SOFTWARE ENGINEERING LESSONS FOR VLSI DESIGN METHODOLOGY

Wm. Randolph Franklin

Reprinted from IEEE 1982 WORKSHOP REPORT ON
VLSI AND SOFTWARE ENGINEERING WORKSHOP

IEEE COMPUTER SOCIETY
1109 Spring Street, Suite 300
Silver Spring, MD 20910

IEEE COMPUTER SOCIETY PRESS

SOFTWARE ENGINEERING LESSONS FOR VLSI DESIGN METHODOLOGY *

Wm. Randolph Franklin

Electrical, Computer, and Systems Engineering Dept.
Rensselaer Polytechnic Institute
Troy, New York, USA, 12181

ABSTRACT

There are several lessons that VLSI designers can learn from the recent history of software engineering. They include 1) the need for the widespread dissemination of existing tools and techniques in addition to the development of new ones, 2) the need for standardization efforts in software and database formats, and 3) the difference between the interests of the manufacturers and the users. This paper presents examples to illustrate these points.

INTRODUCTION

In this paper, we will see several areas where design methodologies from software engineering and computer graphics can teach us lessons for VLSI. In fact, the term "Software Engineering" was coined to express the need for some logical engineering methodology in software development. That field has been very slow to learn, billions of dollars have been wasted, and as Boehm[3] shows, rules formulated over 20 years ago, such as testable requirements, and defensive programming, are still often ignored. We have the opportunity to avoid much of this wasted effort in VLSI design. Some of the areas where design methodology transfer can occur SWE and VLSI that are considered here include:

i) the need for dissemination of existing tools,
ii) the need for standardization, and
iii) the difference between the interests of users and the interests of the manufacturers.

We will now consider them individually.

KNOWLEDGE DISSEMINATION

In this section, we will argue that there is a need for a organization whose job is to take research quality software used in VLSI research and development which has demonstrated a general usefulness, prepare it for public distribution, and support it. There must be a mechanism to disseminate software tools once they have been shown to work at one research location.

This has not happened in software engineering, where there is an enormous difference between the current state of the art and what the average programmer is forced to use. Some of the techniques in software engineering that are slow to propagate are:

i) very high level languages such as Lisp and Smalltalk,
ii) interactive debuggers, and
iii) abstract databases.

In the real world, by comparison, IBM Assembly Language is one of the most widely used languages in the USA, and many people still debug programs in batch mode with hexadecimal core dumps.

Software development is becoming a capital intensive industry in the sense that a large investment in tools is necessary to achieve high programmer productivity[21]. However, organizations that will buy a $250,000 midi-computer system without qualms refuse to buy a powerful screen-oriented programmable text editor for $5,000, or a good document formatter for $10,000. They see software as an insubstantial item that cannot be compared to hard money.

This preparation of tools for public use will not happen unless someone is paid to do this. According to Brooks[6], there is a three to one ratio in cost between a program, suitable for its author to use, and a programming product, suitable for public use. The programming product has not only a working program, but also

i) extensive error checking for invalid input,
ii) help files,
iii) user documentation,
iv) a humane user interface,
v) an internal logic and maintenance manual to help the user customize it, and
vi) a set of test cases that so that the user can do regression testing after a change.

One attempt to create a public set of useful software was the software tools project[17]. This was a project to increase Fortran programming productivity by writing a front end preprocessor adding block structure and macros called Ratfor[12],

* This material is based upon work supported by the National Science Foundation under grant ECS 80-21504.

and then a set of programs ranging up from character manipulation and I/O to an editor and document formatter. With these, a user can move from any computer (that has Fortran) to another while working in the same environment. What makes this project unique is that all the source code is freely available so that users are encouraged to modify it. Modifications that appeared to be generally useful might be distributed to everyone later. Scherrer at Lawrence Berkeley Labs made a valiant effort to use DOE seed money to start a volunteer group to manage this process, i.e. answer user queries, accept mods, test them, combine them with other mods of the same program, and then distribute them. Although everyone involved with the process agreed that the result was useful, no organization wanted to allow its people to volunteer sufficient labor for a long enough period to ensure continuity.

The lesson for VLSI is that there will need to be a funded clearinghouse to handle generally useful software. On a simple level, is it more useful for each institution that receives a large program from another to spend several man-months (at least) bringing it up, or is it more cost effective to have one organization whose job it is to bring that software up to public distribution standards?

In another example, Bernard[2] has studied cooperative computing arrangements that span several independent organizations. (His lessons would also apply to cooperative arrangements involving VLSI research.) These arrangements involved either:

i) a shared application, such as an airline reservation system,
ii) shared processing, such as several institutions buying a supercomputer among them,
iii) shared data, such as the National Crime Information Center,
iv) shared software, such as the Health Education Network, or
v) a general purpose combination of the above, such as the Arpanet.

He identified five common problems:

i) the need for a central authority,
ii) conflicts of interest among participants,
iii) loss of local autonomy,
iv) separate perception of costs and benefits, and
v) coordination and user support.

His lesson was that a network organization, separately funded and separate from any of the nodes of the network, is necessary.

Finally, although it is necessary to have generally useful software distributed, it would be better to have it done by a separate funded organization than as a commercial enterprise, because preparation of software for public use is different from commercialization for sale in several ways. In a commercial system:

i) The source code is often not released,
ii) modifications and extensions by the user are not encouraged, and
iii) the software is not designed to interface with other packages that would be considered as competitors, even though this would benefit the user.

Thus in summary, unless an organization is established to take the best research ideas in VLSI software and prepare it for unrestricted public use, progress in the field will suffer.

STANDARDIZATION

In VLSI design, as ever larger databases of circuits are created and modified, and as more complex design programs are created that must interface with each other, there arises a need for standardization in data structures and program interfaces. We already see CIF with assorted dialects, and other languages. This parallels some of the problems that have occurred in computer graphics concerning standardization and compatibility between different graphic devices. Some of the standardization that resulted includes:

i) the SIGGRAPH Core standard[1] and GKS, the Graphics Kernal System[16], in graphics, and

ii) IGES in CAD[10,14,20].

That different E-beam systems take input in different formats, such as raster scan, random vector, and rectangle, is similar to the problem of different graphics output devices. There it has been found possible to have common display files in spite of the apparent incompatibilities. It is worthwhile to consider the SIGGRAPH Core effort since it involved a large, fast growing discipline of both academic and industrial importance. (GKS is a European graphics standard similar to the SIGGRAPH core. It has been adopted by the ISO and may be adopted by ANSI as the US standard.)

The Core report states that "a successful design activity requires essential ingredients": a body of knowledge and a design methodology or set of strategies and codes of practice for the designer to follow. Thus, before the committee could present a standard graphics package, they had to formulate a graphics standards design methodology.

The purpose of the standardization effort was not only program portability, but also programmer portability, i.e. the ability of people to move from one installation to another with minimal retraining. There are three levels of program portability:

i) No changes required,
ii) Only editorial changes required, such as changing routine names, and
iii) Structural changes required such as when the program assumes multiple segments or subpictures but the new installation does

not allow them.

The first is ideal for porting programs, while if the program adheres to the second level, porting is still reasonably easy to achieve. However, if the program is at the third level, then porting is difficult and error-prone. It is also possible to define a subset of the standard such that programs that adhere to it are easier to port.

All of these general rules concerning graphics program portability also apply to VLSI database portability, and to VLSI program portability when the programs must interface to common data. These are some of the problems that are being addressed by the IGES (Initial Graphics Exchange Specification) effort. It is designed to exchange CAD/CAM geometrical model data between dissimilar systems using commercially standard pre- and post-processors. It is now an ANSI standard, and has been demonstrated to work in practice. It preserves a manufacturer's proprietary internal database by allowing him to write processors to read or write the data onto a tape in the public IGES format. Some variant of this standard might be useful for graphic VLSI databases.

DIFFERING INTERESTS OF MANUFACTURERS AND USERS

In the dynamic field of VLSI development, the various contending parties of the manufacturers and the users have different interests, sometimes possibly to the extent of hindering the development of the field as a whole. This has also happened in several forms in software engineering and graphics:

a) Non-standard extensions to standard languages: Almost every Fortran-66 compiler has extra features added, such as END= in READ, generalized array subscripts, and free-format I/O. These help the user somewhat, at least in the short term. However, software endures longer than hardware; there are probably programs written for IBM1401's still being run on emulators. Thus in five or ten years, people may want to move the programs to a new machine, but non-standard extensions will prevent this. This is especially true when the program is poorly documented and the author has vanished.

There is a related problem of a manufacturer who implements "most" of a language. We see this with some micro-computer implementations of Fortran that may have everything but, say, double precision. Sometimes difficult features to understand, such as nested parentheses in format statements, are simply implemented wrong.

ADA, the common DoD language that is being developed[4,5,7,18,19,22], attempts to counteract this by forbidding any subsetting or supersetting. Nevertheless at least one company is now advertising an "ADA compiler" that the fine print reveals to lack most of the features that make ADA interesting.

The lesson for VLSI developers is that once standards are agreed upon, if certain manufacturers add extra features, to ignore them.

b) Proprietary databases to lock in users: Every CAD/CAM workstation manufacturer has his internal data structure format for the customers' data, and the customers are required to keep that format secret. These formats do not generally advance the state of the art in database design. However, they do

i) prevent add-on packages written by others, and

ii) prevent the user from changing to other equipment.

A user reaction to this has been the IGES effort mentioned above. This effort has been hindered by the need to avoid compromising the security of proprietary data formats. The lesson for VLSI development is to avoid a similar entrapment.

c) Software independent of hardware: In spite of efforts to lock users in, it is possible to build portable interfaces on top of non-portable features such as operating systems. This was the point of the software tools project. Another example is the UNIX (R) operating system,[8], originally written for internal use that grew essentially on its merits without any user support and against competing operating systems designed by the manufacturer of the hardware that it is most commonly used on. As a mark of success, in the last few years many other versions have sprung up that call themselves "extensions" or "UNIX-like". Of course most of them are incompatible, so we have another standardization effort ahead.

Pascal[11,13] is another example of a software product that grew without any initial hardware manufacturer support because it was well designed, implemented and documented.

Thus VLSI researchers and designers must realize that although marketing the products is necessary for the field to survive, their interests do not always match. Thus the VLSI researchers and designers must make an attempt to keep formats portable and standard independent of the hardware.

SUMMARY

We have seen several areas in which design methodology in VLSI can learn from the history of software engineering.

REFERENCES

1. Assoc. for Computing Machinery. "Status Report of the Graphics Standards Planning Committee", Computer Graphics, a Quarterly Report of SIGGRAPH-ACM 13, 3, (August 1979), 1 - V-10.

2. D. Bernard. "Management Issues in Cooperative Computing", ACM Computing Surveys 11, 1, (March 1979), 3-17.

3. B.W. Boehm. "Software Engineering - As It Is", 4th International Conference on Software Engineering, Munich, 1979, sponsored by the IEEE and ACM, 11-21.

4. C.L. Braun. "Guest Editor's Introduction: ADA - Programming in the 80's", IEEE Computer 14, 6, (June 1981), 11-12.

5. R.F. Brender and I.R. Nassi. "What is ADA?", IEEE Computer 14, 6, (June 1981), 17-25.

6. F. Brooks. The Mythical Man-Month, Addison-Wesley, (1975).

7. W.E. Carlson. "ADA: a Promising Beginning", IEEE Computer 14, 6, (June 1981), 13-16.

8. T.H. Crowley et al. (the whole issue), Bell System Technical Journal 57, 6 part 2, (July - August 1978), 1897-2312.

10. IGES Y14.26M Response Committee. Final Draft Sections 1,2,3, and 4 Proposed American National Standard, Engineering Drawing and Related Documentation Practices, Digital Representation for Communication of Product Definition Data, Approved as an American National Standard, September 21, 1981, (January 1982).

11. K. Jensen and N. Wirth. Pascal User Manual and Report (2nd edition), Springer-Verlag, (1978).

12. B.W. Kernighan and P.J. Plauger. Software Tools, Addison-Wesley, (1976).

13. B.W. Kernighan and P.J. Plauger. Software Tools in Pascal, Addison-Wesley, (1981).

14. M.H. Liewald and P.R. Kennicott. "Intersystem Data Transfer via IGES", IEEE Computer Graphics and Applications 2, 3, (May 1982), 55-63.

15. National Bureau of Standards. A Technical briefing on the Initial Graphics Exchange Specification (IGES), Automated Product Technology Division, Center for Manufacturing Engineering, National Engineering Lab, US Dept. of Commerce, National Bureau of Standards, Washington DC 20234. NBSIR 81-2297, (July 1981).

16. H.K. Quigley, Jr. "The Standardization of Computer Graphics", Computer Graphics News 2, 4, (November/December 1982), 4.

17. D. Scherrer. "Editorial", Software Tools Communications, 7, Newsletter of the Software Tools Users Group, Lawrence Berkeley Lab, CSAM-50B/3238, University of California, Berkeley, CA, 94720. (November 1981), 1-5.

18. V. Stenning, T. Froggatt, R. Gilbert, and E. Thomas. "The ADA Environment: A Perspective", IEEE Computer 14, 6, (June 1981), 26-36.

19. United States Department of Defense. Reference Manual for the ADA Programming Language, (July 1980), revised (Summer 1982).

20. J. Wellington (ed). IGES Newsletter, National Bureau of Standards, Bldg. 220, Rm. A-353, Washington, DC, 20234.

21. P. Wegner. "Reflections on Capital-Intensive Software Technology, (Draft: 17 Sepember 1982)", Software Engineering Notes, An Informal Newsletter of the ACM-SIGSOFT 7, 4, (October 1982), 24-33.

22. M.I. Wolfe, et al. "The Ada Language System", IEEE Computer 14, 6, (June 1981), 37-46.