© Euromicro, 1977
North-Holland Publishing Co., Amsterdam

Gaines, B. R. (1977). Hardware engineering and software engineering. Euromicro Newsletter 3, 16-21.

HARDWARE ENGINEERING AND SOFTWARE ENGINEERING

B. R. Gaines

Department of Electrical Engineering Science,
University of Essex, Colchester, U.K.

ABSTRACT

This paper is concerned with the teaching of software engineering techniques to electronic engineers. It is noted that the majority of those designing systems around microprocessors have probably been trained as engineers, and suggested that training in software engineering techniques should be based on conventional engineering experience in both design and manufacturing. Problems with software arise in large part because its design and production is treated quite differently from that of hardware. Concepts such as modularity, process-structuring and virtual machines, are not primarily software ones but reflect concepts and disciplines that are readily assimilated in conventional electronic system design. Even programming, as an activity, can be seen to have its counterpart in other aspects of electronic system design that do not involve computers. Thus the approach to teaching advocated is one that concentrates on the integration of hardware and software design and production techniques and attempts to break down the separation that seems to have grown between them.

1. INTRODUCTION

The majority of those designing systems around microprocessors have been trained as engineers, probably electronic engineers. Conversely, it may well also be that the majority of those currently training as electronic engineers will find themselves using microprocessors in their designs. Whilst much effort has been expended, and much has been written, in recent years on techniques of "structured programming"[2] and "software engineering"[7,1] this generally starts from basic principles and assumes little knowledge of engineering in general. However, it is reasonable to assume that someone who is professionally an "electronic engineer" already has the attitudes motivation and, quite possibly, the experience to appreciate problems of design and production in his own industry. Indeed the electronics industry is one with very well-developed techniques for both design and development, and production engineering. Hence it is desirable that any training in "software" engineering should build upon this existent knowledge and the experience, at least of the industry as a whole. Even if it is not all available to the individual, it will be more comprehensible and motivating to relate software development and production to hardware development and production.

This approach is not only desirable on educational grounds but also on industrial ones. The microprocessor is widely infiltrating the electronics industry masquerading as a piece of electronic circuit. It brings with it, however, what appears to be an entirely new dimension of problems associated with software development, production and general management. Whilst the industry has, painfully over many years, largely resolved the problems of managing product development and production of electronic systems despite the rapidly changing technologies involved, it has not yet achieved comparable control of software. There is no simple answer to why but from observations of a number of different electronics firms with "software problems", a pattern emerges: firstly, programming is itself regarded as a quite separate activity from electronic circuit design - one not within the domain or knowledge of the electronic engineer; secondly, as a result of this, software development and production is not brought under the normal manufacturing disciplines and "outsiders" are brought in to be responsible for it - the outsiders have not generally been trained as engineers, do not have the inclination to accept the discipline involved in product development and manufacture and, because they are regarded as "different" do not develop this inclination, eventually coming to accept themselves as different.

This is a black and white sketch that portrays a complex situation in over-simplified terms. There are many firms who have successful, well-integrated software teams. However, these tend to be the larger organisations who have been supplying large computer-based systems to specialist markets for many years, for example in process control or flight simulation. It is the present ubiquity of the microprocessor that is the source of problems. It is suddenly an attractive electronic component to a wide range of smaller industries. There is a sudden need for software engineering on a very wide front and totally inadequate resources to undertake it if it is assumed that software engineering is a quite separate activity from electronic circuitry design.

The theme of this paper, and the examples to be given, is that software engineering and electronic design are not distinct activities. They are both part of an enlarged scope for electronic engineering. One might wish to call it "system engineering" but this is a grandiose term for an activity that has been the foundation of product development throughout

industrial time. A disadvantage of any new terminology is that it gives an impression of qualitative change, of the requirement for a new discipline. In practice one may expect software engineers to be assimilated into electronic engineering so that programming does become recognised as part of the essential tools of the professional engineer as much as, say, filter design is today.

Why should this happen - why not a new profession, a new industry? The answer is largely pragmatic: firstly, the microprocessor can replace so many conventional circuit techniques, not just those normally regarded as digital, for example in low frequency filtering and signal processing, previously carried out by analogue circuits - it is a major electronic "component"; secondly, "programmable circuits" with functions controlled by digital memories, are becoming increasingly important - these are not computers in the von Neumann sense yet, in terms of "software" requirements, exactly parallel computers; and, thirdly, the use of larger building blocks of all kinds is concentrating design activity in electronic engineering into the system analysis, paper design and simulation phase, with actual system prototyping and testing being far less arduous and time-consuming than it used to be. We are all increasingly "software engineers", whatever we may wish to be.

It is tempting to terminate this paper at this point. The key messages are that: "software engineering" is an integral part of electronic engineering; its teaching should be based on current experience and techniques in the development and production of electronic equipment; and it should be regarded as an essential component of the knowledge of any professional electronic engineer. In particular, software development and production should be managed as if it were electronic system development and production. In the second part of the paper I shall introduce some ways of presenting key ideas in software engineering that aid these objectives. However, I am acutely conscious that these are fragmentary and do not live up to the grand objectives of this introduction. It will require the bringing together of the knowledge and experience of a great many firms and individuals to achieve the wider objective of the proper understanding and integration of software engineering. The following notes are intended as some contribution towards this.

2 INTRODUCING SOME KEY CONCEPTS

A basic precept of all successful systems engineering is "divide and rule", and software engineering is no exception. The concepts of modularity[3], process-structuring[6] and virtual machines[5], are all concerned with structuring the system into components, or levels, with well-defined functional characteristics and interactions. As with the majority of key concepts in computer science, the literature on these concepts primarily exemplifies fairly large configurations. This can readily give the impression that the ideas are most relevant to big configurations, perhaps also to minicomputers, but hardly to microprocessor-based systems. One of the first points to be made is that these concepts and associated techniques of software engineering apply even to the smallest microprocessor configuration.

To illustrate the complexity of software structure possible on small configurations, and the range of software techniques used, I use as an example the system illustrated in figures 1 and 2 - most readers will have experience with comparable systems. This is a clinical patient monitoring system developed for the Department of Health. Even though the processor has only 1024 x 8-bit memory, the system exemplifies many different peripherals, illustrates real-time data acquisition and interactive display, and may be used to illustrate a variety of techniques from file structures to interactive dialogue. Clearly, some part of the system complexity is due to the availability of backing store, but the system is small enough nevertheless to be readily understood and typifies a microprocessor/floppy disc configuration.

I use this system to illustrate the virtues of modularity, considering its design, debugging, and field updating. It may also be used to illustrate process structuring and the problems of communication between asynchronous interacting processes. It also enables one to begin to develop the concept of a "virtual machine" - the operating system shown allows interactive programming and debugging, showing a very different face to that of the basic machine, and a different face yet again to the clinician.

Having introduced the notion of a virtual machine in the context of a computer system, it becomes worthwhile to start asking the question, "Is this concept unique to computers?" - this is a recurrent question that I return to repeatedly - the answer is always "No"! The way in which it becomes advantageous to regard a structure of components as itself a "component" at a higher level is fundamental to the electronics industry. The transistors/diodes/resistors/capacitors become a "regulator", the regulator/power source/rectifier/capacitors become a "power supply" and so on. The organisation of electronics production is very much based on a "bottom-up" development of virtual machines. The reasons for this are "obvious" and meaningful and can be used to introduce the same arguments for software organisation.

The "top-down" design process is now also naturally discussed in terms of first hardware, then software. The integration of "bottom-up" and "top-down" processes so that the design task is reasonable at any level clearly comes out. Those with actual design experience of any nature can relate the discussion to their own experience.

These are concepts that have such general applicability that it is a grave distortion, and loss of clarity, if they become primarily associated with software.

However, the lack of obvious differentiation between levels of software modules may now be introduced to lead the discussion to the problem of how programming difficulties come about. Again, one may illustrate this with electronic system examples - the problem solved by modifying an otherwise standard power supply internally rather than by an external interface, shows both the merits and difficulties of operating at the "wrong" level. One can begin to discuss the roles of accepted rules and disciplines in product design, the production engineering framework adopted for hardware and its equivalent for software. The way in which the same manufacturer who has an elaborate system of "design authority" and "change notes" for re-routing a cable, may allow a programmer to change a "few instructions" on an outgoing production item, and the manner and difficulties of controlling this may be used as another form of case history.

3 THE ROLE OF THE COMPUTER

One of the key attributes of the computer is its versatility and hence the multiple different roles it may play. Even in system design itself the concepts can have three major, and distinct, roles: as a design tool, as an automation aid, and as a design component. I use figure 3 to illustrate both these aspects of the computer and also, again, to emphasise the close relationship between classical electronic design, computer-aided design, and computer-based design.

The comparison of 3(a) with 3(b) brings out the role of the computer in "computer-aided design" (CAD), as a simulation tool allowing rapid interactive design, and as an automation aid eliminating manual preparation of drawings or machine-tool control tapes. It is appropriate here to discuss the general roles of simulation in design and automation in production, again because both occur in other guises in software engineering but are not primarily associated with it. Anecdotal examples of experience with simulation packages that predict problems that have not arisen in practice and yet, when deliberately investigated, do so are important here - there is as yet little general appreciation that in many cases simulation may actually be a better basis for design than prototyping.

The use of CAD in 3(b) should be fully acceptable as part of electronic engineering. The transition between 3(b) and 3(c) is intended as a smooth one between hardware and software with the usual emphasis on the high degree of similarity between the two situations. Here the computer plays a new role also, as a part of the final system. The key points are:

(1) The economic merits of computer-based systems can be seen clearly. They stem from the separation of the actual computer design and manufacture from its application. The computer manufacturer designs a complete piece of electronic equipment that can be manufactured, quality-controlled, and maintained for a mass market. He defers the design of actual end-user systems by making a programmable, general-purpose product. This is another illustration of the power of modularity, the "divide-and-rule" principle.

(2) There is a clear and complete separation between the uses of the computer as: a design tool; a production aid; and a system component. Clearly the microprocessor is far more appropriate to the component role than to the design/simulation aid role. Even its use as a production aid, for example in compilation, shows up as clearly different, requiring different configurations and facilities, from its use as a system component. These differences are obvious and often stressed but one still finds many intending users of microprocessors attempting to use the same configuration for design and development as for the final system.

(3) The role of compilation as "automated drafting" is best emphasised by contrasting a machine tool control language, such as APT, with FORTRAN. In both cases one writes similar high-level language statements. In both cases the compiler translates this into a sequence of machine instructions. In one case these instructions control a machine tool to produce a block of metal. In the other case these instructions control a microprocessor to produce a virtual machine.

(4) The role of simulation in program development and testing is itself important and inadequately understood and used. CAD for electronic systems is very much better developed than CAD for computer-based systems themselves! This is probably because most microprocessor-based system designers are still really following the "classical" path of cut-and-try prototyping of 3(a). The system shown in figures 1 and 2 was developed in large part by simulation using an emulator embedded in a high-level language in a PDP10[4]. The advantages of having a complete system emulation interactively available are even more significant for complex microprocessor-based systems than for the far simpler electronic circuits where CAD is now being extensively used.

(5) The key role of problem specification in any scheme of design is apparent. Too often advantage is taken of the supposed "ease of modification" of software to avoid the degree of product requirement analysis and specification normally expected before development itself commences. Whilst computers do allow "deferred design" this is itself part of the specification; future modification, upgrading, re-application,

etc., are never there for the taking - they have to be designed in from the start.

4. PROBLEMS IN THE INTRODUCTION OF MICRO-PROCESSORS INTO PRODUCTS

Even when the distinctions made in figure 3 are fully appreciated, another class of problem can arise in the real industrial situation because the final product itself contains a programmable element. The microprocessor may have been introduced purely on economic grounds as a lower-cost electronic sub-system, but problems arise because both manufacturer and customer expect the use of a "computer" to provide a new level of flexibility that was previously impossible. The overall system has become programmable so that a product can serve a greater variety of applications, is readily modified to new requirements, and this flexibility can even be put in the hands of the end-user. What needs to be analysed, however, is the cost of this new-found flexibility, and the differences it makes to the way in which design, production, marketing and maintenance are organised.

Four distinct levels of use of microprocessors may be distinguished:

(a) The processor replaces hardware elements. Its programming is equivalent to computer-aided design (CAD) of these elements. This affects product development only and the fact that it is itself "computer-based" is irrelevant to all later aspects of marketing, application and support. The advantage of the microprocessor is that it replaces boards of random logic simplifying production and maintenance. There should be no associated disadvantages, particularly if the design group have already been using CAD packages for logic design, board layout, etc.

(b) Basing a product range on microprocessors allows new product development to be largely an extension of the range through software modification rather than new hardware production. This probably entails changes in production techniques with "software modules" being treated in the same way as hardware modules in terms of documentation, testing, etc., but does not affect marketing or the end-user. The additional advantage of the microprocessor is that it allows total hardware re-engineering to be avoided in what may be very substantial product changes. The disadvantage is that production procedures have to be introduced for installing and checking software modules, but this is a reasonably straightforward problem that can be treated using the well-established methodologies for hardware production.

(c) The customisation and field upgrading of individual products is made part of their specification and a key factor in their marketing. Technically this is, in a sense, "already available". If a new product ranges involve "only a change in software" why should not the range become a continuum with each customer selecting the facilities appropriate to his application. The advantages are clear on the marketing side - one of the biggest attractions of computers has always been that they can cater for individual requirements and that, if these turn out to have been misconceived, the computer can always be re-programmed for something else! However, the new problems that now arise are substantial - documentation and customer support has also to be customised and instead of having a standard product range one is now in the "systems" business. This is a viable proposition and such businesses can be profitably managed, but they are not simple extensions of (a) or (b). They have an entirely different cost structure involving a large element of uncertainty, and the final product cost is largely in terms of people not hardware.

(d) The ultimate level of exploitation of the computer-based system is to market the product as one which may be tailored to individual requirements and where the user himself may perform this tailoring. Technically this may be regarded as an extension of switches and dials on the front panel, but the magnitude of the extension can be such that a major qualitative change takes place. Clearly the level of support the user requires is substantial - a far more complex instrument has to be explained to him. The major change in most applications is that the instrument becomes capable of complex-time-dependent procedures. However, there has also been a change in the customer-supplier relationship that has to be recognised. The end-user now has a relatively simple piece of equipment whose complexity and problems lie in the way he programs it, whereas previously they lay in the way that the manufacturer had "programmed" it. The level and types of responsibility and support that the user expects of the supplier have got to change drastically if the cost-structure is to remain the same. The supplier no longer has "system responsibility" for his product. He does not know enough about the way in which it is being used to foresee and warn of all possible problems, and so on. Clearly a new relationship can be established, but it is in doing that that many of the current problems are arising.

One can take a specific type of equipment, for example, for automatic testing, and discuss firstly how the four different levels of application arise in practice and the different forms of problems they create. Then one can begin to show how they may be overcome (without creating the impression that this is simple in any real situation!) using some of the concepts of modularity, virtual machines etc., already introduced.

For example, figure 4 shows a programmable instrument designed as a 5-level hierarchy of virtual machines in which the "development engineer" sees a computer and designs subroutines

for it, for example to control certain peripheral devices and to make certain calculation facilities available (for example, data smoothing, floating point arithmetic, message communication protocols, etc.); the "product designer" configures a system and writes routines using the modules provided by the development engineer to give the required control, data-logging and data-processing facilities for the system in terms of a "problem-orientated language"; the customer develops a program in this language for his ultimate application, thus finally defining the top-level "virtual machine", an instrument to the end-user.

Clearly, in the spirit of all that has been said before, figure 4 should also be re-analysed in terms of a purely hardware implementation, pre-microporcessorisation - analogies may be found in terms of electronic components, modules, subsystems, user controls, etc. for every level of the system. If these analogies can now be seen and appreciated, and the transfer of technology between hardware and software production techniques becomes apparent, attractive and applied, then a major step will have been taken in bringing the microprocessor and "software" into the domain of electronic engineering - where it belongs.

6. CONCLUSIONS

Whilst this paper has stressed the similarities and integration of hardware and software engineering it is not intended to be completely reductionist, to say that software is hardware, or vice-versa. It is not that there are no distinctions to be made, but rather that it is often the wrong ones that seem prevalent at present. Differences in attitudes, jargon, etc. create distinctions that are misleading and lead to sloppy engineering and production control - to an extent that would not be tolerated if "software" and "programming" were not regarded as different from "circuitry" and "prototyping".

However, we must also be aware that the systems we are designing are becoming more complex. Technology builds upon itself iteratively and we are now working at levels very remote from the simple devices of yesteryear. The microprocessor arises as a necessary way of using large-scale integration - of making general-purpose chips with many thousand gates and yet few connecting leads. Already the uses we make of these chips go beyond those of conventional computers - we can readily microprogram special functions, we can readily connect in parallel channels, we can readily interconnect multiple microprocessors, and so on. This is the domain of innovation in computer systems - microprocessors are not little brothers mimicking the big, "real" machines, but are themselves the vanguard of computer research and application.

From this viewpoint it becomes important that effort is concentrated on the real problems of complex system design - that the full advantages of the close, and interchangeable, partnership of hardware and software are fully realised.

7. REFERENCES

1. J.N. Buxton and B. Randell (eds) Software Engineering Techniques, Scientific Affairs Division, NATO, Brussels, April 1970.

2. O.J. Dahl, E.W. Dijkstra and C.A.R. Hoare, Structured Programming, Academic Press,1972.

3. J.B. Dennis, "Modularity" in F.L. Bauer(ed) Advanced Course in Software Engineering, Lecture Notes in Economics and Mathematical Systems, 81, Springer-Verlag, 1973, pp 128 - 182.

4. P.V. Facey and B.R. Gaines," Real-time system design under an emulator embedded in a high-level language". Proc. BCS DATAFAIR 73, April 1973.

5. B.R. Gaines, "Analogy categories, virtual machines and structured programming", in J. MÜhlbacher (ed) GI-5 Jahrestagung, Lectures Notes in Computer Science, 34, Springer-Verlag, 1975, pp. 691-699.

6. J.J. Horning and B. Randell, "Process Structuring", ACM Computing Surveys, 5, 1973, pp. 5-30.

7. P. Naur and B. Randell (eds) Software Engineering, Scientific Affairs Division, NATO, Brussels, January 1969.

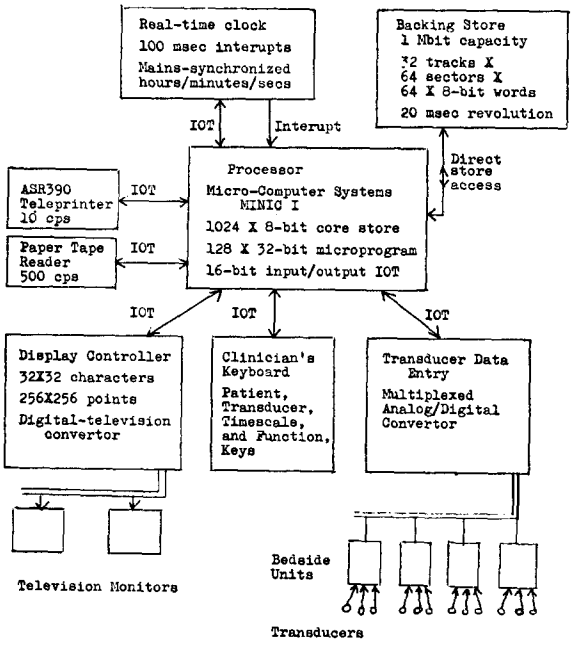

Figure 1 - Hardware Organization of a Small System

MAIN STORE-RESIDENT 256 x 8-BIT PROGRAM PAGES	
PAGE 0 Monitor Routines Status - parameters, flags, communication buffers Interupt service routine Scheduler Overlay caller Utility Routines	**PAGE 1** File Routines 64 x 8-bit buffer area Write block to drum Read block from drum Open and Close a file Read and Write data in a file
PAGE 2 Overlay area - waitable jobs Null-job is teletype listener	**PAGE 3** Overlay area - transient jobs Null-job is clinician's keyboard listener

BACKING STORE-RESIDENT 256 x 8-BIT PROGRAM OVERLAYS
System load, start and restart
System clear and re-structure
Acquire and file patient data
Startup monitoring system
Set up date and time
Update date at midnight
Decode keyboard requests
Display graph labels, lines and data
Read in biochemistry tape
Display biochemistry data
Display drum storage allocation
Housekeep drum files
Core display and debug system
Drum display and debug system

Figure 2 - Software Organization of a Small System

3(a) 'Classical' Electronic System Design

Problem Specification ⇒ Circuit Design, Prototying and Testing ⇒ Drawing Office Specifications ⇒ Production

3(b) Computer-Aided Electronic System Design

Problem Specification ⇒ Circuit Design, Simulation on Interactive Graphics Terminal ⇒ Automated Drawing and Machine-Tool Control Tapes ⇒ Production

3(c) Computer-Based Electronic System Design

Problem Specification ⇒ Program Design, Simulation and Testing ⇒ Compilation ⇒ Production (program in computer)

Computer Manufacture ⇒ (deferred system design)

Figure 3 Comparison of 3 Forms of Electronic System Design

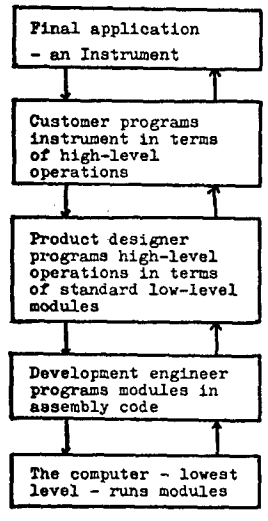

Figure 4 Virtual Machine Hierarchy in Production of a Programmable Range of Instruments

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE
EIDG. TECHNISCHE HOCHSCHULE — LAUSANNE
POLITECNICO FEDERALE DI LOSANNA

Département d'électricité

DÉPARTEMENT D'ÉLECTRICITÉ
19 JAN 1977

LABORATOIRE
DE CALCULATRICES DIGITALES

16, ch. de Bellerive
CH-1007 Lausanne (Suisse)
Téléphone (021) 26 46 21
Télex 24 478

Prof. Gaines.

V/réf. N/réf. JDN/cn Lausanne, January 12, 1977

Dear Author:

We thank you very much for the paper you have submitted for the special April issue of the EUROMICRO Newsletter.

Your paper has been accepted, and you get enclosed the special sheets for preparing your text. We are sorry that this shipment has been delayed because of the problem of getting the special sheets (package lost in the Christmas rush).

We got many papers for this special issue, and we ask you to compactify your paper as much as possible. Text should be typed according the joined instructions, figures and photographs must be reduced to the size of one column. If you cannot make yourself the reduction, leave the correct space and join the original.

can be slightly larger

Since we all want to receive the April issue in April, we must send all the documents to North Holland on February 15. Please organize yourself to send me your paper before February 11 (earlier if you are in Italy or overseas). The papers that will not be in my hands in the night of Monday 14 will not be published in the April issue.

Since no major correction has been requested to your paper, I hope that these rather short delays will not be a problem for you.

I thank you very much for your contribution, and I wish you the best for the New Year.

Please keep your text in 5 pages. (adequate reduction of tables)

Sincerely,

Prof. J.D. Nicoud

Return address: Prof. Nicoud
LCD EPFL
Bellerive 16
CH-1007 Lausanne
Switzerland

I hope to get the refereed paper soon

JDN.

RSI