54*	1·24	1-5	0·62	0·54
7	3·68*	1·00	1-5	3·85*	0·95	2-5	3·65*	0·94	1-5	0·34	0·71
8	3·53*	0·75	2-5	3·63*	0·74	2-5	3·39*	0·75	2-5	0·72	0·49
9	3·62*	0·95	1-5	3·89*	0·89	1-5	3·85*	0·93	1-5	0·77	0·47
10	4·06*	0·78	2-5	3·85*	0·99	1-5	4·12*	0·59	3-5	0·81	0·45
11	3·59*	1·02	2-5	3·67*	1·07	1-5	3·81*	1·06	2-5	0·33	0·72
12	3·35	1·10	1-5	3·56*	1·12	1-5	3·73*	0·92	2-5	1·47	0·25
13	3·65*	0·69	2-5	3·30	0·99	1-5	3·54*	0·76	2-5	1·42	0·25
14	3·44*	1·05	1-5	3·67*	0·56	2-4	3·77*	1·03	1-5	1·01	0·37
15	3·15*	0·86	1-5	3·74*	0·71	2-5	3·58*	0·86	2-5	4·38	0·016**
16	3·82*	0·67	2-5	4·11*	0·58	3-5	4·12*	0·71	2-5	2·01	0·14
17	3·62*	0·99	1-5	3·74*	0·98	1-5	4·12*	0·82	2-5	2·17	0·12
18	2·94	0·81	2-5	3·22	0·85	2-5	3·15	0·93	1-5	0·90	0·41
19	3·59*	1·08	1-5	3·63*	0·88	2-5	3·81*	0·90	2-5	0·41	0·67
20	3·47*	1·11	1-5	3·74*	0·98	1-5	3·81*	0·94	1-5	0·94	0·40

*Item questionnaire responses significantly different ($p < 0·05$) from a response of 'undecided' or neutral
**Item responses of the three classes are significantly different ($p < 0·05$) from each other.

Thus, index scores greater than 3·00 indicate favourable attitudes while index scores less than 3·00 indicate unfavourable attitudes. The questionnaire was administered on the last day of class and is shown in Appendix 1.

Results

The questionnaire responses were analysed with an extension of the Static-Group Comparison Design described by Campbell and Stanley (1963, p. 183). This extension was used for comparisons and single sample inferences of the three classes participating in the structured walk-throughs. It can be diagrammed as follows:

$$\begin{array}{ccc} - & X_1 & - \quad - & O_1 & - \\ - & X_2 & - \quad - & O_2 & - \\ & X_3 & \quad & O_3 & \end{array}$$

where

X_i treatment (structured walk-through) administered to class i;

O_i observation of post-test variable (questionnaire response) of class i;

---: experimental units (students) are not randomly assigned to groups (however, classrooms are randomly assigned to treatments).

Table 2 presents the descriptive and inferential statistics for the three classes on each questionnaire item. Analysis of variance (unequal sample sizes) was used to test for differences between the sections on the individual questionnaire items. Three one-sample t-tests were used to test if student responses to the questionnaire differed significantly from an 'undecided' response. An assumption was made that the population of students had a pretreatment mean response of '3' (undecided) to the questionnaire items. This seems reasonable since students entering the class did not have any previous experience with programming teams.

The results in Table 2 show no significant negative responses in any of the classes. Only 18·3% of the responses are not significantly different for a 'neutral' response. The following two items yielded nonsignificant results from all three classes:

3. Poor team members reduced the effectiveness of the teams.
18. This class will do better on the final examination than if the team approach had not been used.

The test for differences between the sections is concerned with the consistency of responses between the three classes. Only item 15 ('Students in this class felt the team approach was good') results in a significant difference in responses among the three classes (3·15, 3·74, and 3·58).

Discussion

This study presents empirical evidence that the team activities described in this paper elicit significant positive attitudes towards the structured walk-throughs, with no consequent significant negative attitudes. Based on the significant positive responses on 81·7% of the questionnaire items, it seems safe to state that the students were in favour of team activities. This is an important finding for two reasons. First, by polling all students, inferences about the attitudinal effects of the treatment have an empirical basis. All too often we read and hear about new pedagogical approaches without real objective evidence as to the effectiveness of these approaches. For example, many studies conclude with the assertion 'students seemed to enjoy this approach'. This type of data is of little use to instructors contemplating alternative pedagogical approaches. The instructor needs to have a feel for how a particular approach will be perceived by all students (good, fair, and poor).

The second reason the findings of this study are interesting is that the study suggested that the positive results are consistent among classes. Only item 15 (Students in this class felt the team approach was good) results in a significant difference in responses among the three classes. As shown in Table 2, this is probably the result of the statistically 'neutral' response of Class 1 versus the statistically 'positive' responses of Classes 4 and 5.

The data presented in this paper have coincided with my post-experiment experiences in teaching programming languages. Student verbal and written evaluations of the perceived effectiveness of the team activities have been overwhelmingly positive.

The evidence presented in this study strongly supports the incorporation of structured walk-throughs into the programming language class. However, more empirical research is needed in this area to determine if these results can be replicated independent of programming language taught, student population characteristics, or formality of the review process. The undertaking and publishing of such studies will serve to increase our knowledge of effective pedagogical approaches to computer programming.

Appendix 1 Perceived team effectiveness survey

	SA	*Ag*	*Und*	*Dis*	*SD*
1. Working in teams was fun.	.....	.....	.....	.....	.....
2. Working in teams gave students a better understanding of COBOL.	.....	.....	.....	.....	.....
3. Poor team members reduced the effectiveness of the teams.	.....	.....	.....	.....	.....
4. Students in the class made an honest effort to participate in the teams.	.....	.....	.....	.....	.....
5. Students did not take the team approach seriously.	.....	.....	.....	.....	.....
6. The team approach is a good way to learn programming.	.....	.....	.....	.....	.....
7. It would have been better not to use a team approach.	.....	.....	.....	.....	.....
8. Students in the teams didn't really do what the instructor had intended.	.....	.....	.....	.....	.....
9. The team approach should not be used next quarter.	.....	.....	.....	.....	.....
10. Students were upset when others found errors in their programs.	.....	.....	.....	.....	.....
11. The critiques were useful in getting programs to run.	.....	.....	.....	.....	.....
12. Students would have learned more if the time taken in class for team work had been used for lecturing.	.....	.....	.....	.....	.....
13. Students in this class followed the instructor's directions in preparing for the team activities.	.....	.....	.....	.....	.....
14. Critiquing someone elses program makes it easier to do your own programs.	.....	.....	.....	.....	.....
15. Students in this class felt the team approach was good.	.....	.....	.....	.....	.....
16. A person can get good ideas looking at someone elses program.	.....	.....	.....	.....	.....
17. Most students simply copied each others programs.	.....	.....	.....	.....	.....

Appendix *continued*

18. This class will do better on the final exam than if the team approach had not been used. ....
19. Reading a poor students' program was still beneficial since a lot of errors could be found. ....
20. Students appreciated the critiques of their programs. ....

References

CAMPBELL, D. T. and STANLEY, J. C. (1963). Experimental and quasi-experimental designs for research on teaching, *Handbook of Research and Teaching*, N. L. Gage (ed.), Rand McNally and Co.

FAGAN, M. E. (1976). Design and code inspections to reduce errors in program development, *IBM Systems Journal*, Vol. 15, pp. 182-211.

HUGHES, J. K. and MICHTON, J. I. (1977). *A Structured Approach to Programming*, Prentice-Hall, Englewood Cliffs, NJ, pp. 225-236.

JONES, L. (1974). Programming training appraisal a unique specialty, *Computerworld*, Vol. 8 No. 19.

LEMOS, R. S. (1978). Students' attitudes towards programming: the effects of structured walk-throughs, *Computers and Education*, Vol. 2 No. 14, pp. 301-306.

LEMOS, R. S. (1979a). Structured walk-thoughs and student ratings of faculty: effectiveness versus expediency, *Journal of Educational Data Processing*, Vol. 16 No. 1, pp. 1-9.

LEMOS, R. S. (1979b). An implementation of structured walk-throughs in teaching COBOL programming. *CACM*, Vol. 22 No. 6, pp. 335-340.

WEINBERG, G. M. (1971). *The Psychology of Computer Programming*, Van Nostrand Reinholt Co., NY.

YOURDON, Y. (1975). *Technique of Program Structure and Design*, Prentice-Hall, Englewood Cliffs, NJ, 74-77.

To the Editor
The Computer Journal

Sir

Using data base abstractions for logical design

I read with interest Jay-Louise Weldon's paper (*The Computer Journal*, Vol. 23, pp. 41-45) but was puzzled by the inclusion of the spurious 'cluster' entity. In the first example to use it, 'performer' is clearly a generalisation comprising two elements 'musical performer' and 'non-musical performer', each of which is itself a generalisation of two primitive objects.

The second example of the 'cluster' occurs in the decomposition of the category 'individual'. 'Individual is in fact an aggregate of the objects 'citizen ship' and 'university status' and not a generalisation at all. The term 'cluster' may be a convenient way of regarding data structures which correspond to no records but it is also confusing and certainly not required by the methodology. Indeed, as is shown in the paper, the aggregation and generalisation model is inadequate to deal with any but the simplest data base application, anchored as it is to a two-valued logic system. If data base abstraction is to be of general use in logical design then it must be accepted that complex semantics can only be dealt with in higher-valued logics.

For example the problem of relationships changing through time can be handled easily in three-valued logic but is insoluble in terms of the aggregation and generalisation model.

Yours faithfully,

S. N. JOHNSON

25 Stafford Street
Aberdeen
Scotland
22 February 1980

Dr. Weldon replies:

In data base abstraction a 'cluster' is used to classify similar objects according to some characteristic. It represents the inverse of the generalisation process. For example, performers can be classified into groups based on their type of talent (e.g. musical or non-musical performers) or on the basis of some other characteristic such as nationality (e.g. American performers, British performers, Chinese performers, etc.).

In the example given in the paper, 'individual' is indeed a generalisation of the lower level objects shown. 'Citizenship' and 'university status' are simply two characteristics that can be used to classify individuals. In either case individual is still a generalisation of the classes derived. For example, if we disregard university status the objects applicant, prospective student, student, and graduate can be generalised to the object individual. As such, individual would include attributes that are shared by all four groups, e.g. name, address, date of birth, etc. The four lower level objects would differ based on university status and each could include attributes which are not shared by the others, e.g. date of graduation for a graduate.

In addition to being a generalisation of those objects, individual is also an aggregate, but it is an aggregate of primitives, such as name, date of birth, and job title.

For a complete discussion of clusters, see the two articles by Smith (*CACM*, Vol. 20 No. 6 and *ACM TODS*, Vol. 2 No. 2) referenced in my paper

To the Editor
The Computer Journal

Sir

The unconditional branch—a modest proposal

Several years have now passed since Knuth's convincing demonstration (1974) of a number of common problems in programming which can best be solved by the controlled use of the **goto** statement. In that period, no general solution has been produced, and all major programming languages contain an explicit **goto** construct.

I do not wish to repeat here the arguments against the use of **goto's**; the case was well-stated originally (1968) and formal analysis of the semantics of the **goto** has only added weight to those arguments. I would simply propose two restrictions on **goto,** which could be implemented cheaply in any new language or new implementation.

1. Exactly one **goto** must be provided for each label in a program.
2. The **goto** must precede the label (i.e. only forward jumps are permitted).

The first restriction ensures that anyone reading a program can rapidly assure himself that he has found all potential paths through it, and eliminates needless fears about overlooked jumps, since for each label there would be one and only one **goto.** The second restriction ensures that, except for explicit loops, statement execution flows steadily down the pages of source listing, with a further increase in clarity and security. (I ignore the possibility of branching out of called procedures, as this is just an example of the wider dangers inherent in the use of global identifiers from within procedures, and is beyond the scope of this letter).

These restrictions could easily be incorporated as warnings in implementations of current languages, to provide support for improved programming style without affecting the portability of existing programs.

Yours faithfully,

MARTYN THOMA

South West Universities Computer Network
South West Universities Regional Computer Centre
University of Bath
Claverton Down
Bath BA2 7AY
21 May 1980

References

KNUTH, D. E. (1974). Structured Programming with goto statements, *Computing Surveys*, Vol. 6 No. 4.

DIJKSTRA, E. W. (1968). Goto Statement considered harmful, *CACM*, Vol. 3, pp. 147-148, 538, 541.

To the Editor
The Computer Journal

Sir

Using data base abstractions for logical design

Jay-Louise Weldon (1980) reports on an application of the relational methods of Smith and Smith (1977a, 1977b) and draws attention to some secondary shortcomings of those methods. This department of Honeywell is concerned with the development of network models using the role extension. We endorse the concept of generalisation and aggregation although we tend to work from the general to the specific, using the word 'includes' when we are specialising an abstraction.

We believe that the difficulties described are precisely because the role aspect was not recognised, and thus the form of the defined entity types was uniform and permanent. **Table 1** is a fragmentary schema definition defining just those things which caused some difficulty, and **Fig. 1** is the corresponding role diagram. The following brief remarks are meant to clarify those illustrations:

1. An entity is a tuple of attributes. It has at least one (identity) role of the same name.
2. A network is a graph composed of entities as nodes and owner-membership projections as edges. The explicit network relieves the necessity for clusters and reference by keys.
3. Roles are the partitioning of the nodes (entity tuples) into disjoint tuples, each having certain owner and member capabilities. Role types may be shared by different entities and represent the same capabilities and tuple type wherever they occur. They may be essential or optional.
4. Projections (sets) have a syntactic name according to their terminal roles, and they may also have a semantic name. Thus: 'role → role{ESEMANTIC}'.
5. 'dependence' is a precedence relation between roles.

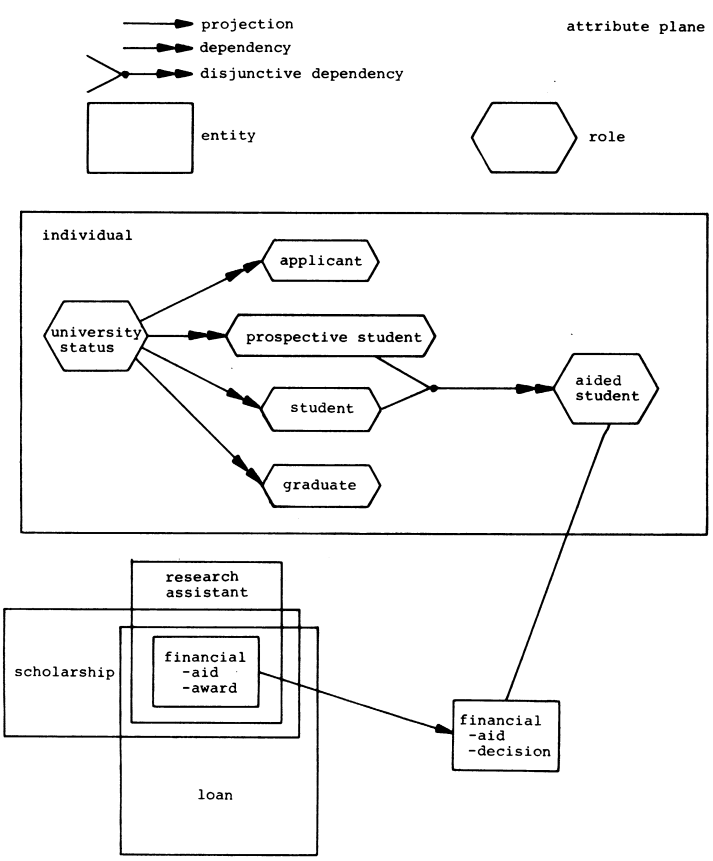

Fig. 1 Part of University schema

Table 1

```
schema University
domain citizenship enumeration 7
        value US Foreign
domain status enumeration 10
        value full-time part-time
domain types enumeration 11
        value loan scholarship assistant
domain number decimal 8
domain names char 32
entity individual
    natural citizen citizenship
role university-status
    essential individual
    primary_natural id:number
role applicant
    non_essential individual
    dependent university-status
    precludes prospective-student student graduate
role prospective-student
    non_essential individual
    dependent university-status
        derive id university-status.id
    precludes applicant student graduate
    natural status:status
role student
    non_essential individual
    dependent university-status
        derive id university-status.id
    precludes applicant prospective student graduate
role graduate
    non_essential individual
    dependent university-status
    precludes applicant prospective-student student
role aided student
    non_essential individual
    dependent prospective-student + student
        primary_derive id prospective-student.id student.id
entity financial-aid-award
    primary_natural type:types
entity loan includes financial-aid-award
entity research-assistant includes financial-aid-award
entity scholarship includes financial-aid-award
entity financial-aid-decision
    member financial-aid-award → financial-aid-decision
        match type financial-aid-award.type
    member aided-student → financial-aid-decision
        match id aided-student.id
end University
```

6. 'precludes' is an exclusion relation between roles.
7. 'derive' is a virtual attribute arising from a dependency.
8. 'match' is a key used to find an owner at instantiation time.
9. 'primary-natural' and 'natural' (non primary) are real attributes.
10. Indenting of statements in the schema definition implies that some context is inherited from less indented statements.

It can be seen exactly how financial-aid-awards are related to certain kinds of individual, and what the prerequisites are.

Not all the difficulties are solved by this approach. There are still various ways to model something, not just one way. We have no auditing facility to record the past and we have no explicit role transition directives. We do have state domains with constrained transitions between allowable values, and we see now that there may be a requirement for 'state' roles as well.

Yours faithfully

R. REEVES

Advanced Systems Engineering
Honeywell Information Systems Inc.
300 Concord Road
Billerica
Massachusetts 01821
USA

10 June 1980

References

WELDON, JAY-LOUISE. (1980). Using data base abstractions for logical design, *The Computer Journal*, Vol. 23 No. 1, pp. 41-45.

SMITH, J. M. and SMITH, D. C. P. (1977a). Database Abstractions: Aggregation, *CACM*, Vol. 20 No. 6, pp. 405-413.

SMITH, J. M. and SMITH, D. C. P. (1977b). Database Abstractions: Aggregation and Generalization, *ACM Transactions on Database Systems*, Vol. 2 No. 2, pp. 105-133.

BACHMAN, C. W. (1980). The Role Data Model approach to data structures, *Proceedings International Conference on Data Bases*, Aberdeen, Herpen/BCS Workshop Series, pp. 1-18.

To the Editor
The Computer Journal

Sir,

With reference to the recent paper 'An efficient predictor-corrector algorithm' by D. Westreich, published in *The Computer Journal*, Vol. 23 no. 2, I would like to make the following remarks:

1. To correct misprints in the published algorithm, step 4 should read 'Let I = $-$I' and step 9 should read 'Compute $y'_{t+1} = F(t_{t+1}, \bar{y}_{t+1})$'.
2. The use of the standard test equation $\frac{dy}{dt} = -\lambda y$, with λ real and positive, to examine the behaviour of the algorithm shows that
   (*a*) the computed values $y_2, y_4, y_6, \ldots$ exhibit fourth order convergence, but the values of $y_1, y_3, y_5 \ldots$ have only second order convergence
   (*b*) the method is stable when used with a step size h, provided $0 \leqslant h\lambda \leqslant 1{\cdot}240$. There is also an extremely small interval $2 \leqslant h\lambda \leqslant 2{\cdot}165$ for which the method is stable as well.

Yours sincerely,

J. SNELL

Department of Computer Science
The City University
Northampton Square
London EC1V 0HB
16 July 1980

To the Editor
The Computer Journal

Sir,

Procedure calling and structured architecture

The paper by Bishop and Barron (*The Computer Journal*, Vol. 23 No. 2) states that ICL's 2900 system programming language, S3, is faced with a procedure calling sequence as long as that given for a particular Pascal implementation. This is not the case.

It is true that, like the Pascal implementation, S3 keeps a copy of the display elements in the local data frame. However there are four important differences:

1. Each display element is only a single word holding an address rather than a full descriptor.
2. A separate static chain is not maintained since this would only duplicate the chain already formed by the first element of each display.
3. On procedure entry all the display elements are copied from the display of the statically enclosing procedure. The display copy is held between the last parameter and the first local data item. The first element of the local display, the static chain pointer, is passed as a parameter in the precall sequence.
4. Unlike the Pascal implementation a local display copy of an S3 procedure contains elements not needed by this procedure. However, elements for the nearest n levels are omitted if neither this procedure nor its containted procedures require them.

The code sequences involved for S3 procedures are:

Previous entry (level m)				*Bytes*	
$m <= 2$:	no code				
$m > 2$:	LXN	(LNB + base$_{m-1}$)		2	
	LSS/		repeated		
	LSD/	(XNB + base$_{m-2}$)	as	4	
	LSQ		necessary		
	ST	TOS		2	[intermediate instructions are combined into stack-and-load SLSS etc.]

Precall (level m calling level n)			
	PRCL	4	2
	(i.e.	STLN TOS	
		ASF 4 as in paper)	
	.... parameters .... (ending ST TOS)		
$n = 1$:	no code		0
$m < n$:	STLN	TOS	2
$m >= n$:	SLSS	(LNB + base$_{n-1}$)	0 [replaces the ST TOS at the
	ST	TOS	2 end of the parameters]

Call			
	RALN	5 or 6	2
	CALL	proc	4

Exit		
EXIT		2

Access (to item at level n)			
LXN	(LNB + base$_n$)		2 or 0 [may be set up
	access via (XNB + item)	+2	already]

Note that this access method uses XNB in the intended manner (XNB = Extra Name Base—i.e. extra to LNB, the Local Name Base).

Comparison of maintenance and access costs

Table 4 of the paper is reworked to include the 2900 S3 figures. The 2900 Pascal figures have been modified to take account of the PRCL instruction.

Notes

*Extra 1 instruction and 2 bytes if there are no parameters and $m >= n$. (level m calling level n).

†Less 2 bytes if XNB already set up.

	First call		*Other calls*		*Nonlocal access Overhead*	
	Instr	*Bytes*	*Instr*	*Bytes*	*Instr*	*Bytes*
B6700	5	11	4	5	0	0
2900 Pascal	$11 + p + q$	$24 + 2p + 2q$	7	14	1	4
2900 S3 $p = 0$	4	10	4	10	—	—
$p = 1$	5	12	5	12	1	4 ††
$p > 1$	$7 + \frac{p}{4}$‡	$16 + 4\frac{p}{4}$*‡	5	12*	1	4 ††
1900 subr.	$5 + p$	$18 + 3p$	$5 + p$	$18 + 3p$	p	$3p$
1900 inline	$12 + p$	$36 + 3p$	$12 + p$	$36 + 3p$	p	$3p$

‡ $\frac{p}{4}$ is rounded to the nearest integer.

§ $p = n - 1$, for S3 decremented by the number of immediately enclosing unaccessed levels (see above), i.e. the value of p for S3 may be smaller than that relevant to Pascal implementation. q = number of levels accessed ($< = p$).

Nothing in the above strategy is particular to S3 and it could be used for any block structured language including Pascal. Therefore we conclude that the above data for S3 are a better comparison of 2900 with the other machines.

There is almost no difference between S3 and Pascal in the overhead of access. Obviously B6700's hardware display gives the expected advantage in access to outer level data.

However, the difference in the cost of procedure call between the Pascal and S3 implementations is substantial. The number of instructions executed by 2900 S3 is closer to the number executed by B6700 than the number executed by 2900 Pascal. In some important cases 2900 appears to equal or even better B6700.

Yours faithfully,

D. K. MESSHAM and A. ELLIOTT

International Computers Limited
Westfields, Kidsgrove, Stoke-on-Trent ST7 1TL
24 June 1980

Erratum

We apologise to Dr Westreich that errors crept into his paper 'An efficient predictor—corrector algorithm', published in the May 1980 issue of the Journal, at the typographical stage. We are therefore reprinting the algorithm and complete references.

We take this opportunity also to publish Dr Westreich's up to date address. He is now at Department 3605, Israel Aircraft Industries, Ben-Gurion Airport, Israel.

The algorithm

Consider the system of first order differential equations

$$y' = F(t, y) \tag{1}$$

with initial value

$$y(t_0) = y_0$$

where y is an n vector to be determined and $F(t, y)$ is a continuous n vector function of t and y. To find the solution at t_f

1. Choose a step size h.
2. Find the solution y_1 to the equation at $t_1 = t_0 + h$ (say by a Runge-Kutta algorithm).
3. Set $i = 1$, $t_{i+1} = t_1 + h$, $y_0' = F(t_0, y_0)$
   $y_1' = F(t_1, y_1)$ and $I = 1$
4. Let $I = -1$
5. Use the predictor
   $$\bar{y}_{i+1} = y_{i-1} + 2h\, y_i'$$
   to obtain an approximation to the solution at $t_{i+1} = t_0 + (i + 1)h$.
6. Compute $\bar{y}_{i+1}' = F(t_{i+1}, \bar{y}_{i+1})$
7. If $I = -1$ go to step 12 otherwise go to step 8.
8. Use the corrector
   $$\bar{y}_{i+1} = y_i + \cdot 5h(y_i' + \bar{y}_{i+1}')$$
   to find an approximate solution at t_{i+1}.
9. Compute $y_{i+1}' = F(t_{i+1}, \bar{y}_{i+1})$
10. Use the corrector of step 8 again to obtain
    $$y_{i+1} = y_i + \cdot 5h\,(y_i' + y_{i+1}')$$
11. Go to step 14
12. Use the corrector
    $$y_{i+1} = y_{i-1} + h(y_{i-1}' + 4y_i' + \bar{y}_{i+1}')/3$$
13. Compute
    $$y_{i+1}' = F(t_{i+1}, y_{i+1})$$
14. If $t_{i+1} = t_f$ we stop, otherwise we let $i = i + 1$ and return to step 4.

References

KERSHAW, D. (1974). Volterra Equation of the Second Kind, in *Numerical Solution of Integral Equations*, L. M. Delves and J. Walsh, eds, Clarendon Press, Oxford, pp. 140-161.

WESTREICH, D. and CAHLON, B. (1979). Solution of Volterra Integral Equations and Differential Equation with Continuous or Discontinuous Terms, Math. Tech. Report, Ben-Gurion University, to appear.

Book review

Computing Principles and Techniques by B. L. Vickery, 1979; 182 pages. (*Adam Hilger*, £11·95)

As the author explains, the aim of the book is to provide a basic introduction to computing principles. This would seem to be a worthwhile objective as the greatest difficulty in understanding a new subject is often to understand the jargon. What this book provides, at one level, is almost a dictionary of common computer terms. This is particularly useful, for while the author deliberately avoids the use of jargon whenever possible, the same is certainly not true of the computer world. In fact the inverse is true. Another pleasing feature of the book is the section dealing with computer arithmetic. Presumably the beginner will be vaguely aware that computers use binary arithmetic, but such terms as hexadecimal or octal may be far less clear. Examples are also given which show how numbers of one base can be converted to another. It could be argued that the average user of a large computer system may never need to know such details, but this is certainly not true of the micro or even the minicomputer user.

The next two chapters cover the first elements of programming. The topic is introduced at the level of operation codes and memory addressing and leads on to consider mnemonic programming. The subject of bit manipulation is also discussed and some very basic concepts in Boolean algebra are mentioned. This section highlights one of the fundamental weaknesses of a book of this type, i.e. the desire to introduce a whole range of topics like Boolean algebra, which can only be touched on in the most superficial way. The next section covers the basic concepts of communicating with a computer from a remote device like a terminal. The common problems involved in an operation of this type are discussed, e.g. the type of character codes which are in common use. To this end one of the appendices provides a useful comparative table for octal, decimal and ASCII.

The last three chapters which discuss the general concepts of computing cover higher level operations. Another important topic discussed in this section of the book is that of errors. Mastery of the various types of error which are bound to occur in any program is essential particularly for the beginner.

The final chapter covers certain aspects of medical computing. Although interesting, the examples chosen are somewhat specific and in practice the programming problems involved are rather complex. The reason for including the final chapter was, presumably, to justify the book's inclusion in a medical physics series. The author might have been better advised to extend the applications section considerably because the reader is likely to be interested in the medical applications of computing. Nevertheless, on balance the objective of the book is a good one and the author has produced a text which should go quite a long way towards meeting the needs of someone entering the field of medical computing.

R. I. KITNEY (London)