Int. J. Engng Ed. Vol. 8, No. 6, pp. 419–426, 1992
Printed in Great Britain.

0742-0269/91 $3.00+0.00

Where is Software Engineering in the Technical Spectrum?*

T. F. LEIBFRIED, Jr
Department of Computer Science & Engineering, University of Houston—Clear Lake, Houston, TX 77058, *U.S.A.*

R. B. MACDONALD
NASA/JSC, Research and University Programs & RICIS/UHCL Technical Officer, Houston, TX 77058, *U.S.A.*

With the meteoric rise in the capabilities of computer hardware there is general agreement that software as a discipline has not kept pace. The need for some sort of software design discipline is generally acknowledged but its precise definition is a matter of dispute. Indeed there is some question as to whether 'software engineering' is indeed a valid engineering discipline at all. Here the historical roots of other engineering disciplines are traced and a parallel is drawn to clarify the proper perspectives on 'software engineering'. A set of criteria is constructed and a view of software engineering vis-à-vis *computer science is presented with a system criticality in mind.*

INTRODUCTION

AS SOCIETIES move further into an 'information age' as more people become computer literate and more of us become dependent on computing and information systems, greater concern is necessarily being given to critical complex issues associated with this evolving technology. While over the last four decades we have witnessed tremendous advances in computer hardware technologies, we are increasingly experiencing the adverse effects of critical deficiencies in our abilities to develop the software needed to harness the generic capabilities of our hardware technology.

In the computer journals and tabloids, there have been a plethora of articles written about the software engineering field. But while we are advocates of the need for an engineering approach to software development, we are at once impressed by how many authors have treated the subject of software engineering without adequately addressing the fundamentals of what engineering as a discipline consists of. In addition, the role of supporting sciences to engineering, in the context of the relative roles of computing science and software engineering is not sufficiently addressed. The purpose of this article is not to once more prove that Lord Acton was correct when he wrote, 'What we learn from history is that we do *not* learn from history'. Instead in this treatise, we have attempted to bring together a discussion of the various related facets of this issue in a logical framework to advance the thesis that software development is necessarily an engineering process, and that we need formally educated software engineers to develop our complex software systems of the future.

The purpose of this paper, then, is to examine whether or not the design and development of software for digital computer processing systems should be both viewed and treated as a legitimate field of professional engineering. In addition, we examine the type of academic and professional-level education programs that would be required to support a software engineering discipline. Lastly, we comment on the future role and importance of computer science in the support of software engineering.

BACKGROUND

It is prudent to recall the dangers of what the conventional wisdom portrays as the features of engineering disciplines. It has often been said that engineering is more of an art than it is a science. An art may make use of the *findings* of a science, but it presumably does *not* advance the knowledge-base of the science itself. Others have commented on the 'cookbook recipe' approaches of engineering practitioners from some decades earlier. These approaches were the common views of the conventional engineering disciplines prior to World War II when major changes in technology were still relatively slow-paced. This all began to change in the 1940s with the stimulus provided by World War II. There is an interesting anecdote about Dean Terman's stint as a founder of the Lincoln laboratories which was credited with developing advanced and somewhat novel concepts in radar

* Paper accepted 4 January 1993.

during and after World War II. When Dr Terman first sought technical personnel for Lincoln Laboratories, he brought in engineers. They were to develop new experimental equipment to implement the concept of radio ranging. Much to Terman's dismay, he observed that most of his engineers seemed to lack adequate backgrounds to develop the new innovative designs required, but tended to know only how to design and build equipment which was quite similar to that built before. He observed that physicists on the project were generally better able to do the innovative design-work that he thought of as 'engineering' work. Later, this and similar experiences of others during this era, had a profound effect upon the thrust of engineering education. In particular, electrical engineering experienced a most profound revision as a result of the 'explosion' of new and advancing technologies after World War II. David Parnes, an electrical engineer, commented on Dijkstra's work, 'On the cruelty of really teaching computing science' that 'My own engineering education included more courses in the Mathematics Department (taken side-by-side with mathematics students) than courses offered by the Electrical Engineering Department' [1]. What he has eloquently established is the 'new' (and perhaps over-reactive) philosophy engendered by Dr Terman's experiences; as we have moved into a fast changing, high technology culture, it is no longer sufficient for an engineer to be capable of doing what has been done before. They need to be well-versed in fundamentals of the theories derived from the supporting sciences, be well-versed in the appropriate mathematics, and be capable of extending the derivation of the principles and formulae to apply to new situations. 'Recipe engineering' proved not to provide sufficient flexibility, especially in an age of fast changing technology such as we are experiencing in the computer field.

WHAT IS THE CONVENTIONAL WISDOM?

The emergence of the recognition of the need to treat the software development process as an engineering process, and the advent of the emergence of the first software engineering curriculum, have some similarities to the past.

In 1989, an article relating to software engineering curriculum quoted a Taxonomy of Educational Objectives [2]. This taxonomy declared that students of software engineering needed knowledge of computer methods, classification and abstractions but not any deep understanding of them. We think this oversimplifies the problem, as breadth, without depth, is not enough.

Just as the first electrical engineering curriculum evolved from physics, we see the software engineering curriculum evolving primarily from that of computer science and to a lesser extent from electrical engineering. The conventional wisdom of software engineering, therefore, has a tendency to be more biased by the tenets of science and mathematics and, in particular, those of computer science, than by those of engineering. However, there is a question about computer science as it is currently studied, as we shall see.

Software engineering practices and supporting educational curriculum can benefit from drawing more heavily on the conventional wisdom of modern-day engineering. This approach calls for a clear understanding of the important complementary differences between engineering and science [4]. It is also beneficial to keep in mind that the transition from a 'practitioner' of science to engineering, and vice versa, is not usually a trivial one. The practice of a disciplined engineering approach to the design of a new, or better, product is quite different from the scientific practice of defining a research experiment to discern new knowledge. To compound the difficulty, often a difference in the temperament and motivation of engineers and scientists exists. In making a leap to engineering, it is critical to keep in mind that engineering is a 'design discipline'. As such, engineering design methodologies lie at the heart of engineering. While their application can benefit from 'tools,' without well understood design methodologies, tools are fairly useless.

The early history of software development practices has meant that conventional wisdom has a tendency to view software engineering as programming, and that, at most, a proper set of tools and appropriate management practices are sufficient to overcome the critical issues of complex software development [5].

In addition to overcoming such common-wisdom views as these, we have considerable evidence that software engineering and its associated methodologies are going to be as fast-changing as those experienced in other engineering fields over the past 50 years. This indicates that software engineers are going to need a rich background in a variety of supporting subjects if they are to be able to stay up to date in the field. This will be true for both those who engineer future 'systems' software as well as the developers of 'applications' software.

IS SOFTWARE ENGINEERING AN ENGINEERING DISCIPLINE?

What we seek to show is *not* that software engineering is a 'doomed discipline' a view espoused by Dr Dijkstra, but that it needs to address the subjects of computer science and engineering from a formal methods point of view, as well as a so-called practical viewpoint. Mathematical modeling is as essential to the software development process as it is to other engineering fields. It is also not desirable for software engineers to continue to design software that requires all of its parts to be newly created in order to implement the design. The use of properly designed and verified

(i.e. engineered) 'component parts' in software engineering is as critical to the success of future software development projects as it is in other engineering disciplines. Just as in conventional engineering, the software designer needs to take advantage of existing proven component parts wherever possible. We believe that it is at least now clear that the products of software engineering, for example, design specifications, software system designs, attendant coded implementation packages, and testing, are in fact, representative of the products of engineering [5]. The resulting stored instructions in a computer system's hardware memory elements are just as real as the 'electrical components' that do the work of the design in any electronic system.

Science and engineering are different disciplines (professional fields of endeavor) that are, nevertheless, significantly related and mutually supporting of one another. In a fundamental way, science supports engineering since the utilization of scientific theory provides the foundations of the engineering disciplines. It is interesting to note that engineering also supports science, in that the engineering of scientific instruments, components and materials, etc., is critical to advanced research in the sciences (physics, chemistry, etc.). Clearly during the course of one's profession, it is not unusual for scientists to engage in engineering activities and engineers to engage in science. However, these two endeavors are significantly different and should be recognized as such.

WHAT IS SCIENCE?

The quantitative sciences of physics, chemistry, meteorology, astronomy, as well as certain subfields of geology and biology are now generally viewed as separate scientific disciplines which have somewhat fuzzy, difficult-to-define, boundaries. The sharp lines of demarcation of these fields have become less rigid as fundamental unifying principles have been discovered which tend to unite them. An example is the underlying discoveries of atomic physics which have had a profound effect upon the concepts of chemical bonding. The boundaries between chemists and physicists are often blurred.

Scientists are the professional practitioners who are educated and trained in the use of so-called 'scientific methods' to produce new knowledge in the various fields of science. This ever-advancing knowledge is comprised of collections of facts, classes of models and generalizations which systematize and correlate observed facts to support predictions of phenomena that can be compared with later observations or experimental results.

Scientific theory is in a nearly continuous state of advancement. Historically, we have seen nearly all the once-accepted scientific conceptual schemes of the past disproved by increasingly innovative experimental and observational techniques and replaced by either modified or new theories.

WHAT IS ENGINEERING?

Engineering is the practice of applying the existing bodies of scientific knowledge to develop new capabilities or to develop more cost-effective and/or reliable products and services to improve already existing facilities or to generate new ones. Engineering combines the use of knowledge together with the 'art' and know-how which has been built up and passed along over time by the body of practicing professionals.

This is not to say that engineers do not require the analytical ability possessed by scientists. There is no way that an engineer could have developed the technology which has played a major role in the advancement of science without having an appreciation of the phenomena that technology was to observe. A clear understanding of underlying theory is critical to its application in engineering. Engineering may be viewed as similar to the applied sciences, in the same way that the practice of medicine is applied biological science. Physicians must understand and appreciate the biological findings before they can apply them intelligently.

WHAT IS THE DIFFERENCE?

The approaches of science, the so-called scientific methods, are substantially different from the more commonly practiced approaches of engineering. Scientific methods all start with a collection of facts, followed in turn by considerable thought, speculation and conjecture. The result is the development of alternative working hypotheses that can be tested and evaluated. Hopefully, these hypotheses lead to generalizations of predictive theory.

A well understood example of this process is the historical sequence leading to the development of Newton's second law. The astronomical observations of Tycho Brahe were followed in turn by the development of Kepler's laws which captured the relationships among Brahe's facts. Then, the generalizations of Newton's principles culminated the process with their inherent broader capabilities of prediction.

On the other hand, engineers are practitioners who are educated and trained principally in the comprehension and use of knowledge, empirical evidence, and more importantly, disciplined problem-solving methods, to achieve solutions to technical problems and produce new and/or improved capabilities [4]. Commonly accepted engineering approaches generally begin with the identification of the need for a new or improved capability and this is often described in terms of requirements. These are in turn translated into specific product or system specifications. Analy-

tical and physical modeling techniques are used to analyze and develop a design. To be useful, a resulting design must be such that it can be implemented with existing materials and processes or alternatively with materials or processes that can be developed. In the next step, the design is fabricated and tested to determine how well it meets the initially determined requirements of the design and implementation specifications. An economic or 'business' decision may then be made as to the viability of the product. If it is found to be useful and cost-effective it is then further engineered for manufacture. An example of this process is the development of the diesel engine by Rudolf Diesel. He had hoped to design and develop an internal combustion engine which approximated the ideal thermodynamic Carnot cycle. When the prototype design literally blew up during testing, he redesigned the engine to use a less efficient but more practical cycle for which materials and fabrication practice were available. The ideal cycle performance requirements were relaxed and the rest is history.

WHAT IS COMPUTER SCIENCE?

With the general definitions thus delineated, it is clear that some practitioners of computer science already are practicing some sort of engineering, or at least applied software science. Not all computer scientists are searching for new algorithmic knowlege continually. Many are developing software systems to accomplish specific purposes. Is not the design of a data base system an application of software design and development?

WHAT DOES ELECTRICAL ENGINEERING HAVE TO DO WITH SOFTWARE ENGINEERING?

Electrical engineering is that branch of engineering primarily responsible for designing and developing the hardware portions of computing and information system products, and systems. The sub-field of computer engineering has been established under electrical engineering just as those of power, communications, radio, television, etc.

It is interesting to note that the field of electrical engineering is some 150 years old and was initially established on the basis of the needs and opportunities suggested by the theories of the physicists and mathematicians of that time. The first 'electrical engineers' were physicists who became personally interested in going 'outside' the sciences to design and build the first products promised by the newly-found scientific theory. Over 150 years, the discipline of electrical engineering has gone through a metamorphosis and become formalized and marked by the use of 'engineering approaches'. The first curriculum in electrical engineering in the US appeared in the early 1880s [6]. Today, electrical engineers are educated within accredited university schools of engineering and enter the workplace in a continuing process. The specialties in computing hardware systems arose in electrical engineering during the 1960s. One of the major outcomes of their engineering progress was the development of general-purpose digital computing systems having a generic capability to handle and compute data. This concept of 'general purpose' required that these generic systems be 'programmed' to accomplish the varied specific purposes desired. These programs became known as 'software' and the practitioners who developed these programs came to be called 'programmers'. Programmers came from many different fields and had widely differing backgrounds. Initially, programming was primarily an art, and program development methods were, for the most part, left as an individual practice. This 'artisan' approach was reasonably satisfactory until the late 1960s when software applications and systems began to become increasingly complex.

LO! THE COMPUTER SCIENTIST!

In the 1960s, the first departments of computer science were established in academia. Initially, these concentrated on mathematical techniques necessary for numerical analysis. The development of non-numerical programming algorithms and theory, and the development of 'higher level' computer languages followed.

By the late 1960s, electrical and other engineers were producing computer hardware systems of considerable capabilities. More importantly, there was the promise of new systems having much improved capabilities with significantly reduced costs in the near future. Computer engineering had been born as a viable branch of electrical engineering. However, the state of the software was not keeping pace with the hardware.

Computing science departments became the primary source of people who went into the 'programming field' and were the professionals faced with solving the so-called 'software crisis' as the software problem was first termed in the late 1960s.

THE SOFTWARE ENGINEERING COMETH!

During the late 1960s, the idea of the need to 'engineer' software developed and matured. First, the special-purpose digital computer, and then the general-purpose computer showed that software systems could indeed replace some special purpose hardware. A programmable machine could perform the function of a hard-wired circuit. An early example of this was the implementation of a phased lock loop for an aircraft tracking system through use of a software program running on a PDP-11 digital computer. This type of implementation

punctuated the dissolution of the lines of demarcation between electrical engineering and computer science.

During the period of the 1970s and certainly by the mid-1980s, the idea and various descriptions of a 'discipline of software engineering' were advanced. By the end of the 1980s, academia had produced the first academic programs of software engineering at the masters level.

Software engineering is the missing branch of engineering needed to design and develop the software systems required to run the hardware products and systems. In a sense, software engineering is to what is conventionally called computer science, as production engineering is to design engineering. (In a very real sense in the future, software engineering needs to be to computer science what electrical engineering has been to physics.)

The basic aspects of software engineering [4] which need to be understood are:

- technical;
- managerial;
- legal;
- security.

The technical aspect includes methods and tools for:

- deriving system (product) specifications;
- designing products and systems;
- building or implementing designs;
- testing (verification and validation) of products and systems;
- specifying the operation and maintenance of these products and systems.

The legal aspect includes: basic legal considerations which are needed to survive in business, and protect product rights, as well as specific processes associated with conducting a business including:

- software program protection (patents, copyrights, trade secrets);
- contract law;
- negotiations;
- tax considerations;
- labor law;
- warranty of software performance;
- ethics.

The security aspect includes:

- concepts, the theory of, and need for, security in differing software systems;
- tools for generating secure software;
- questions of individual privacy and methods of preventing the compromise thereof:
- methods for measuring system security.

The management aspect includes:

- the need for a thorough insight into those basic environmental elements that a manager must control to attain smooth, effective and efficient product development which will have desirable operation- and user-interface characteristics over the product life cycle.
- management tools and techniques of importance in the design, development and sustaining the engineering of products.

WHAT ABOUT THE CURRICULUM REQUIREMENTS?

The engineering qualities of importance to a software engineer which need to be addressed in any educational program fall into three main categories: basic knowledge, skills and attitude:

- basic knowledge comprised of fundamental and applied science;
- fundamental knowledge of physics, chemistry and mathematics which provide the foundation upon which all knowledge is assembled;
- knowledge of applied sciences consisting of computer software and hardware basics and practical network theory, and including knowledge of other engineering disciplines such as practical thermodynamics which bridge the gap between the knowledge base and engineering design;
- basic skills such as good communication, judgement skills, logical and orderly decision processes, good analytical capabilities and a concept of teamwork;
- attitude, which is hard to teach but necessary for effective engineering reflecting a curiosity, a questioning attitude, objectiveness and reasonableness.

The general steps of the engineering method must not be overlooked. The principal steps are:

- problem formulation;
- problem analysis;
- search for alternative solutions;
- decision;
- specification preparation;
- implementation;
- test and evaluation.

WHAT ARE THE OBJECTIVES OF MODERN ENGINEERING?

Whereas the objectives of science are to discover and create 'knowledge', the objectives of engineering are to create new and improved capabilities which support man in his endeavors. Therefore, while engineers need to be well versed in the underlying sciences, they also need to be well versed in the resulting techniques of engineering analysis for a variety of physical phenomena. Only in this way, can an engineer hope to create related new and improved products.

WHAT THEN SHOULD BE THE FUTURE OBJECTIVES OF COMPUTER SCIENCE?

Freed of the pressure to produce computer scientists capable of serving as 'engineers', the computer science community can, in the future, focus upon creating a body of fundamental, theoretical, scientific and mathematical material upon which advances in software engineering can be solidly built! We assert that such a knowledge is critical to important advancements in future software design processes and to the development of an engineering capability to verify important characteristics of a proposed design, or to determine the trade-offs among alternative designs. Such a capability is essential for achieving and maintaining the reliability and correctness of a design, which other engineering disciplines provide for their clients and to which society has come to demand. No longer do engineers build structures on the basis of rough 'sketches' and then have to resort to testing to discover fundamental design flaws followed by major and empirical reconstruction. Important work by computer scientists is needed to advance the concepts of, and to provide the underpinnings of, 'provable software design'. Other advances are needed to support better simulation methods especially for real-time systems simulation, faster artificial intelligence systems and expert systems. Indeed, some of these advances may be closely linked to developments in the science of parallel processing.

HOW CAN THESE OBJECTIVES BE OBTAINED?

An engineering curriculum must expose the student to a wide cross-section of different fields while providing specialized training in a particular engineering specialty. As an example, consider that an electrical engineering student takes courses in a variety of sciences (physics, chemistry, mathematics, etc.) as well as from the various classical engineering disciplines (mechanical, civil, chemical, etc.). In addition, there are specialized courses in the electrical engineering field (communications, control, power, electronics, etc.).

The standard undergraduate engineering curriculum is now usually regarded as a five-year program providing this broad background in engineering and fundamental supporting sciences. Many of these programs approximate or exceed 160 credit-hours. Graduate degree programs lead to a more in-depth education in a selected area within a particular engineering field.

Software engineering appears to be generically similar to other engineering fields in that it builds upon the knowledge of supporting sciences. It is involved with engineering new capabilities stemming from the development of computer software products and systems. The engineering of these new systems is also predicated on a number of related engineering disciplines depending upon the nature of a particular specialty. For example, the engineering of a software system which implements a complicated automatic control system. Thus a software engineer, who plans to develop large control systems, needs to understand the mathematics of control and the physics of the so-called 'plant' which is controlled.

SO WHERE ARE WE NOW?

The state of development of the fields of software computer engineering at this time may be compared to the states of already existing engineering disciplines at the time of their emergence. In the absence of academic and/or professional educational programs, the early practitioners of an engineering field came from a variety of related fields with considerably different specialized training.

The first practitioners of electrical engineering, some 150 years ago, were physicists, chemists and others with little formal education in any engineering methodology, and only the practical experience of their laboratory work. In addition, few, if any, standards existed to guide engineering development. During this period of transition, much was done in an *ad hoc* fashion. While the other engineering fields each followed a somewhat different course of development, in detail, their development followed a similar pattern. Early 'trial and error approaches' followed by less qualified practitioners resulted in designs which were inadequate from the viewpoints of reliability, cost and often safety. In each field, professional approaches based on good engineering practices and the educational study necessary for them were gradually adopted. It was only at that point of advancement that 'costly mistakes' could be avoided by early detection through modeling and engineering analysis. This is an interesting precautionary point: software engineering must imply an approach to this vocation which would be recognized by other engineering professions as having the basic characteristics typical of those other conventional fields [3]. Systematizing work, tools and the application of management principles to the organization of work, useful as they may be, are *not* the essence of engineering and are not sufficient to transform our occupation into a professional engineering field.

By comparison, in almost all of the curricula in software engineering we have reviewed, the broad background necessary for the practice of engineering is absent. What most seem to do is to structure courses in specification writing, production management, and testing of large software systems, regardless of the nature of the system itself. With this direction, it would seem that the source of students for the masters in software engineering should come from technical areas other than computer science, but with perhaps some experience of computer studies.

SO WHAT SHOULD WE DO?

It is our hypothesis that the software engineering discipline is a new engineering field in the early stages of transition from science and practice to a professional engineering discipline. To make it viable there should be care taken in selecting applicants with the appropriate backgrounds which will lend itself to complement the training which the current software engineering curriculum can offer. The structure of the curriculum must contain the fundamentals of computer science, just as the other engineering curricula contain physics and chemistry. In addition, the principles of software system design must be studied with the appropriate laboratories included. As a by-product of this, computer scientists will be free to do the important work of extending the knowledge base that is critical to important advances in the field of software engineering.

SUMMARY AND CONCLUSIONS

If software engineering is to become an accredited and operative field of engineering, then a first step is for schools of engineering to commit to the development of an undergraduate curriculum that provides for a background in the standard natural and applied sciences, in mathematics and associated computer sciences. Importantly, the undergraduate curriculum needs to provide the standard cross-section of introductory course-work common to all undergraduate engineering programs.

This would include the standard and much needed background in mechanics, strength of materials, fluid flow, thermodynamics, electrical and electronic systems. In addition, software engineers would benefit from introductory courses in business and management. Such a program would provide software engineers with an exposure to business applications that would prove useful in the engineering of 'management information systems'. Courses in technical writing, economics, engineering law and ethics would be candidates to extend the engineering background of the undergraduate software engineering program. Secondly, engineering schools need to develop master's level programs that allow excursions of considerably more depth in specific specialty areas, such as operating systems modeling, simulation, networks and distributed systems, system optimization and software design methodologies. In addition, the master's level would have a core which includes at least one course in advanced linear systems.

Lastly, doctoral programs need to be derived to produce individuals with backgrounds suitable for participating, and leading needed engineering research in the important frontiers of future software engineering topics. Again, this can be observed to have been critical to the development of the already established, mature engineering disciplines. Clearly, software engineering researchers are going to be critical to the development of engineering methodologies required to design ever-increasingly complex, sophisticated, reliable and cost-effective systems of the future.

The current trend appears to be that the first software engineering programs appear at the master's level. Often students are not required to have an undergraduate engineering degree and often have not even had an introduction to generic engineering goals, methods, ideologies, etc. These master's level programs often take on more the form of advanced computer science than advanced engineering. It is the authors' contention that until academic institutions develop more suitable 'full-range' educational opportunities in software engineering, students will continue to leave our colleges and universities with educational backgrounds not good enough to serve them well. 'Full-range' educational progams in software engineering and computer science will enhance both the practice of computer science, that is successfully advancing the state of underlying knowledge and theory needed, and the application of common engineering practices in the design and development of the complex software systems required by current and future applications.

The authors would point to the field of chemistry as a good example where precedence for success has already been established. Chemical scientists and chemical engineers with their different academic training are equipped to advance the theory as well as practices to produce solutions to complex chemical applications.

It is the authors' opinion that rather than some type of crisis, we have a well defined and continuing need for software engineering professionals to complement the pursuits of our computer scientists and computer (electrical hardware) engineers through the engineering of the 'heart' of future computer and informational systems. We cannot afford to place the responsibility for the design of safety- and mission-critical systems to unqualified and improperly educated personnel. Cases like the Theric 25 radiation disaster caused by a software malfunction can be avoided by proper software engineering [7].

REFERENCES

1. David L. Parnas, A reply to Edsger W. Dijkstra's 'On the Cruelty of Really Teaching Computer Science', *CACM*, **32** (12), 1405 (1989).
2. Gary A. Ford and Norman E. Gibbs, A master of software engineering curriculum, recommendations from the Software Engineering Institute, *IEEE Comp.*, **22** (9), 59 (1989).

3. Robert L. Baber, *The Spine of Software, Designing Provably Correct Software: Theory and Practice*, p. 14, Wiley (1987).
4. Randall W. Jensen and Charles C. Tonies, *Software Engineering*, pp. ix, 10, 13, 14, Prentice Hall (1979).
5. C. R. Vick and C. V. Ramamoorthy, *Handbook of Software Engineering*, xii, xv, Van Nostrand Reinhold (1984).
6. Education: the challenges are classic, origins of the issues, *IEEE Spectrum*, November, p. 38 (1984).
7. J. Jacky, Programmed for disaster: software errors that imperil lives, New York Academy of Sciences, September/October 1989. (As referenced in S. L. Pfleeger, *Software Engineering*, p. 2, Macmillan (1991).

T. F. Leibfried, Jr is an Associate Professor in the Computer Science and Engineering Division at UHCL in Clear Lake City, Texas and has over fifteen years of experience in academia (since 1975) and prior experience in the US electronics industry. He holds the degree of Ph.D. in Electrical Engineering from Rice University, and the degree of MS Engineering Physics from University of Virginia as well as an MBA from Stanford and an ME from Stevens Institute of Technology. Recently, he participated in a program to bring computer engineering and software engineering programs to UHCL. Further, he has been a computing researcher for the NASA Johnson Space Center for the past six years and is currently Chair of the Mission for Safety Critical Systems for UHCL Research Institute for Computing and Information Systems (RICIS).

Robert B. MacDonald has over thirty years of experience with industry, academia, and government in the development of computing systems and their applications. After graduating from Purdue University's Electrical Engineering program, he worked as a research engineer for the IBM Corporation in the development and applications of aerospace computing systems in the late 1950s until 1966. In 1966, he accepted an appointment with the School of Electrical Engineering at Purdue University to direct a national program focused on the use of computer processing of remotely sensed Earth resources measurements. In 1971, he took a position at NASA's Johnson Space Center to direct the Center's Earth Resources Technology Satellite Program. Since 1984, the author has been responsible for the development and management of a NASA co-operative program of computing and information systems research and education. His professional experience has led him to become an advocate for the development of software engineering programs in academia and he has been an active supporter of the newly approved software engineering degree programs at the University of Houston–Clear Lake.