FORMAL DEFINITION AND DESIGN

Henry F. Ledgard
Andrew Singer

COINS Tech. Report 78-01

February 1978

This work was supported by the U.S. Army Research Office and the National Science Foundation.

ABSTRACT

This paper discusses some of the issues resulting from extensive experience with formal definition and makes the case that the right place for formal definition is during the *design* process.

"Another grave difficulty was the utter impossibility of remembering all the variety of motions brought simultaneously into action. Obviously, nothing but the most complete harmony and precise order among such a system of movements could avoid an obstruction among the parts, and he soon found the need for some aid to the memory, as well as the ability to see what every moving part of machinery was doing at each instant of time. This formidible obstacle, which might have tried the courage or baffled the ingenuity of most men, proved only an incentive to further effort on his part, for he realized that unless he could devise some means of adequate pre-vision, he would have to abandon the scheme altogether.

> *'He soon felt that the signs of ordinary language were far too diffuse to admit of any expectations of removing the difficulty and being convinced from experience of the vast power which derives from the great condensation of meaning in the language it employs, he was not long in deciding that the most valuable path to pursue was to have recourse to the language of signs.'*

This system he called the mechanical notation, and it was applicable to all machinery. It consisted of an arrangement of signs which enabled him, by directing his eye along a line, to trace the motion of every piece of the machine from affect to cause. The signs included the Arabic and Roman alphabets, capital letters, lower case letters, letters in italics, and dotted and broken lines ..."

Irasible Genius
The Life of Charles Babadge
Maboth Moseley

1. INTRODUCTION

The reader may recall the "Sampler of Formal Definitions" [1]. In this paper, four different formal definition techniques were applied to the definition of a small language, ASPLE. Like many formal definitions, these were written after the language had been completely developed: for ASPLE this meant an already published subset of ALGOL 68.

2. AN EXAMPLE

Over the past year we have been using formal definitions as a practical aid in the design of computer systems. This is part of a more general effort to promote better human engineering of computer systems. Formal definitions allow the designer to capture and explore even the most "trivial" design decisions. As we have pointed out elsewhere [2], many of these seemingly trivial decisions have great impact on the human factors of systems.

Our effort was based on a small but moderately powerful interactive editor. As our first design document, we produced a user's manual for the editor. We next attempted a complete formal definition. It was not our intent in this effort to stick with one existing formal notation, but rather to use off-the-shelf schemes like productions systems, VDL, and axiomatic approaches wherever they were convenient for a particular aspect of the problem. The formal definition itself was attempted ***before*** any coding.

During the writing of the definition, we discovered several difficult problems with conventional techniques. For one thing, there were several aspects of the system that were very difficult to define in an easily understandable way. These were mainly the "interactive" parts; for example, the sequence of events in a user dialogue and the mechanisms for dealing with interrupts. For another, the attempt to do a truly complete definition was suspended because of the apparently excessive detail. This problem was exaggerated by the difficulty of specifying general rules of behavior, which instead were often described on a case-by-case basis.

As a result, we became quite aware of the limitations in using current definition techniques. It is our belief that, except for syntax (including context-sensitive rules), such techniques are not yet ripe for easy definitions of complete systems. We estimate that roughly half of the full definition was finally completed. Our standard of "completeness" is rigorous, for we attempted to define the entire system as seen by the user, including all special cases and message interactions. The partial definition comprises about ten pages of text. The full implementation comprises about eighty pages of code.

The major consequence of the effort, however, was a deepening belief in the need for formal definitions as a practical design tool. As we wrote the definition of the editor, we gained a much deeper insight into a system we thought we understood well. The definition was especially helpful in treating special cases. Null strings, null patterns, empty line sequences, missing files, text boundaries, ends of lines, and the use of replacement strings were exposed in detail. The difficulties in these areas are known, but rigorously dealing with these problems before coding was a major benefit.

Most importantly, the definition allowed us to develop a high level and uniform view of the entire editor. In any effort there can be deep problems in organizing a view of the system that leads to a simple integration of all of its parts, including the special cases. At every rewrite of the definition, we developed a clearer idea of a sensible organization. Getting the semantics of pattern matching to work uniformly in each request, specifying the complete state before and after each request, and insuring that both forward and backward scanning have a simple symmetry were typical of our concerns. Formalization of our intent had a great effect on our thinking.

Finally, the definition was used heavily during the entire implementation where it considerably simplified the work. Not surprisingly the structure of the finished product mirrored, to a degree, the definition itself. Also not surprisingly, the issues left out of the definition posed severe problems. The handling of interrupts and the user-editor message interaction proved especially difficult. Moreover, controlling the implementation on points not spelled out in the definition was difficult. If you don't say what you want to do, you may get surprises, and we did.

3. CONCLUSION

As mentioned in the Sampler, the following reasons have traditionally been given to justify the use of formal definition techniques:

1. *Unambiguous definitions.* There is a need to find a single source for answers to questions and, in particular, questions about the details of a language or system. User manuals and reference manuals, almost always written in English, do not serve this purpose very well.

2. *Manufacturing specifications.* At present, we have no precise mechanisms to serve as manufacturing specifications for use with vendors. It is impossible to make a contract with a vendor and be assured that the product will conform to our expectations.

3. *Standardization efforts.* Standardization efforts have been impeded by lack of an adequate formal notation. While it is true that there are larger, non-technical considerations in developing standards, the lack of suitable formal notations is certainly a major obstacle.

4. *Study of languages.* There is a need to study the implications of language design decisions carefully. It is claimed that with a common meta-language, we can analyze and compare several source languages in terms of a common definition mechanism.

5. *Resolution of detail.* The use of a formal definition mechanism exposes many design decisions. It is claimed that use of formal definition techniques forces one to resolve details that would otherwise be overlooked in an informal specification.

6. *Detection of design flaws.* In writing a formal definition of a language or system, many constructs may be difficult to define. It is claimed that with a suitable definition mechanism, the inconsistencies of a system will be more easily detected. This is a somewhat risky claim, for a given notation may be more suitable to one language or system than another.

Each of these reasons points out some of the benefits of a formal definition, and we support them. However, the task of defining a language or system formally is so difficult (at present) that it is unlikely one would resort to a formal definition to resolve one or two of the above problems.

Because of our experience with the editing system, we now believe that the major benefit of formal definitions is as *a basis for the detailed design of computer systems.* It is our contention that writing a formal definition serves system designers and implementors much as the development of an *architectural blueprint serves in the design and construction of a building.* If precise definitions are developed during the design process, a much deeper understanding of the entire system results. Furthermore, the definition readily points out the difficulties and special cases that must be resolved before implementation. Finally, the definition allows one to develop a view of the entire system, including special cases into a coherent whole.

In a sense, the writing of formal definitions is a design tool for programming at the very highest level. It provides a view of the system from which actual implementation can proceed at a much more rapid *rate*, and most importantly, with a much higher *quality*.

REFERENCES

[1] Michael Marcotty, Henry Ledgard, and Gregor Bochmann
"A Sampler of Formal Definitions"
COMPUTING SURVEYS, Volume 8(2), June 1976, p. 191-276.

[2] Andrew Singer and Henry Ledgard
"The Case for Human Engineering"
COINS Technical Report #77-11, University of Massachusetts, 01003