Production Systems: A Formal Notation for Defining The Syntax and Translation of Programming Languages.
H. F. Ledgard.
University of Massachusetts, Amherst. Dept. of Computer Science and Information. Aug 75, COINS Technical Report 75A-4

This paper presents the formalism of Production Systems and investigates its application to define the complete syntax of a computer language and its translation into a target language. Several properties appear well-suited to this task: (a) the formalism can be used to specify exactly the syntax of a computer language, including context-sensitive requirements. (b) the specification of the context-sensitive requirements on syntax can be isolated from the context-free requirements. (c) the same formalism can be used to specify the translation of one language into another (d) the notation has been reworked many times to promote ease of readability. The following example applications of production systems are given: (a) a specification of the complete syntax of a small but difficult subset of PL/I. (b) a specification of the translation of lambda-calculus expressions into normal form.

technical contributions

STRUCTURED DOCUMENTATION

by

Daniel M. Berry†
University of California, Los Angeles
Computer Science Department
Los Angeles, California 90024

Introduction

In current studies of software engineering, all aspects of the program construction process are coming under closer scrutiny; these include the processes of problem inception and specification [Lis72], project management [Bak72, Mil72, Tsi73], program feature selection [DDH72, Wul72, WS73, Knu74, LZ73, Hol73, HW73], proving program correctness [Hoa69, Wir72, Mil72, Mil72a], and program documentation [Kat71, LSD, Goo73].

†This research was supported by the U.S. Energy Research and Development Administration, Contract No. AT(04-3)-34, PA 214.

This paper gives some personal observations on documentation style, top-down structured programming, and written descriptions of structured program development. From these observations, a suggestion for a "structured documentation" style is offered.

Documentation

It has been long thought and taught that the proper program documentation consists of:

1) a verbal description of the intent and behavior of the program, including a statement of input and output (functional) requirements,
2) a flowchart of a slightly higher level than the program itself but (hopefully) accurately reflecting the program,
3) comments added to the program itself.

This documentation style will be referred to as the Verbal Description, Flowchart, and Comments (VDFC) style of documentation.

Structured Programming

The currently advocated programming style variously termed as structured programming, top-down programming, or step-wise refinement consists in successively refining ever more detailed descriptions of a program's intended behavior into a complete program implementing that behavior. In developing a program by structured programming, one starts with a functional description of a program in English or in programming language "statement" notation. If this "statement" is a bona-fide statement in the programming language being used to write the program, then the program is complete. Otherwise, the "statement" must be refined into more detailed "statements". The refined "statement" is said to be an abstraction of the more detailed "statements" obtained in the refinement step. At each step, if all as yet unrefined "statements" are bona-fide language statements, then the program is complete; otherwise, some unrefined "statement" is selected for refinement into constituent actions. At each refinement step, there is the obligation to prove that the combined intended behaviors of the constituent "statements" correctly implement the intended behavior of the refined "statement".

Additionally, it is required that in decomposing "statements" into constituent actions, that the constituent "statements" be composed by one of a few program structuring constructs. These constructs always include

sequencing of statements
if then else
while do

and almost always include

case selection
repeat until

as well as other simple one entry-one exit constructs. Others also permit use of limited disguised goto's to effect abnormal exits from the above listed constructs.

Observations

In the course of

1) teaching an intermediate level undergraduate course (E20 at UCLA) requiring VDFC documentation for programming assignments,
2) reading numerous published descriptions [DDH72, Wir71, Wir72, MK75] of structured programming applied to specific programming problems,
3) doing some structured programming on my own [Bry74],
4) teaching a graduate course in software engineering and structured programming in which students were required to apply structured programming to the development of a moderate length (about 600 PL/1 statements) program and to hand in a written description of the refinement process leading to the program with no additional documentation,

I have come to the following conclusions:

1) To me, the VDFC type of documentation is usually of marginal help in understanding the students' programs. The verbal descriptions are usually at some uniformly too high a level to convey any really useful detailed information about the program. The level of the flowcharts is usually too close to that of the program to convey any information not directly deducible from the program itself (also if the program is poorly structured so is the flowchart). The comments are usually too sparse and/or too cryptic to convey any helpful information. I usually end up having to hand-simulate the program for a while to determine what is going on. The problem is that all information given in the documentation is either too global or too local to give the proper perspective on the abstractions used in the programming process. I am forced to deduce the abstractions from my hand simulation. It is not clear whether this is the fault of inexperienced programmers not writing good enough VDFC documentation or the fault of the VDFC style in general, but it seems to be the latter because there is no discipline to force an exposition of the levels of abstraction.
2) It seems preposterous to me (and to others) that the programs described in the published descriptions of structured programming were developed as cleanly as was described in the papers; i.e., with no false starts, little backtracking up the abstraction tree, and with no inter-level thinking. It also seems incredible to me that the various optimizations mentioned in the papers were discovered in as structured a manner as described. My own experiences in doing structured programming make me skeptical. In doing several examples of structured programming, including the one described in [Bry74], I discovered that

a) I made many false starts, going down refinement paths whose stupidity I would be ashamed to admit;
b) I had to backtrack frequently from these inappropriate refinement paths;
c) I found myself thinking across several levels of refinement and across several independent paths of refinement (both of these are "cardinal" sins);
d) optimizations were discovered much too late to be included as a natural part of the refinement process.

In general, I bungled my way through the refinements. However, in writing the description of [Bry74], the refinement processes were presented as a logical orderly progression. False starts and the resulting backtracking were presented only if they were instructive. All thinking took place at the right level, and optimizations were "discovered" and introduced at precisely the right place. An absolutely beautiful well-structured document was produced for public consumption.

The program described in [Bry74] was assigned to the students of my graduate class. Some of them produced quite elegant descriptions. The ones that I talked to admitted to bungling their way through the structured programming but reporting a relatively clean structured development. This was also the case for the larger projects done by the class [Rem75, Ros74].

This observation should not be construed as claiming that I think that the authors of the published structured programming descriptions are falsely and misleadingly claiming that they developed their programs in as clean a manner as reported. In no paper is there such a claim. Also, as we shall see, it is totally irrelevant as to whether the final program development description is an accurate reporting of history.

3) In spite of the reservations expressed above (or, as we shall see later, because of these reservations), I have found more of the descriptions of structured programming to be more helpful in understanding the developed programs than the VDFC documentations. In most of the published descriptions and in a couple of the students' descriptions, the abstractions were well enough chosen, and the description of the refinement process was well enough written, that the program could be understood with a single careful reading of the refinement description.

The conclusion I draw from these observations is that the VDFC style of documenting a program should be replaced by a structured style of documentation in which the major documentation of a program is a well-written description of a refinement process yielding the program. This structured documentation should have the following properties:

1) The structured development should start with a simple functional description of the whole program.
2) The refinement process should yield a set of abstractions carefully chosen to maximize clarity of the description.
3) At each refinement step, some informal proof should be given of the correctness of the refinement. In practice, this involves

nothing for sequential or conditional composition and stating an invariant for loop composition.

4) The final program yielded should be the documented program.

Note, in particular, that the structured development described in the documentation should not necessarily be that actually taken to obtain the program, but rather one that will convey the most understanding.

When a program is modified in the course of natural maintenance, the documentation assists in identifying the exact point at which the refinement of a high level abstraction needs to be changed to yield the desired change. The affected parts of the documentation should be rewritten to show the development of the new parts. This results in a new complete structured documentation for the whole program.

This documentation style is by no means restricted to use only with those programs developed by step-wise refinement. Indeed, it might very well be worth the trouble to do an after-the-fact structured development of a non-structured program to discover the levels of abstraction that would lead to the given program. This refinement process could then form the basis of the documentation (also some unknown bugs may be found in the process).

REFERENCES

Bak72 Baker, F. T., "Chief Programmer Teams," IBM Systems Journal, 11:1 (1972).

Bry74 Berry, D. M., "An Example of Structured Programming," UCLA Computer Science Department Quarterly, 2:3 (July 1974).

DDH72 Dahl, O. J., E. W. Dijkstra, and C. A. R. Hoare, Structured Programming, New York: Academic Press, (1972).

Den73 Dennis, J. B., "The Design and Construction of Software Systems" and "Modularity" in Bauer, F. L. (Ed.), Advanced Course on Software Engineering, Berlin: Springer-Verlag, (1973).

Goo73 Goos, G., "Documentation," in Bauer, F. L. (Ed.), Advanced Course on Software Engineering, Berlin: Springer-Verlag, (1973).

Hol73 Holt, R. C., "Teaching the Fatal Disease or Introductory Programming Using PL/1," SIGPLAN Notices, 8:5, (May 1973).

HW73 Holt, R. C. and D. B. Wortman, "Structured Subsets of the PL/1 Language," Technical Report CSRG-27, University of Toronto, (October 1973).

Kat71 Katzenelson, J., "Documentation and the Management of a Software Project - A Case Study," Software - Practice and Experience, 1:2 (April-June 1971).

Knu74 Knuth, D. E., "Structured Programming with the Goto Statement," STAN-CS-74-416, Stanford University, (May 1974).

Lis72 Liskov, B., "A Design Methodology for Reliable Software Systems," Proceedings FJCC, (1972).

LSD "Documentation Guide for LSD Programs," Brown University, Division of Applied Mathematics (author and date unknown).

MK75 McGowan, C. L. and J. R. Kelly, Top Down Structured Programming, Auberbach, (1974).

Mil71 Mills, H. D., "Chief Programmer Teams: Principles and Procedures," IBM-FSD Report No. FSD71-5108, Gaithersburg, (June 1971).

Mil72 Mills, H. D., "Mathematical Foundations for Structured Programming," IBM-FSD Report No. FSG72-6012, Gaithersburg, (February 1972).

Mil72a Mills, H. D., "How to Write Correct Programs and Know It," IBM-FSD, Gaithersburg, (December 1972).

Par72 Parnas, D. L., "On the Criteria to be Used in Decomposing Systems into Modules," CACM 15:12, (December 1972).

Rem75 Rempe, B. E., Structured Programming of a Compiler for a Structured Language, M.S. Thesis, UCLA Computer Science Department, (1975).

Ros74 Rosenberg, P. A., Canfield Solitaire; an Example of Structured Programming, appendix to Internal Memorandum #135, UCLA Computer Science Department, (1974).

Tsi73 Tsichritzis, D., "Project Management," in Bauer, F. L. (Ed.) Advanced Course in Software Engineering, Berlin: Springer-Verlag, (1973).

Wir71 Wirth, N., "Program Development by Stepwise Refinement," CACM 14:4, (April 1971).

Wir73 Wirth, N., Systematic Programming: An Introduction, Englewood Cliffs: Prentice-Hall, (1973).

Wul72 Wulf, W. A., "Programming without the Goto," IFIP Proceedings, (1971).

WS73 Wulf, W. A. and M. Shaw, "Global Variables Considered Harmful," SIGPLAN Notices 8:2, (February 1973).