Understanding the software problem

by JOHN B. SLAUGHTER

Naval Electronics Laboratory Center
San Diego, California

INTRODUCTION

In addressing the question of the high cost of software it is essential, first of all, to surround the issue in such a way that it is possible to achieve some understanding of the problem and its causes. The first question to be answered is that of whether or not software costs are, in fact, too high. In order to answer this question requires a definition of software itself. Continuing this vein of questioning obviously can lead to a circuitous path which never leads to anything but tangential conclusions about pseudo problems and pseudo causes.

Many articles, pamphlets, and government documents have been written about “the software problem” and what can be done to solve it. The fact that there is so much attention attributed to this concern is, in fact, indicative that some form of problem is extant and that the future of computer utilization is strongly dependent upon the manner in which software developers and users direct their energies to solve it.

Computer technology has advanced to an era where hardware development costs are declining per unit of capability. Extrapolation tends to indicate that for the foreseeable future, at least, this trend will continue. Conversely, software development costs usually measured in terms of lines of validated code per man-hour expended, are rising and expected to continue to rise. It is enlightening to examine the reasons why this difference exists and attempt to discern what inherent properties the hardware development process has that are not similarly possessed by software production.

There are three major characteristics that can be identified immediately. They are design procedures based upon engineering discipline, highly developed automated manufacturing processes and technology, and the existence and use of design and production standards. None of these exist in a macroscopic sense for software.

A successful design effort must begin with a carefully conceived requirement that can be stated in the form of performance and design specifications which can be understood and followed by design engineers. Furthermore, it allows the project manager to keep track of the development and maintain control of the activities. This does not exist in the realm of software development. The reasons are several. Managers generally do not understand software and are unable to generate adequate specifications for its performance and design. Neither are they very capable of managing its development since, in comparison with a hardware device, it is an intangible quantity whose final form defies description in an engineering sense.

The absence of agreement on languages, programming aids, documentation methods, etc., for software is another major problem. Software development is a highly individually oriented activity where the programmer’s skills, biases, and motivation govern the process. Automated program development is used by a small minority of programmers; most coding is performed by pencil and program worksheet. The wide range of abilities found in the members of the programming profession, the absence of a prescribed curriculum of educational experiences which are required for a person to qualify as a programmer, and the relative immaturity of the software profession all combine to create this facet of the problem.

The absence of design and performance standards which are used to govern software design is a key reason for some of the software development difficulties encountered by industry and government. Such standards are required if modular programs are to be developed since interfaces must be defined and met and care must be taken to assure total system compatibility of the component modules. Standards are also the essential link in software transferability from one application to another. The cost of generating a new program module each time it is required to perform a function could be eliminated or at least reduced greatly if adequate design and documentation standards were applied.

It is obvious that much of the software problem is due to the non-existence of the discipline and rigor which characterizes hardware design and production. Steps are being taken to introduce these needed features into the software process, e.g., structured programming, and the results are expected to provide improvements in software quality in addition to reductions in software development costs.

During the workshop sessions it was stated by one participant that the overwhelming characteristic of software is uncertainty. What constitutes reasonableness in terms of performance, cost, and schedule? The unavailability of sound answers to questions such as these is a major concern to
developers, purchasers, and users of software. An attempt to define approaches to obtaining answers was a major theme of Workshop No. 1.

THE SOFTWARE PROCESS

Rather than attempt to state or develop a dictionary definition of software, it seems more meaningful and expedient to define the elements of the software process. These elements can be viewed as line-item cost factors that might appear in a budget for a major software system development. Thus they provide a very useful means of identifying where costs occur in the process.

PHASES

<table>
<thead>
<tr>
<th>Elements</th>
<th>Development</th>
<th>Maintenance</th>
<th>Major Redesign</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analyze</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compile</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integrate</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Document</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manage</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Provide</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>facilities</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

THE SOFTWARE PROCESS

The production of software requires the successful completion of several specific activities. In the foregoing chart ten activities have been identified as cost elements which must be taken into consideration in the development of a software product. Included among these are the elements of software production management and the provision of facilities and equipment on which the development will be accomplished. These are two significant factors which must be taken into account in any analysis of cost even though they are usually not addressed in most examinations of the investment required to produce a software output.

The chart also delineates three phases which occur during the life-cycle of any sizable software product. They include the initial development of the software which has been fully designed, tested, and documented for some particular application. The cost associated with this function is only a fraction of the total cost which must be allocated to the product. Also the maintenance of the software, i.e., the correction and elimination of faults which are discovered only after the package is put into use, minor improvements which are made to enhance the operation of the program, and expansion required to allow the program to work with minor changes in function, procedures, or equipment, must be considered as an important component of the total cost algorithm. Finally, major redesign or revision of the software is often required to accommodate major changes in mission, equipment, or operating doctrine. The extent to which the software is designed and documented initially to allow the maintenance and redesign functions to be accomplished easily are major considerations which must be addressed. Lifecycle cost, therefore, is the sum of the individual costs associated with each of the three phases.

Each of the cost elements will appear to some extent in each of the three phases. The weighting of each element will no doubt differ depending upon the phase being considered. Although time was not available during the workshop to quantify the extent to which a particular element is involved in the three phases, it was the consensus that such an effort could be very productive in focussing on where major cost problems occur.

As stated in the introduction, uncertainty is one of the major deterrents to a clear definition of the software problem. This is due in part to the paucity of data which can be used to examine where costs are incurred in the software process. An approach such as that described above, although perhaps somewhat simplistic in the exact form shown, would be useful as a means of acquiring needed information and providing a basis for improvement.

SOFTWARE PROBLEMS

What are the problems associated with the development of software? How do these problems manifest themselves and what are their causes? In answering these questions it is useful to assume the perspective of the individual or agency that contracts for a software development. The problems that are viewed from this vantage point are basically the following:

- Software Quality
- Life-Cycle Cost
- Delivery Schedule

It is obvious that these three factors are not independent variables and that perturbation of any one of them has a significant impact on each of the others. Nevertheless it is possible to examine each of them separately and attempt to identify the form in which the problem appears and some of the prominent reasons for its occurrence. From such an analysis, it is possible that the shape of potential solutions may become evident and that future research and development can be directed toward solving the basic problem.

Problem: Software Quality

Symptoms:
1. Unreliable
2. Unresponsive
3. Incompatible
4. Non-adaptable
5. Non-transferable
6. Uncertified

From the collection of the Computer History Museum (www.computerhistory.org)
Principal Causes:
1. Inadequate statement of requirements by user
2. Inadequate understanding of user requirements
3. Poor testing and certification practices
4. Lack of standards by which performance can be measured
5. Inadequate documentation
6. Lack of appropriate management attention and control
7. Improper use of current technology
8. Inadequate programmer skill levels
9. Inadequate hardware/software trade-offs
10. Lack of adequate support software

Problem: Life-Cycle Cost

Symptoms:
1. High initial development cost
2. High operational and maintenance costs
3. Poor utilization of machine resources
4. Costly to modify
5. High cost of documentation

Principal Causes:
1. Poor estimation of production costs
2. Poor procurement practices
3. Poor software development practices
4. Lack of automated programming techniques
5. Improper or non-use of existing developments
6. Inadequate system hardware
7. Inadequate programmer skill levels
8. Poor system requirements and specifications
9. Lack of management control of costs
10. High salaries of programmers
11. Uncertainty of cost allocation
12. Inadequate attention to system integration and testing
13. Poor documentation practices

Problem: Delivery Schedule

Symptoms:
1. Failure to meet schedule
2. Long development time
3. Untimely software documentation

Principal Causes:
1. Poor estimation practices
2. Inadequate definition/understanding of job
3. Variable programmer skills and productivity
4. Poor management control and monitoring
5. Unrealistic milestones
6. Inadequate use of existing developments
7. Long lead-time procurement
8. Inadequate support software
9. Lack of automated programming activities
10. Inadequate attention to documentation

The repetitious presence of such factors as insufficient requirements, inadequate attention to testing, documentation, and integration, poor software management, lack of support software, and utilization of outdated techniques and tools points to these as the primary woes of the software community.

Many solutions to these problems have been postulated but their success requires a significant change in thinking by the developers and users of software. For example, the concept of structured programming in which software teams of specialists are formed is perhaps an approach which strikes at the heart of most of the faults attributed to the software generators. Structured programming follows a top-down design approach which inherently possesses an effective managerial structure, an ever visible and current set of documentation, an emphasis on testing and integration, a clear delineation of responsibility, and the development and utilization of new tools essential to the completion of the project. Experience with structured programming teams has shown that productivity of an order of magnitude can be achieved by the use of such a team over that obtained from a well-qualified individual programmer.

Changing user's habits may be more difficult. A more knowledgeable and aware set of project managers must be developed to generate meaningful requirements and specifications for software, to understand the dynamics of the hardware/software trade-off equation, and to provide the overall direction necessary to successfully integrate a complex software program with state-of-the-art hardware.

PROPOSED RESEARCH EFFORTS

One of the key causes of the uncertainty characteristic of software development is the absence of meaningful standards. Not only are there inadequate measures of performance, there are also inconsistencies in the jargon of the profession. Productivity of programmers is difficult to define because of different yardsticks. A standard data base is needed to provide benchmarks which allow software people to communicate meaningfully and effectively.

The question of standard languages is a particularly meaningful one to address. When one considers the number of different languages available (FORTRAN, COBOL, JOVIAL, CMS-2, PL-1, . . . ) and the many versions which exist for any one of these, an appreciation of the lack of language standardization is readily gained. There is a strong and direct relationship between the lack of language standards and the cost of software. It can be measured in terms of the price of additional documentation, non-transferability
of code, compilers, etc., development of new and often unique tools, unreliability, and the resultant large software inventory required. There is a need for standard operating system interfaces which can achieve a reduction in errors over that obtainable with current job control languages. A standard data definition language which eliminates the need for many data structures is another important requirement. Research into the costs of language standardization and how languages can be brought closer together represent sensible efforts to be initiated.

Workshop No. 1 addressed several "causes" of the software problem to attempt to develop approaches which could be recommended as potential means of obtaining solutions. The three chosen for consideration were:

- Lack of uniform measures of performance
- Inadequate exchange of information within software community
- Non-transferability of technology developments

It has been stated earlier in this paper that software development and procurement are handicapped by the absence of a meaningful and uniform system of metrics. Because of this, it is not possible to quantitatively measure the performance of either the programmer or his product and be assured that the reported values can be understood by others in the software community. As a result of this situation, it is difficult to identify what, if any, progress is being made in the area of increasing programmer productivity, for example. The recommended action to overcome this situation is the initiation of an R&D program designed to establish measures of programmer productivity. The approach to be followed is to collect objective statistics on such items as errors/lines of code, lines/day/programmer, and program documentation. In order to collect accurate statistics and make the results of the research meaningful, automated aids which accumulate statistics and on-line code production facilities are needed.

In spite of the numerous journals, conferences, symposia, seminars, etc., which address various aspects of software technology, working-level programmers are not sufficiently aware of the contributions which currently available techniques and tools can provide. Neither are they able to articulate precisely the help which they feel they need. It seems reasonable to make a concerted effort to identify users (persons who write software) and ascertain the form of the problems which they encounter and the kind of tools which they perceive would be useful to them. The recommended action is that there be established a 4-6 person ad hoc group consisting of tri-service, ARPA, FCRC members who during a period of 6-8 weeks will do a nation-wide study on the following questions:

- Who creates software? Where?
- What software functions are performed (requirements, analysis, coding, documentation, ...)?
- What are the applications (Scientific, MIS, C2, ...)?
- What tools are in use?
- What tools are perceived as needed?
- What problems or difficulties exist?

Strongly related to the previous two issues is the question of transferability of technology developments throughout the community. Because there is no efficient information network linking software developers, useful and available tools are not used widely, and feasible techniques and tools are often not carried to completion. To solve this problem it is recommended that a network-based (ARPANET, for example) software evaluation and exchange facility be established. This facility would perform the following roles:

- Evaluate new software tools
- Disseminate evaluation results and experiences of other users to requestors (a system of charges to use the services can be established)
- Provide for general information exchange on techniques, standards, languages, etc.

CONCLUSIONS

Near the termination of Workshop No. 1, the question of what future course of action, if any, should be followed was discussed. Unanimously it was felt that some follow-on program was required in order to further the coupling of the deliberations of our meetings to users. The suggestions ranged from scheduling a regular series of meetings of our workshop group to establishing a full-time DOD activity to address the important issues of software. Certainly the sponsors of the symposium will want to consider a wide range of possibilities. It is believed by the participants of our workshop that a valuable exchange of information and ideas occurred as a result of the symposium and we would like to thank the sponsors and the chairmen for providing the opportunity. I would like to thank each member of my workshop for his (and her) candor, enthusiasm, and support.