oughly understood and agreed before design, but causes impatience in some participants and management.

• Need one “friendly” group to try method first. Afterwards it’s better to “offer” the method to other groups than to try to “sell” it—hard sell doesn’t work.

• “Discipline” in use of method is very important: syntax, conventions, rules, author-commentator cycle, completion of a level of decomposition before going to next, each diagram must increase understanding, etc.

• Large amounts of paper generated. Project librarian must be assigned, as recommended by SofTech.

• Formal training is lengthy—two to three weeks full time—but necessary.

• Potential authors must have a project in mind during training and be assigned full-time to it immediately thereafter.

• Follow on assistance by a trained “monitor” is obligatory during initial application of method.

• Monitor must confine his attention to proper use of the method rather than become involved in the substance of the analysis.

• Training plus monitoring by SofTech is costly: 20 to 40 thousand dollars per course for up to ten authors, but worth it.

• Ideal team during SA phase for our type of large switching projects seems to be two or four persons each from systems, hardware and software.

Recent experience—1977

It is our estimate that about two-thirds of the value of SA/FP2 is in the SA part. Nevertheless, the availability of a follow on design method, in our case FP2, makes the application of SA easier because the SA authors are more ready to defer design considerations to the FP2 phases.

The development and acceptance of FP2 has proved extremely difficult and its success has not yet been fully demonstrated. We wanted to use the same basic precepts and syntax as in SA so that the designers, some of whom will have also participated in the analysis by SA, would not have to learn a new syntax and set of principles—we wanted a sort of “continuous” method, starting with a list of requirements and constraints and ending up with detailed coding specifications. This “continuity” and similarity of syntax is important to software maintenance personnel and will assist comprehension by interested customers.

This section concentrates on our 1977 experience with FP2; subsequent experience will be covered during the presentation at NCC 78.

• Underestimated importance of providing detailed guidelines for carrying out the Transfer phase.

• Initially called the output of the Transfer phase, the “process model.” “Process” has many meanings and caused great confusion. The neutral phrase “action group” conveys the intent without confusion.

• Software design still requires great skill, but FP2 permits easy comprehension after key design decisions have been made.

• FP2 criticized for not rendering high level software design “semi-automatic” or at least “semi-algorithmic.”

• Direct coding from flowgrams is in most cases straightforward and can be done by programmers other than the designers.

• The fallout of a “structured program” for each action has proved very appealing.

• Test plans can be developed throughout FP2 in increasing detail and related directly to diagrams.

• Much debugging is done by reference to flowgrams instead of listings.

REFERENCES


EXPERIENCE WITH AN APPLICATION OF STRUCTURED DESIGN—J. A. Rader

INTRODUCTION

The application of structured design to a 20 man year project which generated 100,000 lines of code is described. Included are a description of the project, productivity figures, and a discussion of strengths and weaknesses of the technique as practiced on the project.

THE APPLICATION

Introduction

The Computer Aided Design (CAD) Department in the Hughes Aerospace Groups contains about sixty employees. The department provides Computer-Aided Design/Test/Manufacturing software and services; it operates and supports a DEC system 10 computing facility; and it operates and maintains a Gerber photoplotter and several digitizers.

Most of the software is data manipulative in nature—files are read; fields are extracted from records and massaged; arrays are built, operated on and sorted; reports are generated; and data bases are accessed and updated. The primary language has been and continues to be FORTRAN. In addition, there is an extensive library of FORTRAN-callable assembly language routines to perform bit and character manipulations as well as other special functions. Where very heavy CPU utilization is expected, assembly language is also sometimes employed.
Conversion decision

Several years ago, a corporate decision was made to phase out the Honeywell G635, the computer on which the CAD System at that time ran. Among the many alternatives considered for rehosting the CAD System, the one finally chosen was to purchase a DEC system-10 and convert the CAD system to run on that computer. The primary reason for selecting the DEC-10 was a proven time-sharing capability.

The conversion from the G635 to the DEC-10 was a conversion only from the standpoint of function. Existing programs were inventoried and for each it was decided which would be converted essentially as is, which would be modified, and which would be discarded.

Goals and advanced plan

As we started to plan, we recognized that it was very important to firmly establish our goals, and to determine very specific milestones. Thus we would be able to measure our progress and to report on it to our management, who was picking up the tab.

Succinctly stated, the major goals were: (1) to create a unified system tied together by a central data base; (2) to create software that was reliable and maintainable; (3) to provide a user interface that was easy to use and that was consistent across all software; and (4) to proceed in a manner that allowed us to measure how well we were meeting schedules.

In late 1973, an advanced planning and development activity was formed. The advanced planning group was to define the overall structure of the system, and to specify the procedures to be followed in specifying and implementing the system.

In November of 1973, the author attended a 6-day in-plant seminar on structured design. This seminar was taught by Larry Constantine and proved to be of immeasurable value. The value arose not from revolutionary concepts but rather from a well reasoned and coherent discussion of the relevant concepts. The ultimate result of attendance was the generation and documentation of a methodology for practicing structured design in our organization. This methodology is described in the next section.

Outputs of advance group

In the approximately 18 months of its existence, the advance group produced 4 basic outputs. First, it produced a system concept. Second, it defined the standards and procedures to be followed in rewriting the system. These were documented in a Standards and Procedures Manual issued to all programmers. Third, it defined most of the applications support software. Fourth, it provided a test vehicle for the standards and procedures defined, and provided the first productivity figures for the methodology. These figures were used in estimating the effort required to implement the main body of CAD software.

RESULTS OF APPLYING STRUCTURED DESIGN

Productivity figures

The advanced planning activity implemented 336 modules of applications support software. These modules contained
Problems encountered

Although our experience with structured programming has been strongly positive, we did encounter some problems. Moreover, looking back, we see areas where improvement is needed.

The biggest need for improvement has to be in the area of specification. Once a good specification has been generated, things become very manageable. However, we have found it extremely difficult to write a specification which on the one hand a user can read and understand, and which on the other hand defines things well enough to allow design to begin in earnest.

A second problem area is related to one of the design goals. From the start it was impressed upon the programmers that they were to put design before efficiency. As a result a couple of activities were implemented which were very much more expensive to execute than they should have been. These subsequently, had to be modified for efficiency. In most cases this efficiency problem might well have been avoided if we more strictly followed one of our own published procedures. We did not require each module to be reviewed by another programmer as we said we would. The excuse tends to be "we just don't have the time", and is used by programmers and supervision alike. In the face of tight schedules, this excuse is not easily dismissed. This problem will doubtless be struggled with for some time to come.

A final problem is the difficulty in training enough programmers to be good designers. The training problem is particularly troublesome because there is no way to give years of experience, and the attendant design maturity, to even a highly capable junior individual. This is important because the structured design of a large process requires the judgment to make numerous decisions, which involve trade-offs between strict adherence to structured design principles, on the one hand, and effective use of human and machine resources on the other. The best solution is to employ the most qualified designers on critical designs, and to use less critical designs for training. But when schedule constraints force many important designs to overlap, less desirable compromises have to be made.

Summary of benefits

Although we did not have any solid prior productivity data, we definitely feel that structured design has improved productivity. This, however, was not a goal in our adopting structured design, but has been just a happy side-effect. What was anticipated were increased reliability, increased maintainability, and increased visibility.

There is no doubt that we have achieved increased visibility. Moreover subsequent experience with modifying structured programs convinces us that increased reliability and maintainability have been achieved.

It was found that structured design allowed us to make very good use of personnel. We have been able to really load up an activity in the module coding and checkout phases without introducing confusion. It has also been easy to quickly move a programmer from one activity to another, with little loss in effectiveness. Moreover, we have gotten excellent productivity from beginning programmers.

Only mild reluctance to adopt structured design techniques was manifested by the staff. Junior personnel adopted easily with no apparent "loss of individuality" response. There was some initial thrashing with senior personnel as we all strove to understand the implications and tradeoffs of modularity. For instance, not everyone accepted at first that structured design was indeed distinct from what they were already doing.

A final word on interpreting our experiences with structured design. We feel that our experience is unique to our application and our environment. A different application in a different environment might yield better or poorer results. Nonetheless, we are confident that, for most applications, structured design will yield more reliable and maintainable systems, while providing good visibility of the design and implementation processes.

BIBLIOGRAPHY