Titre: Title:	On an experimental tool for general software development
Auteurs: Authors:	Pierre N. Robillard
Date:	1981
Type:	Rapport / Report
Référence: Citation:	Robillard, P. N. (1981). On an experimental tool for general software development. (Technical Report n° EP-R-81-05). https://publications.polymtl.ca/6207/

Institution:	École Polytechnique de Montréal
Numéro de rapport: Report number:	EP-R-81-05
URL officiel: Official URL:	
Mention légale: Legal notice:	

DÉPARTEMENT DE GÉNIE ÉLECTRIQUE

SECTION COMMUNICATION, INFORMATIQUE

Rapport technique EP81-R-5

Classification: Library of Congress no ....

ON AN EXPERIMENTAL TOOL FOR GENERAL SOFTWARE DEVELOPMENT

par

Pierre N. Robillard
Professeur agrégé
Département de Génie Electrique

Janvier 1981

Ecole Polytechnique de Montréal

Campus de l'Université
de Montréal
Case postale 6079
Succursale 'A'
Montréal, Québec
H3C 3A7

ON AN EXPERIMENTAL TOOL FOR GENERAL SOFTWARE DEVELOPMENT

par

Pierre N. Robillard
Professeur agrégé
Département de Génie Electrique

Janvier 1981

ON AN EXPERIMENTAL TOOL FOR GENERAL SOFTWARE DEVELOPMENT

by

Pierre N. Robillard, ing. Ph.D
Ecole Polytechnique
Montreal, Canada

Abstract

An experimental automated software tool for both design and implementation is described. The tool links together the manager and the programmer in the development of a software project. The key elements of the methodology are a word-graphic type of communication called graphical structured flowchart. Examples of projects conducted under the Software Development Assisted by Computer Systems supervision are presented. Typical outputs and listings are reproduced.

1- INTRODUCTION

Development of a software system is considered successful if implementation is completed on schedule and performance fulfills specifications. To better monitor the progress of a project, software managers have identified six stages in a software development cycle. (1),(2).

Requirements analysis, specification, design, coding, testing, maintenance and operation.

However, each of these six stages is part of the same thinking process and they naturally have some mutual overlap. Considerable effort has been invested in recent years to improve both the creation and the writing of programs (3),(4),(5).

Furthermore, a number of software engineering techniques focus on improving the software development process rather than improving the software developed by the process (6),(7). In structured methodology, attention is focused on form and organization. In structured programming, the programming process is identified with a step-by-step expansion of mathematical functions into structures of logical subfunctions, carried out until we can easily perform the subfunctions in a programming language.

Can these two "structured activities" be blended?

The task to be performed at each level requires the same methodology: DIVIDE-TO-CONQUER. This allows the software engineer to concentrate on a problem at a given level of abstraction and to keep to a minimum the number of details which must be considered at any one time. The main difficulty stems from a lack of easily transferable procedures to aid in the processing from one level to the other.

Systematic procedures, that everyone involved in the software project can follow, are required so that each programmer can blend his contribution naturally with the whole. Finally, software engineers should be able to manage the project effectively within time and cost limits. (8)

This paper reports on the description of an experimental tool and its automation. The tool links together the manager and the programmer on the development of a software project: It simply provides a way for initiating feedback at every level. This tool is fully interactive, since the software system design is arrived at only after several trial and error attempts at every level of design.

The essence of an automatic programming system is that it assumes responsabilities otherwise borne by a human and thereby reduces the human task and makes it more manageable.

Software system developers need a change in their programming environment which will relieve them of at least some of the tasks that they must currently perform. It is the goal of automatic programming research to effect this change by transferring responsability for some segments of the programming process from the human programmer to an automated computer-based system.

This tool is called SDAC which stands for Software Development Assisted by Computer of "Système de Design Automatique et Centralisé".

2- COMMON LANGUAGE

One of the key elements of the methodology is the scheme used for communication between the manager and the programmer. Graphics, words and "codes" are widely accepted as the basic elements of communication. Graphics alone do not carry enough readily accessible information. Words and codes portray poorly the structure of the project. Programmers do not like the documentation task, and computing managers who are most responsible for the quality of software dislike programming details. On the other hand the unique property of word-graphics type of communication can satisfy the needs of both the manager and the programmer.

In SDAC, the graphic symbolism is exclusively designed to represent the structure of the process while the words are used to express the "activity" of the structure. Both go together. Words without graphics symbols are meaningless as are graphic symbols without words. We call this representation graphical structured flowchart. Table 1 stresses salient features of such a representation.

NOTATION THAT BRIDGES THE GAP BETWEEN "SOFT MAN"S NATIVE LANGUAGE AND COMPUTER LANGUAGE

1. EXTENSION OF THE THINKING PROCESS
2. CLEAR GRAPHICAL REPRESENTATION OF THE PROBLEM STRUCTURE PROGRAM
3. DEPICT FUNCTIONAL PROPERTIES OF A PROGRAM
4. DEAL WITH THE PROBLEM AT VARIOUS LEVELS OF ABSTRACTION
5. NOTATION AMENABLE TO THE STEPWISE REFINEMENT PROCESS
6. NO CONCERN WITH THE SYNTAX OF A SPECIFIC PROGRAMMING LANGUAGE
7. OPERATIONS ARE SPECIFIED AT ANY LEVEL OF COMPLEXITY
8. VERY FEW SYNTACTICAL RULES (TRAINING TIME ~ 1 HRS)
9. SAME COMMUNICATION CHANNEL FOR THE ANALYSIS, DESIGN CODING
10. CAN BE FULLY AUTOMATED

TABLE I

Papers are regularly published in graphical representation of control structure. (9)-(12). We do not claim that the representation presented here outperforms any existing one*. All the graphic symbols used are shown in Fig. 1.

C_i and E_i are words describing conditions and resulting actions respectively. A condition (C_i) should be specified in such a way that if it is true then a resulting action E_i will (90^o shift from incoming flow) be executed; otherwise (condition being false) the control resumes its flow.

After the action is executed, a dash indicates if the control is proceeding forward or backward. In the latter case the condition (C_i) is verified again and the action E_i is repeated until the condition becomes false. A straight branching is used to represent a non-trivial action. Such an action can be further refined as a subprogram or module later on.

Choice of control structures is an important issue and is subject to much debate. (14),(15) The structures presented in this paper are experimental and

*Note: This representation was first introduced by D. Cowan et al (13) and further developed by Lavoie, Tellier and the author at Ecole Polytechnique and by Graham at Waterloo, Canada.

more work is needed to specify the criteria for the selection of control structures.

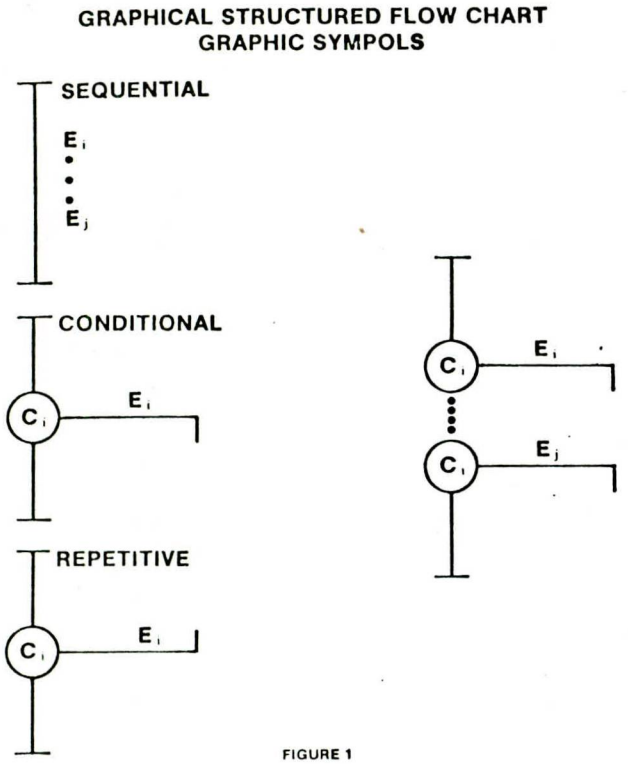


FIGURE 1

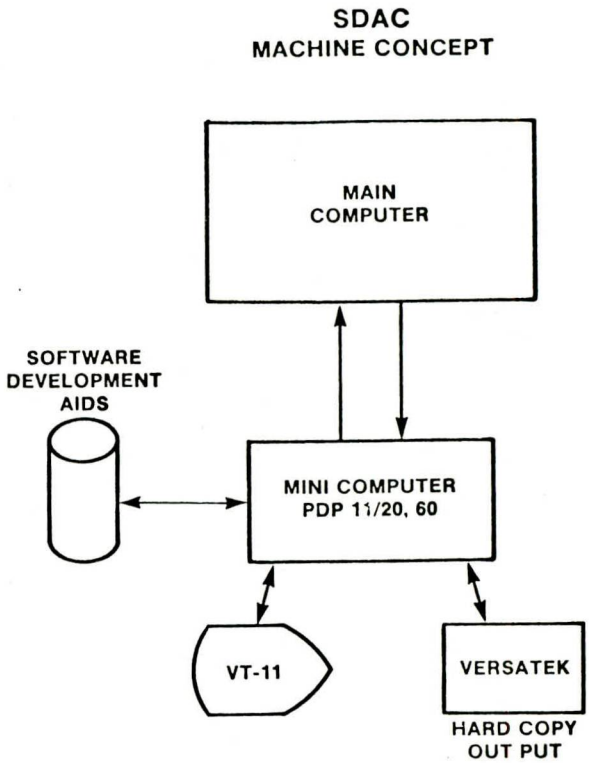


FIGURE 2

Words used to describe the conditions and the actions of the graphic control structure are parts of natural languages. No fixed rule is necessary for the analysis and design phase of a software project. Only in the final phase of implementation will the automated tool require the programmer to express any detailed action in a formal computer-like language.

3- SYSTEM DESCRIPTION WHAT IT IS

The overall structure of the SDAC system is shown in Fig. 2 SDAC is a software package running on a PDP 11/20. The VT-11 terminal makes it fully interactive. The primary task of SDAC is not to execute user programs but to assist users in the development, design and coding of software. It usually transmits the source program to the main computer (in our installation an IBM 360).

SDAC assists the software developer in defining and refining his problem. Although top-down design and functional decomposition are encouraged, the user is still free to carry out bottom-up design as far as possible. Alternatively, if one thinks of performing stepwise refinement, SDAC will assist in successively refining the solution into further details.

SDAC performs its task by keeping track of all the processes involved (whatever they are, bottom-up, top-down, stepwise refinement, or functional decomposition), and making sure that they are properly done and integrated.

Finally many programmers can work simultaneously, or in different time schedules on the same project. SDAC will integrate and log every effort such that the project manager can have at any time a complete record of the work being done.

SDAC is a multitask system in the sense that it:

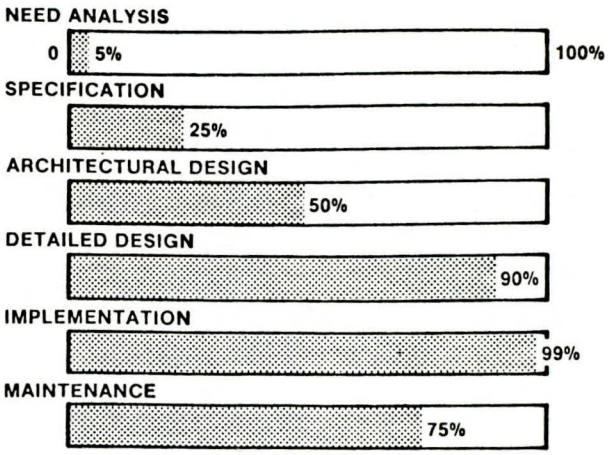
1. assists analysts in defining the problem
2. assists architectural designers in dividing and structuring the solution

3.	assists	programmers in building logic at modular levels It verifies every control structure
4.	assists	programmers in defining conditions and statements in the formal language of their choice
5.	creates	a source module in the host computer language
6.	integrates	automatically all refinements
7.	asks	the user to document all logical steps
8.	integrates	automatically all documentations to the source module
9.	rebuilds	a module after a modification, automatically
10.	provides	full security to the project. Only authorized tasks can be performed by authorized users.

Modest changes are made in the way programming is done. These changes are large enough so that programmers and managers are able to perceive the improvement, but not so large that they disrupt contemporary practice. The areas which are most improved are: standards for programming language usage including comments and control structures; documentation guidelines covering internal comments; team interaction patterns via graphical structured flowcharting and the programming environment involving the software tools. An estimate of human versus machine involvement in the cycle of software development is pictured in Fig. 3. Chaotic control structures are being replaced with more orderly and easier-to-follow high-level control structures. Clever programming tricks give way to comprehensible and modifiable strategies. Well-defined and functionally decomposed modules are the building blocks of big programs.

4- SYSTEM DESCRIPTION: HOW IT WORKS

We describe, in the following, how a project is typically conducted under SDAC supervision. Once the need or the task to be performed by software is perceived correctly, the systems analyst enters the main control structure at the terminal (VT-11).



MACHINE VS HUMAN INVOLMENT IN THE CYCLE OF SOFTWARE DEVELOPMENT.

FIGURE 3

Fig. 4 shows a typical CRT display (reverse video). This structure illustrates a looping structure (i.e. monitoring real time operations, data-validation, etc.).

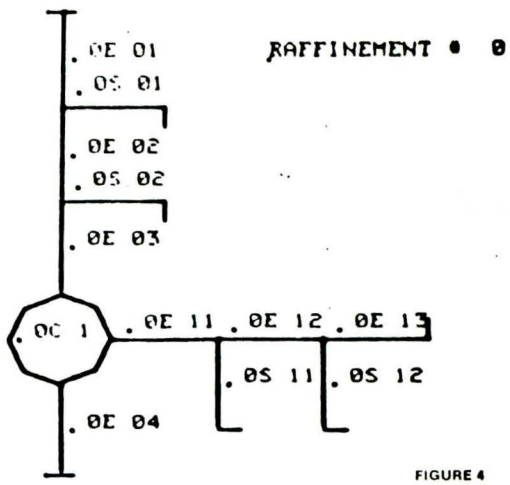


FIGURE 4

SDAC automatically codes every segment of the structure. This first step is called "raffinement #0" the French word for refinement. The first "raffinement" or any subsequent refinement can contain up to 10 structures. This limit is not restrictive; it appears rather as a way to force top-down approach. Moreover George Miller's (16) classic paper "The Magical number seven...plus or minus two" described a number of

experiments which suggested that information perceived by any sensory organ was limited to seven units.

The selected code for labelling is straightforward. The first number refers to refinement level. The second decimal number following the letter indicates to which condition (C) the action (E) belongs (first digit) and the second digit counts the number of actions associates to the same condition. (S) stands for SUBROUTINE.

SDAC asks the user to identify each of the label segments. He does so by writing a comment in a natural language up to 70 characters long. These will be automatically integrated later on as comments in the formal language program.

The user can go one step further by refining one branch. In Fig. 5 refinement is done on branch 0E 13. Raffinement #1 begins and finishes with label 0E 13, which means that all structures just defined are part of that branch. SDAC will ask for identification of every newly defined branch condition and subroutine.

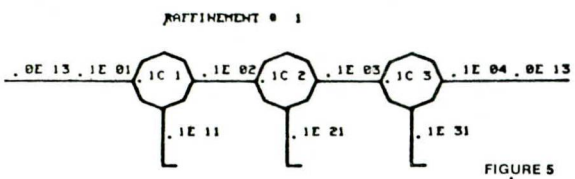


FIGURE 5

The process can go as far down as required by the user (see Fig. 6): At every level of refinement, those involved in the project can discuss the proper choice of structure, its description and the way to refine it. After the desired level is reached by the users, SDAC will automatically integrate all the steps and provide a complete drawing of the process. Fig. 7 (see at the end of the paper) illustrates a partial output of the development. All the conditions are listed separately.

User can zoom in on any structure, or group of

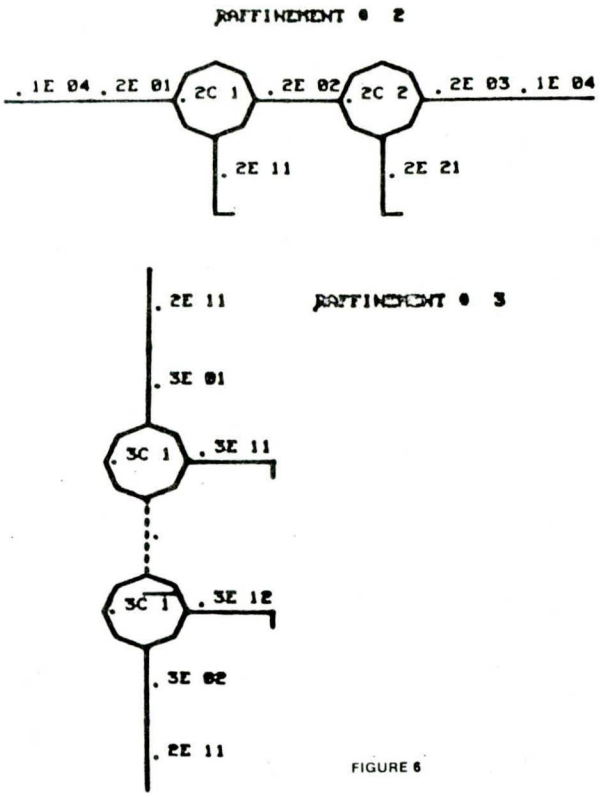


FIGURE 6

tures he wants, and redraw or modify it. All modifications on structures, or comments, will automatically be integrated to the whole. When the process is pursued to the code level a structured listing is provided in the selected language. Fig. 8 shows a standard Fortran example. The labelling and paragraphing are done automatically. The listing is associated with Fig. 4,5,6. The listing of Fig. 7 is too big to be presented here).

SDAC provides security features. All the programs including graphical structured flow chart, comments and codes are secured on a diskette. This media offers fully controlled access. Only authorized users have access to the diskette and the programs. Any modification performed on the software will be automatically documented and logged by the system.

A real advantage provided by SDAC is that every program developed has a unique up-to-date documention and listing. It is physically impossible un-

der normal operations to get access to the software without full control of SDAC.

```
C THIS PROGRAM COMPILES A PSF/3 PROGRAM
      COMMON/SOURCE/CARD
      CHARACTER*1 CARD(80)
      COMMON/CURSOR/IP,STRNO,COLUMN
      INTEGER IP,STRNO,COLUMN
      COMMON/LIFO/STACK,TOP
      INTEGER STACK(20),TOP
C     INSERT HERE OTHER IDENTIFICATIONS
C
C     INITIALIZE DATA SET
      CALL  INIT
C     READ AND PRINT A PSF/3 CARD
      CALL  GETPUT
20000 CONTINUE
      IF(CARD(1).NE.'/')GOTO 20001
         GOTO 20002
20001    CONTINUE
C        READ AND PRINT NEXT CARD
         CALL  GETPUT
C        COMPILE ASSIGNMENT,PRINT,WHILE...DO,END WHILE OR RETURN
C        ACCORDING TO LENGTH AND FIRST LETTER OF FIRST WORD
         CALL  SCANWD(FIRST,LENGTH)
         IF(LENGTH.EQ.5.AND.FIRST.EQ.'P')GOTO 20003
            GOTO 20004
20003       CONTINUE
C           INSERT HERE STATEMENTS TO COMPILE PRINT
20004       CONTINUE
         IF(LENGTH.EQ.1)GOTO 20005
            GOTO 20006
20005       CONTINUE
C           INSERT HERE STATEMENTS TO COMPILE ASSIGNMENT
20006       CONTINUE
         IF(LENGTH.EQ.5.AND.FIRST.EQ.'W')GOTO 20007
            GOTO 20008
20007       CONTINUE
C           INSERT HERE STATEMENTS TO COMPILE WHILE
20008       CONTINUE
         IF(LENGTH.EQ.3.AND.FIRST.EQ.'E')GOTO 20009
            GOTO 20010
20009       CONTINUE
C           SEE IF STACK HOLDS LOCATIONS OF ONE OR MORE
C           WHILE...DO'S , IF  SO, COMPILE END WHILE; OTHERWISE
C           IGNORE FINAL END OF PROGRAM
            IF(TOP.GT.0)GOTO 20011
               GOTO 20012
20011          CONTINUE
C              INSERT HERE STATEMENTS TO COMPILE END OF WHILE
               GOTO 20013
20012       CONTINUE
C              INSERT HERE STATEMENTS TO IGNORE END OF PROGRAM
20013          CONTINUE
20010       CONTINUE
         IF(LENGTH.EQ.6.AND.FIRST.EQ.'R')GOTO 20014
            GOTO 20015
20014       CONTINUE
C           INSERT HERE STATEMENTS TO COMPILE RETURN
20015       CONTINUE
      GOTO 20000
20002 CONTINUE
      RETURN
      END
```

Figure 8

5- CONCLUDING REMARKS

Our research directed toward automating the software development process is based on the knowledge-based system approach, and the very high level language (VHLL) approach. VHLL enables the programmer to specify and organize his program within a problem-oriented framework. This task is achieved with a simple schematic notation which represents salient features of the system's requirements.

The knowledge-based approach interacts with the user, who may not be a programming specialist. The user expresses his desires in the natural language of his field of interest. Through top-down approach, SDAC assumes complete responsibility for the synthesis of a software system.

The size of the programs that the existing prototype is capable of synthesizing is not small (3000 coded statements and up). The kinds of program structures and organizations that SDAC can actually construct are limited to simple and highly regular structures. SDAC is far from being the complete answer: it is still under development. However, research on an advanced schematic notation is in progress. Preliminary results have shown that more complex structures can be implemented. (double exit loop, select etc).

We believe that the greatest potential application of SDAC is its specification language. It is an easy to use language for clearly and unambiguously describing systems both to machines and to people.

The benefits of what we have proposed are both tangible and intangible. Documentation savings alone can justify the cost of development. Further savings will be accrued through reduced testing efforts, shorter development cycles, and increased productivity.

Software system design requires, beyond documentation, preciseness, consistency, completeness and communicability: SDAC is the enforcer of these needs and its method of enforcement is computer automation.

It offers aid to the Project administrator. Documentation, source code, object code, design data, and test information are all kept under configuration control.

Users of any method are reminded that method alone is not sufficient. Standards and automated aids and procedures for their application are vital for the achievement and retention of quality, throughout a system life cycle.

ACKNOWLEDGEMENTS

Research done with partial support form CRSNG grant No. A-0141. The authors wish to thank everyone who has contributed to the development of the tool

and in particular Mr Tan Phalkun, who did most of the programming.

REFERENCES

1. Zelkowitz, M.V. "Perspectives on Software Engineering", Computing Surveys, Vol. 10, No. 2, June 1975.

2. Boehm, B.W. "Software Engineering", IEEE Transactions on Computers Vol. C-25, No. 12, Dec. 1976.

3. Bergland, G.D. and Gordon, R.D. "Tutorial: Software Design Strategies", IEEE Computer Society,Catalog No. EHO 149-5.

4. Freeman, P. and Wasserman, A.I. "Tutorial on Software Design Techniques" IEEE Computer Society,Catalog No. 76CH1145-2G.

5. Basili, V.R. and Bakert "Structured Programming" IEEE Computer Society,Catalog No. 75CH10 49-6.

6. Reifer, D.J. "Tutorial: Software Management" IEEE Computer Society, IEEE Catalog No. EHO 146-1.

7. Miller, E. "Tutorial: Automated Tools for Software Engineering" IEEE Computer Society, IEEE Catalog No. EHO 150-3.

8. Lehman, J.H. "How Software Projects are really Managed" Datamation January 1979, pp. 119-129.

9. Misra, J. "A Technique of Algorithm Construction on Sequences" IEEE Transactions on software Engineering, Vol. SE-4, No. 1, January 1978.

10. Williams, M.H. "Generating Structured Flow Diagrams: The nature of unstructuredness" The Computer Journal, Vol. 20, No. 1.

11. Lindsey, G.H. "Structure charts a Structured Alternative to Flowcharts" Siplan Notices, Nov. 1977.

12. Witty, R. "Dimensional Flowcharting" Software Practice and Experience, Vol. 7, 553-584, 1977.

13. Cowan, D.D., Dirksen, P.H., Graham, J.W. "An Introduction to COBOL with WATBOL" A structural Programming Approach, WATFAC, Waterloo, Canada, 1976.

14. Leavenworth, B.N. (ed) "Control Structures in Programming languages" Siplan Notices, 7, 11, Nov. 1972.

15. Ledgard, Henry "A Generalogy of Control Structures Com. ACM, 18, 11, Nov. 1975.

16. Miller, George A. "The Magical Number Seven; plus or minus two: Some limits on our capacity for processing information" Psychological Review, 63, 1956, 81-97.

BIOGRAPHY

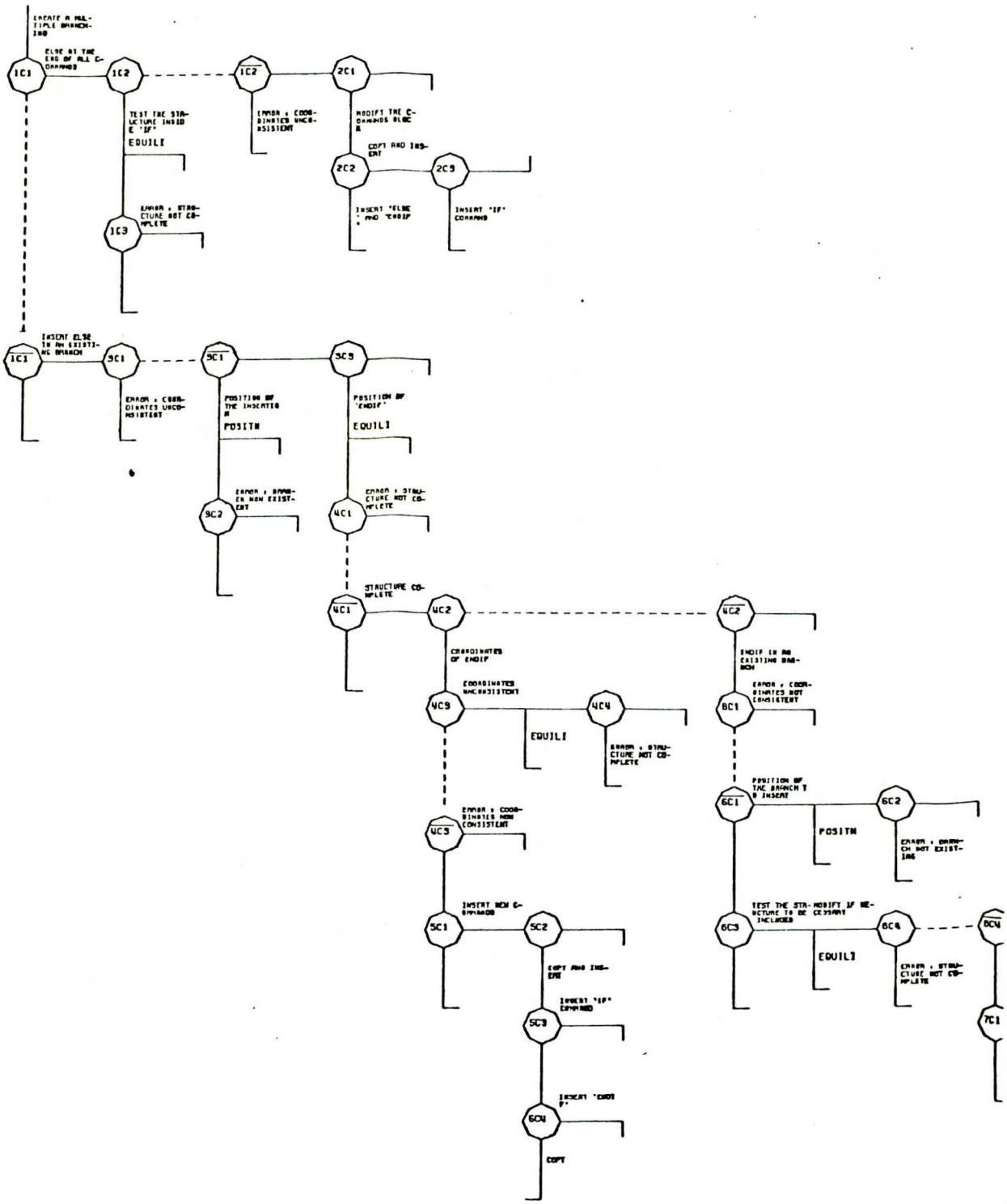
Pierre N. Robillard was born in Sherbrooke, Canada on May 5, 1948.

He received the B.Sc. degree in physics from the University of Montreal, and the M.Sc. (physics) and the M.A.Sc. (computer sciences) both from University of Toronto, Canada, and the Ph.D. degree in Electrical Engineering from the Université Laval, Quebec, Canada.

He is currently an Associate Professor of Computer Science at the Ecole Polytechnique at the Université de Montréal, Montreal, Canada.

He has published papers on a wide variety of topics requiring large software development: High energy physics, computer communication Networks, brain research and tool development. He has been a consultant to branches of Canadian Governments and to Universities.

His principal interests are actually in Software Engineering and tool development for software management.



PARTIAL OUTPUT

Figure 8