JNW DRAFT
7/23/86

REFLECTIONS ON SOFTWARE DEVELOPMENT

John N. Warfield

There is no question that, in its current prevailing state, software development is much too expensive, and of low quality. This has been pointed out by many observers, including persons intimately acquainted with and a part of the computer software development industry. Persons not acquainted with this industry or its problems need not assume that the foregoing remarks are highly opinionated and idiosyncratic to the writer of this piece.

The purpose of this article is to identify and discuss a variety of factors involved in software development, to try to present perspective on how these factors affect or influence software development, and to offer diagnoses concerning what will and will not improve the industry.

Who Should Care?

Who should care about the state of the software industry? First of all, the computer industry and especially the software component of it should care, for several reasons:

- Any bloated, inefficient, industry with poor quality products ultimately is going to be replaced with an efficient industry with high quality products, or at least is very vulnerable to this possibility.
- The size of the market is limited by knowledge of poor quality products, so growth of the industry is inhibited by the status of its products
- In certain applications, such as national defense or automatic control, many lives may depend upon the quality of this product
- Although just one industry, we have here a large and growing industry, and have already lost part of this industry, as well as many other industries to foreign competition, and this industry should worry about that

That part of the software industry that contracts primarily with the Department of Defense for software should worry about it. Some might believe that because of the special character of defense, American industries may be protected from foreign competition, and does not seem likely to be put out of business, even with the current high-cost of acquiring its products, and the deficiencies exhibited by them. In response to this argument, we may offer the following:

- Every industry that has been lost or is declining was not perceived by its management as being in peril until it was too late to recover
- On the hardware side of the defense business, where a similar argument might be made, we have already lost over half of the defense semiconductor chip business to other countries
- Countries who lack a solid manufacturing infrastructure can, nonetheless, move into software work, as has India. So competition can arise readily and be more widespread than in more capital intensive industries.
- Many potential competitors have people who are better educated in relevant subject matter and are considerably greater in number than what we have

The average citizen should worry about it. The software industry is much like a service industry. Every time we lose manufacturing, we rely on service industry to pick up the job slack. But if we can't even keep those kinds of industries, we are ultimately going to become a poor nation. We do this one step at a time, and to prevent it, we need to stop taking those individual steps downhill.

The universities should worry about it. They are offering curricula that are not really quite suitable to meet the challenge. While considerable curriculum development discussion is taking place relative to software, the results of this still have not met the competitive challenge.

But the universities must have a deeper cause for concern. The role they have frequently played is one of support for the existing industrial and military establishment. The universities have to share responsibility for the malpractice in the software industry. Tradeoffs where individuals seek their own best interests, and avoid the kind of controversy that is required to make progress in this field, will serve the university well in the short run, and will serve the university, the industry, the region, and the nation very badly in the longer run. The plain fact is that it is only the universities that can provide the high-quality guidance that is needed, and they cannot implement this guidance. The task is then twofold:

- Explain to the government and industries what they have to do, and be convincing about it--get it down on paper, and get it straight before starting
- Train the governmental procurement people to write the kinds of contracts that make it possible for industry management to demand proper performance from their technical people

Given the very limited slack in the university systems, if university personnel simply serve short-term industry interests, we will never be able to marshall the critical mass to carry out the above responsibilities.

What About the Whistleblower?

Recently we have seen in the press report after report of indictments or charges against defense industries for cheating the government. Among those named are General Dynamics, TRW, and Litton Industries.

If these companies are deliberately cheating the government in their billings, why should one assume that they are competent or even interested in developing high quality software products?

What about those who see the sorry record of peformance in the software field, and want to correct it?

There has already been developed a kind of approved professional approach to some of the issues. The DOD and some of the industry professionals have collectively produced books about the software problem, and articles are being published in the technical literature about what is wrong. Under these circumstances, is there a need for a whistleblower? And what kind of attitude should be taken toward one?

There have been developed in the past few years several defense initiatives toward improving the software situation. We know of most of these. They are typically run by or associated with universities. Should one not then suspect that the forces have been set in motion to correct the situation?

Regrettably the answer to this question is probably "no". What is the evidence or rationale for this answer?

The same people who have been creating the problems are also running these new initiatives. You can change the name of the play from Hamlet to A Streetcar Named Desire, but if you keep the same cast that made a flop out of Hamlet, how can you be confident of academy awards or Emmys for "Streetcar"?

The only thing that can really provide confidence here is an open management posture, disciplined by the available knowledge, and using management criteria that lend confidence that change will really occur, of the type present knowledge shows must happen.

What are the Obstacles to Success?

Let us first review the obstacles to success, and then let us review some of the potential for change.

The major obstacles to a first-rate software industry include the following:

- Large vested interest in the status quo in both industry and university
- Those in positions of responsibility are not doing what is needed to correct the situation, for any or all of the following reasons:
    * They generally don't understand the problem in depth
    * They do not know what steps to take to correct the situation
    * They get bad advice from those with vested interests
    * They lack the historical perspective that is needed to understand this industry and its science roots
- Those who perceive themselves to be in the midst of education in this area (e.g., people in computer science departments, engineering deans, etc.) have never systematically reasoned in depth about what is required or what is important in the underlying science; consequently the programs that we have are unbalanced, and omit critical knowledge
- The faculty who are preparing people for this field are greatly overworked because of the number of students enrolled, and have not had the time to develop the science; therefore most curriculum and course decisions lack the in-depth consideration that one would like to see in a new and growing field
- Practitioners are generally not aware of the fundamentals of the field, and are making decisions somewhat remote from the fundamentals, thereby engendering practices that are scientifically and economically unjustified, at least from the point of view of a science and a consumer, even if they serve the short-term needs of the industry

- A culture has developed that is self-perpetuating, resistant to change, and intolerant to critical examination
- The active roles are all deficient, therefore any one actor tends to make decisions that are very conservative in terms of expectations; the actors lack confidence in management and in other roles, and adjust their behavior accordingly

What are Some Specific Deficiencies?

Some of the specific deficiencies that are readily discernible in the software field are:

- Methodology that is too complex for the human actors, guaranteeing that they will produce defective products
- Piecemeal methods that have not been linked in a pattern of development that takes the product through a sensible life cycle, rejecting methods that serve one part of the life cycle to the great detriment of the rest of it
- Far too many choices to be made, with far too little beneficial distinctions among the choices, leading to a gross communication-deficient industry
- Job assignments that at one and the same time call for personalities that are very creative and very dedicated to routine, uninspiring work
- A significant gap between management and software developers, preventing any effective check and balance situation
- Failure to use extensively in software development the very computer-oriented assistance that the industry tries to convince its customers they must have
- The lack of any justifiable criteria for managing software development projects

- Failure to define and apply carefully thought out design theory, principles, and methods
- A wealth of ad-hocracy

What Prescriptions are Available?

Given the foregoing remarks, an uninformed observer might reasonably assume that what is needed to rescue this industry is a great deal of scientific invention and discovery. It might be assumed that nothing is in sight that could be put to use effectively to save this industry any time soon from the vulnerability to foreign competition.

While this might reasonably be assumed, it is not true. On the contrary, what is needed to rescue this industry is readily determinable, and most of what is needed could be put in place within three years or so under a management and resource environment dedicated to accomplishing it.

Let us try now to explain why the scientific means are available, for if we can establish this then we can see that the real problem in this industry lies in developing the management attitude to install the scientific basis for good software development.

We do anticipate that the "revelations" we will make will encounter two types of predominant reaction:

- Indifference -- because the incentives to change are not great to the individual actors without a concern for the nation, and because the change agents will have to fight the prevailing culture
- Understructured rebuttals -- because people will see their situations threatened and, if they are not able to see the big picture, will try to attack our views in a piecemeal way

We want to make it easy for these people to attack this paper, for it is only by giving them enough rope to hang themselves that they will be unmasked as bottlenecks that threaten our national economy, just as the labor unions, steel management, and the federal government of the 1950's and 1960's ruined our

steel industry, and just as the auto unions and auto manufacturing management drove our auto industry to the brink of collapse in the 1970's.

To make it easy for the antis to attack, we organize this discussion around a set of numbered topics, with just enough references given to suggest to the serious reader that there is much more to be learned.

POINT 1. The social system involved in software development does not consist of the right set of roles; nor are the people filling the existing roles qualified to provide the necessary leadership.

The development of the software industry and its auxiliary operations such as university computer science departments has been ad hoc, but driven largely by a frantic business situation that is not conducive to thought, but only to action.

While computers are "systems", they can be built by people that mostly know nothing about systems theory, and invent methods and practices without being informed. Moreover, while computers are mathematically based, they can be built by people that know nothing about mathematics.

While software cannot be created without some knowledge of computers, most software has been written by people that don't know much systems theory, don't know much mathematics, don't know how to articulate complex ideas in plain English, lack personal discipline, may have great creativity and no propensity for communication, or may communicate well but have no creativity.

Engineers and physical scientists have made great strides in developing hardware that is reliable. This is why the costs of hardware have been coming down drastically over the years, and the reliability has been going up. And this is in sharp contrast with the software situation where the cost has been going up and up and the reliability remains bad.

The most gross deficiencies in roles are:

- Software designers are ignorant about what human beings can be expected to do well, therefore they create tools or languages for people that violate everything that is known about human intellectual capacity; and this is why we get so much bad software

[Refs. 1 and 2]

[See references 7 and 8]

- More recently, engineers have begun to recognize the great difficulties and fundamental importance of documenting the requirements on software, and to begin to understand the need for eliciting knowledge about these requirements to provide a certain kind of discipline to the overall design approach, but having done so, they are now busily engaged in reinventing the wheel, as though the fields of psychology and management have nothing to say about getting information from people. The entire computer software industry has ignored the work that is well known in other arenas, and which presents a very thoroughly defined and behaviorally sound means of eliciting information, which has been very heavily tested around the nation and been found to be very effective.

  [See reference 9]

- The Department of Defense, as a major buyer of software, has understood the need for some standardization as a way of eliminating much of the confusion due to hundreds of languages, operating systems, conventions, etc. But even this high-resource branch of government has elected to standardize on ADA without even realizing what other important criteria should be applied in choosing a programming language besides simply that of reducing the numbers used. We are seeing here a first-class example of the gap that exists between management and technical decision-making, undisciplined by awareness of what criteria should inform a decision having [redacted] widespread ramifications.

- Recently more attention is being given to a strange quirk of humanity, in which people advocate things to others (especially their clients, students, or customers) but, strangely, do not demonstrate their dedication to what they advocate by following their own preaching in their own practice.

  There is probably no field that could benefit more from the use of computers to assist professional people than computer programming. Now it is true that computers are being used, but the degree of sophistication and scholarship embodied in such uses is almost trivial.

  The same physical workspace that is used to add up check totals is also used to help a programmer deal with writing a system of 10,000 or more lines of software, all of which must be sequenced logically, etc., etc. In other words, there is no matching of the work environment to the cognitive work demands. This is the same kind of "human factors" thinking that went into the design of the Three-Mile Island nuclear reactor controls, described by John Kemeny as 40 years out of date.

  Why can't the computer industry invest in large displays, of the same size as those that are used to control train passenger routing, instead of being permanently wedded to small screens that take their size because of marketing thinking instead of criteria for effective human performance in software design?

POINT 2. Structured programming offers the necessary discipline to make good documentation possible. It then leads into validation of program modules, giving mathematical proof of validity. The latter leads into computer software to assist in structural operations related to structural programming and proof of validity, which takes extensive cognitive burden off the programmer, and which leads naturally into high quality documentation and easy means for program modification.

Anyone who looks deeply into this situation will discover that, in computer science departments, and in practice, people are doing some of this, giving lip service to some of it, and not doing some of it. But it is the integrated system that is critical, not the parts.

Virtually any theoretical question related to the foregoing either can already be answered, or an answer can be developed within a few months.

But the state of the art in the industry is to treat special cases of generic results as unique developments, not blessed by any connection to sound theory, and just to be applied in an ad-hoc way, with ad-hoc practices.

To see how bad it is, look at the overview on programming practices in Reference 10.

To see how, on the one hand, there is recognition of the gap between the state-of-the-art and defense contractor software practices, and at the same time a bias against moving against this gap, look at Reference 11 (the Eastport Report developed for the government Strategic Defense Initiative by a blue-ribbon panel). The panel acknowledges the huge gap, and at the same time in another part of the report, disavows the very basis for correcting it.

REFERENCES

1. G. A. Miller, "The Magical Number Seven, Plus or Minus Two: Some Limits on our Capacity for Processing Information", Psychol. Rev., 63(2), 81-97 (1956).
2. J. N. Warfield, "Structural Analysis of a Computer Language", Proc. 17th Annual Southeastern Symp. on Systems Theory, IEEE, New York (1985).
3. _____, "Developing a Design Culture in Higher Education", Proc. SGSR, Seaside: Intersystems, 1985, 725-729.
4. _____, "Education in Generic Design", Proc. SGSR, Seaside: Intersystems, 1986, H22-H33.
5. _____, "Dimensionality", Proc. IEEE SMC Society, to appear, 1986.
6. _____, "Organizations and Systems Learning", General Systems, Vol. 27, 1982, 5-74.
7. R. C. Linger, H. D. Mills, and B. I. Witt, Structured Programming: Theory and Practice, Addison-Wesley, 1979.
8. James Martin, An Information Systems Manifesto, (esp. Chap. 9), Prentice-Hall, 1984.
9. Andre L. Delbecq, A. H. Van de Ven, and David H. Gustafson, Group Techniques for Program Planning, Scott Foresman, 1975.
10. David King, Current Practices in Software Development, (esp. Chap. 3), New York: Yourdon, 1984.
11. Eastport Study Group, A Report to the Director, Strategic Defense Initiative Organization, 1985.