Fermi National Accelerator Laboratory

FN-528

Human-Computer Systems in the Military Context

Thomas Nash
Fermi National Accelerator Laboratory
P. O. Box 500
Batavia, Illinois 60510

and

Center for International Security and Arms Control
Stanford University
Stanford, California 94305

December 1989

Operated by Universities Research Association Inc. under contract with the United States Department of Energy

Human-Computer Systems in the Military Context

Thomas Nash

Computing Division
Fermi National Accelerator Laboratory
Batavia, IL 60510

and

Center for International Security and Arms Control
Stanford University
Stanford, CA 94305

December, 1989

For all the time and frustration that humans expend interacting with computers, it is surprising that more attention is not paid to the impact the computers used by armed forces might have on security and peace. The military, like the rest of us, is dependent on its computers and interacts with them unceasingly, often in critical situations. Never before have our armies been so intimately dependent on their tools, certainly not ones as complex, or as apparently capricious as misapplied computers can be. This paper looks at the ways in which this peculiar human involvement with computer systems requires special attention, most critically in the military context, from policy makers and researchers.

By any measure, computers have infiltrated every aspect of modern armed forces. In weapon systems, computers and digital electronics are used for communications, navigation, weapon-path prediction and control, and for target identification, tracking, and projection. Computer systems are at the heart of all activities in command, control, communications, and intelligence (C^3I), including, increasingly, data management and interpretation. Management of military logistics depends on the same kind of extensive computer support found in any large-scale civilian activity. The total electronics budget for the Department of Defense (DoD) in 1988 was about $55 billion. For 1987, $22 billion was budgeted on C^3I equipment[1] – and this sum does not include the substantial, secret, "black," budget in this area. The importance our society implicitly attaches to military uses of computers is dramatically evident in the military impact on civilian computing-technology research. About 70 percent of government-sponsored computer science at universities in the United States is now funded by the Department of Defense.[2]

From the strictly parochial American viewpoint, computer and digital systems embody much of what gives the nation's armed forces a technological edge. For one thing, these systems are beginning to make available increasingly precise, so-called surgical, weapons for both regional and strategic situations, at a reduced risk to allied personnel. As this technology matures, it should expand military options and allow for planning a more finely tuned and smoother pace when the escalation of hostilities are required during a crisis. Technological leadership in this area could enhance a nation's ability to exert political influence in a crisis.

Information about the status and activities of both friendly and hostile forces has always been essential to success on the battlefield. The speed of modern mechanized and aerial warfare has enormously reduced the time available for gathering, disseminating, and assimilating

---

[1] Office of the Under Secretary of Defense for Acquisition, *Report of the Defense Science Board Task Force on Command and Control Systems Management*,Washington, D.C.,1987), 10; see also Editors of Defense Electronics,*The C^3I Handbook*, (EW Communications, Palo Alto,1986), 263.

[2] Joel Yudken and Barbara Simons, *Federal Funding in Computer Science: A Preliminary Report* , (IBM Research, Yorktown Heights, N.Y., 1988).

information, and for decisively acting upon it. Computer management of information is, therefore, essential, and effective implementation of computers may well provide the critical advantage on the battlefield. Similarly, the time scale for logistics support, which also requires widespread computer tools for its management, has been sharply reduced.

American leadership in military computer technology is looked at with considerable envy by others, whether superpowers or developing countries. Computers and digital technology will become a ubiquitous factor in all of the world's military activities. The special kinds of problems I discuss in this paper will, therefore, increasingly concern those who worry about the peace and security of the world.

The complexity of systems with embedded computers, and their widespread heavy use, is at the core of a major qualitative change in military process.[1] The most telling aspect is the way in which modern computer and digital-electronics technology has widened the information horizon. This change amounts to a phase transition in which the domain of immediate communication has expanded from the commander's own bunker to the whole battlefield, the regional theater, and, in essence, to the whole world. The result is a tight coupling of widely spread decision-making and action units. During a crisis or in battle, the information that computer systems make available can greatly help in avoiding mistakes that can result in serious, immediate loss of life, or escalation to even more disruptive levels of warfare. But, when mistakes are made, they can resonate widely. When failures occur, the common mode can be disastrous, creating chaos with consequences so widespread that von Clausewitz could never have conceived of them. Unreliable systems in such circumstances are unacceptable.

The modern military infrastructure has the clear potential, particularly if misapplied, to amplify von Clausewitz's famous "fog and friction" in war. To take a simple example, battle analysts confronted with relatively complete information will show some variation in their perceptions of a particular threat. The spread in the distribution of their threat evaluations will increase as information is lost in the fog of an unreliable or ineffective computer system. More will evaluate the threat as very high or very low. Such a spread can obviously result in inappropriate and dangerous escalation. (Risk/threat perception by humans is very complicated and the subject of psychological research, some of which is relevant to this paper's discussion of human-computer interfaces.)

The goal of this study of the military use of computers Is to identify key technical problems that could lead to serious instability in crises. The emphasis is on tactical systems and the potentially serious consequences of their misapplication or misuse. Many of the problems are

---

[1] The significance of the heavy use of computers by the military is not widely appreciated, despite the efforts of Computer Professionals for Social Responsibility and others to bring some of the issues to public attention.

common to strategic command and control which naturally has gotten more military and academic attention.[1] Because of worldwide operations and the global information horizon, unreliable tactical operations, though primarily nonnuclear, are potentially as destabilizing and dangerous as unreliable strategic operations.[2] Computers are embedded in "all critical [tactical] systems – logistics, intelligence, surveillance and warning, communications, command and control, weapons guidance, training, etc." Many aspects of tactical operations cannot function effectively without computer networks.[3]

Very disturbing is the widespread confusion, among both military and civilian decision makers, about what tasks computers realistically can be expected to accomplish and on what time scale. This matter is disturbing no matter whether one looks at it from the point of view of military effectiveness, from a concern about arms control and stability against crisis escalation, or from an interest in the economic strength of the United States and the proper application of the nation's public funds and human-talent resources. Unrealistic application of computers leads to fundamentally unreliable systems. For this reason, I also attempt in this paper to provide some guidance on what tasks computers do well and what tasks it would take extraordinary (or infinite) expenditures of resources to accomplish with them.

Computers are fundamentally different from other technology used by civilian and military sectors of society in two basic ways: the degree of system complexity and the level of human involvement. Measured in terms of the numbers and dynamics of their internal states, computers are systems of extreme complexity. This complexity carries some of the apparent opaqueness and unpredictability that is characteristic of the human personality. The sometime resemblance to human capabilities is what research in artificial intelligence (AI) tries to exploit. Computer technology is the highest of high tech. Yet, unlike other technology, it is extremely difficult to quantify. What do you mean by complexity when you don't know all the system states? What do you mean by reliability when you don't know all (and, probably, not even most) of the failure modes?

Also unique is the degree of intimate human involvement with computer systems. The human component is obvious and essential for specification, programming, definition of input, interpretation of output, and attempts to confound the system. Humans are involved in some of

---

[1] For a complete discussion of strategic C^3I, see Charles A. Zraket, "Strategic Command, Control, Communications, and Intelligence", *Science* 224 (1984):1306-11, and P. Bracken, *The Command and Control of Nuclear Forces* (New Haven, Conn.: Yale University Press, 1983).

[2] The important cause and effect relationship between having effective command and control systems and appropriate response was emphasized in (Office of the Under Secretary of Defense for Research and Engineering, *Report of the Defense Science Board Task Force on Command and Control Systems Management*, Washington, D.C. 1978), 6; see, also Gary Chapman , "The New Generation of High-Technology Weapons," in D. Bellin and G. Chapman, eds., *Computers in Battle, Will They Work?* (Boston: Harcourt Brace Jovanovich, 1987), 93-94.

[3] Charles A. Zraket, private communication, July 31, 1989.

these activities in other enterprises, such as the development of weapons, bridges, and vehicles, but never with anywhere near the intensity required for computers. What is different is that computers aid the intellectual rather than mechanical functions of humans. These are attempts to replace our most complex and "human" capabilities. Their specification and design require a deep and most intrusive self-analysis. This requirement for intensive human involvement is a profound and neglected aspect of the computer problem.

Computer systems must be considered as part of the overall system in which they are embedded, not as one "of a collection of components"[1] or as "appliqué".[2] In particular, it is essential that they be understood as human-computer systems. When something goes wrong, it is a failure in "integrating a complex, highly contingent world of analog variations with both human subjectivity *and* discrete-state machines," as Gary Chapman puts it.[3] The human-computer interface needs particular attention in critical situations requiring rapid decision making. Computers must be accepted as the useful but limited tools they are, not as some kind of allied organism with the all-too-common anthropomorphic qualities people attribute to them.

This report covers selected key aspects of human-computer systems. The starting point is an attempt to explain the extensive difficulties associated with improving – or even defining – the reliability of computer systems. Following is a discussion of hope and reality in the much-emphasized research field of artificial intelligence. The next section describes the importance of human interfaces with computers; and the one that follows flags a particularly acute matter, the security of networked computers, and the severe implications for stability in crises. The final section provides conclusions and a number of recommendations about important areas of research and policy related to human-computer systems.

---

[1] This point was emphasized in the SDI context in *Report of the Defense Science Board; Task Force Subgroup on Strategic Air Defense (SDI Milestone Panel)*, Office of the Under Secretary of Defense for Acquisition, Washington, D.C ., 1988), 3.

[2] The Eastport Study Group, "Summer Study 1985," in *A Report to the Director of Strategic Defense Initiative Organization* (Washington, D.C., 1985.)

[3] Gary Chapman, personal communication, August 10, 1989.

Reliability of Computer-Based Systems

Computer *hardware* is subject in many, but not all, ways to the standard techniques of reliability engineering.[1] Redundancy has been exploited to produce extremely reliable duplex processor machines for situations requiring high availability, such as banking, airline reservations, and telephone switching. Tandem Computer Corporation specializes in such systems, and AT&T's 3B20D (et seq.) computers are widely used for switches. Their downtime, planned and unplanned, is less than two minutes per year in practice.[2] The requirement on the development, sponsored by the Federal Aviation Administration (FAA), of an automated flight-control system, including its emergency mode, is for less than three seconds down time per year.

It will most likely be possible to build isolated networked systems to similar reliability standards with appropriate redundancy. This expectation is reasonable, despite some concern about unpredictable chaotic effects, which have been observed in experiments involving asynchronous networks with nonlinear scheduling. Most important, it is possible to measure hardware reliability in terms of mean time to failure (MTTF) and to predict MTTF from measurements on components for a system in which the overall complexity is kept under control. This process is what allows one to design for improved reliability with redundancy, component selection, and reduction of component count.

In computer *software* the situation is entirely different. Hardware performance and reliability have improved in the last 25 years by factors like a million and a thousand, respectively. There has been barely an order of magnitude improvement in any performance or reliability aspect of software – despite extensive research directed at what is ominously referred to as *The Software Problem*. Why is this the case? We use computers because they handle massive amounts of data that they can manipulate with data-dependent paths through complex logic. Convoluting the data states with program paths leads to a huge number of internal and final states. No human being can specify all the states for any but the most restricted applications. No human programmer can conceptualize all the states. No human tester can test all the states.

Why can't we test software until it can be certified bug-free? Because, as E. W. Dijkstra said, "Testing can show the presence of bugs, never their absence."[3] At the beginning of his book on software testing, Glenford Myers provides a simple homework assignment for his readers. The idea is to point out how difficult it is to test fully software written to even the simplest of

---

[1] Daniel Siewiorek and Robert Swarz, *The Theory and Practice of Reliable System Design*, (Bedford, Mass.: Digital Press, 1982).

[2] W. N. Toy and L. E. Gallaher, "Overview and Architecture of the 2B20D Processor," *The 3B20D Processor and DMERT Operating System*, (Bell Laboratories, Naperville, IL) No. 2, (April 1983), 8.

[3] J.Dahl, E.W. Dijkstra, and C.A.R. Hoare. "Notes on Structured Programming," *Structured Programming* (London: Academic Press, 1972), 6.

specifications. The spec: *read 3 integer values which are the lengths of a triangle's sides; print a message stating whether the triangle is scalene, isosceles, or equilateral.* What test cases would *you* define for a program written to this spec? Experience with real bugs for this example suggest that the test cases must satisfy at least 14 issues. (Did you check the case (0,0,0)?) A group of highly experienced programmers averaged only 7.8 out of 14.[1] This problem is so simple that formal logic methods could easily be used to prove the program carries out its specified task. The code required is a few tens of source lines. For large problems involving several million source lines of code (megaSLOCs), formal methods are impossible, and even highly experienced programmers would be able to define tests that would catch only a tiniest fraction of possible bugs.

Everything said here about software applies to firmware. Firmware involves data, program instructions, or logic that is fixed in the memory or the wiring layer of an integrated circuit. The name firmware applies to read-only memory (ROM), programmable logic arrays (PLAs), gate arrays, and a large variety of other devices complex enough to contain extensive information. Generally, this information is prepared in a variant software language. What goes into one chip may consist of thousands of lines of code. The problem of firmware may be even more insidious than the problem of conventional software. Firmware is so intensively used in designing digital systems like computers that increasingly one sees hardware systems whose functionality traces almost entirely to software programs burned into silicon. The problem goes beyond firmware, strictly speaking. Large portions of VLSI chips, which dominate most digital hardware, are now defined in hierarchical design languages (HDLs) and then "logic-synthesized" or "silicon-compiled" automatically into the diffusion mask set used for the chip's fabrication.

Redundancy works for computer hardware and for noncomputer systems; why not use it for software? The problem is, what do you make redundant? Running two identical copies of software (on reliable hardware) will give the same result, right or wrong. The failures in software are those of the humans in this aspect of the human-computer system. So what about redundant human programmers? The idea is referred to as n-version software development.[2] Several independent and isolated teams build software to the same specifications. It is run in separate computers, and the results are compared. If the number of versions, n, is greater than 2, the computers can vote. Multiple-version software and multiple, isolated, independent testers are used in critical situations to improve reliability, despite the extra expense. One of the five computers on each of the space shuttles runs a second version of the software. (In fact, a delay in the first shuttle launch was caused by a veto from this computer. The veto resulted from a synchronization bug that had 1 chance in 60 of occurring.)

---

[1] Glenford J. Myers, *The Art of Software Testing* (New York: Wiley, 1979), 1.

[2] J. C. Knight and P. E. Amman, "Issues Influencing the Use of *N*-Version Programming," (unpublished paper, Software Productivity Consortium, Reston, Va., 1988).

Without question, this technique improves reliability, but much less than one would expect from reliability theory for systems other than software. In hardware, if the probability of failure of one component is p, the probability that a system of n independent (parallel) components will fail is p^n. If p is small, even $n = 2$ will result in very high reliability. However, the redundant components in software are human programmers or testers, and they are not independent. Human errors are correlated by culture (humans read the same textbooks and learn the same algorithms) and by genetics (the thought processes in different brains may differ, but not all that much). However, even n-version techniques are not a panacea that will eliminate software errors, they can be useful in controlling hostile infiltration of software development groups (an issue to be discussed in terms of the security of networked computer systems). If there are n, dependent, geographically isolated teams developing the same software, the probability of infiltrating all of them successfully does go as p^n. Successful infiltration would be essentially impossible.

Measuring software reliability is also a serious problem. Normally, to get a measure of MTTF, one averages the failure rate in a large sample. But the concept of multiple "samples" of software has no meaning. IBM has developed techniques that get around this problem for certain software products. Their answer is to measure a large sample of different usage. IBM measures the rate at which errors are found during an extensive debugging process involving release of the software to an ever-increasing circle of users. This technique is very effective for the errors of concern to IBM, which is "interested in failure-free execution intervals, rather than ... [an] estimate [of] errors remaining" IBM finds that 33 percent of errors are found after 5,000 user-years of being exercised (which IBM defines as 5,000-year MTTF) and therefore can be caught in a short time. The goal of this type of measurement is to determine a product-release date. [1]

What is special about the IBM experience is that it deals with compilers, linkers, operating systems, and other products that have a huge and diverse circulation. Users exercise all aspects of the software and quickly find the problems that most other users will find; these are the problems that matter to IBM's overall reputation. In other situations, the concern extends to the sum total of all the uncommon errors, those that could, for example, result in the crash of an airplane controlled by the software. In fact, so-called fly-by-wire software can at best be measured with present techniques to have a MTTF of 10^{3-4} years. This compares to the normal standard for commercial aircraft components in the neighborhood of 10^6 years.

---

[1] Harlan Mills *et al.*, *IEEE Transactions on Software Engineering*, SE-12, (1986), 3.

The problem for military computers[1] is, in this sense, similar to that for civil aviation and unlike that faced by IBM in its bug-rate measurements. David Parnas has identified a number of factors that make military computing unique. The five most critical factors can be paraphrased:

(1) Realistic testing of weapon systems is dangerous and expensive.

(2) Enemies don't think the way software designers think they do.

(3) Antagonists will exploit otherwise rare flaws.

(4) Battle induces common mode hardware failures, difficult for software to handle.

(5) The input error rate is high because of human and equipment stress. [2]

Projects for the development of military computer systems are often extremely large and complex and take more than a decade from conception to implementation in the field. There is a serious shortage of trained software – and systems – engineering project managers, a shortage not unique to the defense sector.[3] Few officials at the senior levels of the military have any in-depth knowledge of computers. Promotions are biased toward those in traditional field-commander roles. Those with computer expertise may opt for the silver lining and accept well-paid opportunities on the outside. The resulting shortage is compounded by the frequent rotation of management personnel, particularly in the military where the "three colonel" scenario is stereotypical. One colonel (or general) is the founder; the next keeps the project moving and gets promoted; the third takes the blame.

Those working in the area of military computers and software generally agree that the biggest problem is the definition of requirements. A Defense Science Board (DSB) task force stated in 1978, "The major difficulties are not primarily technical, but conceptual (What should the system do?)."[4] The real requirements are only determined in actual use. Certainly the best, if not the only, way to develop correct requirements is to evolve them with repeated use. The implications of repeated real use of software for testing and evolution of requirements are not acceptable in the context of military combat. The military needs more "realistic tests and exercises," particularly for tactical forces, that include "processes and procedures for crisis management and transition to war."[5] Exercises, simulations and careful design are essential, but they are far from complete in anticipating what will really happen under the stress of a crisis or battle. The 1987 DSB Command and Control Task Force emphasized, "A viable system for use in

---

[1] For an excellent discussion of software and computers in the military context, see the report prepared by the Office of Technology Assessment, *SDI Technology, Survivability, and Software* (Princeton: Princeton University Press, 1988), particularly Chapter 8, "Software" and Appendix A, "Technology for Producing Dependable Software"; see also C. A. Zraket, "Uncertainties in Building a Strategic Defense," *Science* 235 (1987) 1600-06, and Alan Borning, "Computer System Reliability and Nuclear War ," in Bellin and Chapman, *Computers in Battle*,101-48.

[2] David Parnas, "Computers in Weapons," in Bellin and Chapman, *Computers in Battle*, 209.

[3] David Shore, "Danger: Software Ahead," *Signal*, (May 1989), 27-29.

[4] *Report of DSB Task Force on Command and Control* (1978),7.

[5] *Report of DSB Task Force on Command and Control* (1987),22.

time of war or crisis must therefore embody robustness and the potential for graceful degradation."[1]

The "software problem" perhaps may be understood best in terms of software's not being amenable to the usual kinds of quantitative-design disciplines practiced by engineers dealing with other technologies. The problem persists in spite of being recognized and attacked. Since as early as 1960, software engineering[2] has addressed the issues of reliability in human programming of computers. In 1972 Frederick Brooks wrote the classic book on this subject based on his experience managing IBM's OS/360 development.[3] It is amazing how little has been added by others in the years since.

Traditionally, efforts have focused on means of organizing programs and making them understandable and readable. During the 1960s IBM provided free-flow charting templates to programmers. It is still considered undisciplined to fail to use such tools (or modern equivalents) to prepare a diagram of the logic flow of a program prior to starting on it. The flow chart should show what happens to data, from when it is input to when it is written out, including conditional actions and branches in the program that depend on the immediate state of the data.

In the 1970s a number of different methodologies evolved and were promulgated. These formed frameworks intended to define paths toward analyzing requirements and then writing programs that are well structured and, thereby, more readable and understandable, less error-prone, and easier to test and debug. Now often generically lumped under the terminology *structured analysis, structured design* (SASD), these methodologies are much used. In the 1980s they have been incorporated into computer-workstation software as computer-aided software engineering (CASE) tools. CASE systems are now marketed by over a hundred, mostly start-up, companies. The crusade for more structure in software has left its impact on programming languages with "structured" constructs like `do while` and `if then`. Programmers have been steered strongly away from from using conditional and absolute `go to ...` branches, which, according to the dogma, represent the evil that results in much unstructured, disorganized code. SASD and CASE tools have been valuable and have improved productivity, especially for complex programs, but they have not proven to be a panacea as some had hoped. The fact is not surprising if viewed in the historical context. One experienced software engineer comments that SASD and CASE are "just glorified flow charting, no different from the 60s."[4]

---

[1] Ibid., 12.

[2] At this first mention of "software engineering", it is important to remind that "engineering" here does not carry with it the usual implication that we are referring to a quantitative discipline, like, say, "mechanical engineering". We do not know how much stress will cause a software structure to collapse. In fact, as I try to show in the rest of this section, we do not even know what such a question means in any quantitative sense.

[3] Frederick Brooks, Jr., *The Mythical Man-Month: Essays on Software Engineering* (Reading, Mass.: Addison Wesley,1972).

[4] Private communication (1989).

Another traditional feature of research focus for software engineers has been in the area of large project management. Projects are divided up into phases of requirement, specification, design, coding, and testing, very much as in other fields of technology. [1] Extensive measurements support the common belief that the cost of errors increases by something like an order of magnitude for each project phase that passes before they are corrected. Extensive project-management methodologies are in place at IBM, AT&T, and other major software and computer-system developers. These insure that there is a careful review and documentation at each phase in an attempt to catch problems early. Design and specification changes are carefully tracked and controlled because of their possible impact on other parts of a system. When the emphasis is on the quality of reviews, rather than on paperwork and bureaucracy, such project management is very effective, perhaps the most effective means to ensure production of reliable software that does what is wanted.

The DoD has recognized the importance of such project management. It promulgates software standards like DoD 2167A which defines a rigorous and complex sequence of reviews and documents for DoD software contractors.[1] DoD 2167A requirements are so pervasive in the area of defense software that companies with CASE tools for the defense sector advertise "automatic 2167A document generation." Contractors staff offices just to produce the requisite 2167A paperwork. A defense contractor executive with extensive project experience put it this way: "Contractors set up two teams, designers and book writers, to meet the requirements [of 2167A]. It is coincidental if the results of the two teams are the same." [2] Military officers do not pay enough attention to the very difficult requirements phase, and give-and-take between those defining requirements and those specifying and designing a system is rarely adequate.

Although 2167A allows changes, it is normally used in one pass from requirements through design. The process is very difficult and results in requirements that bear little resemblance to reality. Requirements are typically 5 to 10 pages long per thousand lines of code, though there are many cases well outside this range in both directions. They are generally neither read nor reviewed properly, certainly not when the code "is over a million lines in scope". The project can't be stopped if, at any stage of the review sequence, it proves unsatisfactory. The executive continued: "Economic forces say proceed. So pretty soon you are reviewing something different. The books are irrelevant. You eventually have to go through a very painful merge because of the final tests." The result is a bottom up, rather than an intended top down design. The 2167A "process fosters problems by being too detailed." The result is that irrelevant issues get in the

---

[1] One should note, as David Weiss comments, the significant absence of a production phase in software technology.

[1] Department of Defense, *Military Standard: Defense System Software Development*, DOD-STD-2167A (Washington, D.C., 1988).

[2] Private communication 1989.

way of understanding what the project really needs to be able to accomplish. Another person, familiar with DoD software development, complained about the "excessive document load" that results from taking 2167A "as gospel" and requiring every document item.[1] The irony is recognized by many of having CASE tools automatically generate huge quantities of documents which are supposed to be read by humans without the benefit of automated aids. The other side of the coin is that prior to the present kind of standards DoD often had to suffer "the maintenance [costs] of undocumented, poorly developed systems."[1]

CASE tools and the way in which DoD 2167A or other rigorous review methodologies are conventionally applied represent the traditional approach to software engineering. More than this is clearly needed. Recognition has grown widely in the last decade of two concepts that are key to further improvement in software reliability: *evolution of requirements* and *software reuse.* A realistic and complete understanding of requirements is central to the goal of software that is reliable, effective, timely, and economically affordable. Big, complex projects are just too difficult to define completely from the beginning. In the military situation, successful evolution of requirements can be accomplished by keeping documents of requirements short at first with general descriptions of what is desired. As a design proceeds, the client and designers must work together on evolving and customizing requirements. Such a process requires a commitment of time and competent personnel to a continuing requirements process. Nothing is more important for obtaining military software that has a chance of being effective in a crisis.

The difficulty is focusing the attention of clients, military or otherwise, on the requirements process. In the military this problem is exacerbated by frequent rotation of personnel. Those with the initiative to start a project are rarely around to see it through to completion and thus to influence a consistent evolution of requirements. Nonetheless, clients are rightly most concerned about their interaction with the computer system. This point of interaction is the human interface, where users input requests and receive output as display screens or paper reports. The technique of *rapid prototyping* is proving very effective in allowing quick turnaround of ideas in the development of human interfaces. Special or general-purpose computers are set up so that displays are easily changed. No consideration about performance or computer cycles is expended during prototyping.

The process of rapid prototyping involves clients deeply in the decision-making process for requirements. A quick creative process to determine what a system will be like, using (often glitzy) color displays, seems to attract and hold clients' attention at this critical phase much better than the traditional exchange of arcane documents. This fun approach to requirements focuses on displays and screens, exactly where one should focus to get started on a proper

---

[1] Private communication 1989.

[1] David Weiss, private communication, August, 1989.

evolutionary top-down design. One can only hope that rapid prototyping will be strongly encouraged as part of the DoD procurement process. Deeper user involvement and understanding that lead to more accurate requirements, and evolved designs that meet real requirements, should overcome the puritanical sense that if a process doesn't hurt, there must be something wrong with it. Unfortunately, the benefits of rapid prototyping and extensive end user involvement, are less applicable in some areas, like aspects of automated weapons systems, where there is little human interaction.

The deep involvement of users in the requirements and design process is so important to the ultimate success of a system that creative management methods are being applied to encourage such involvement. The problem comes in breaking away key client personnel from their ongoing responsibilities, the very responsibilities that the new computer system is supposed to assist. In a very successful and interesting experiment, DuPont isolated teams of users and software developers for periods of as long as six months. Each team was given a basic project definition and a "time box" in which it was to complete the new software project. The team was rewarded individually and together and celebrated in the house newsletter, if it succeeded within the allotted time. All teams succeeded, and the savings in cost to the corporation, compared to estimates based on experience with traditional approaches, was extraordinary. On average, the seven initial projects came in at 33 percent of the estimates. The key to this strategy was the use of rapid prototyping techniques, combined with full-time access by software developers to knowledgeable and competent users. The user-clients' understanding of their own requirements, and the realities of implementing them, deepened during the development process. In this way, not only were systems completed with great efficiency in the use of personnel, but the systems were also more likely to meet the real needs of the organization.[1]

Rapid prototyping can be made more efficient by *software storming*, a short and intensive software development effort intended to get a first order approximation of the system requirements. According to an individual familiar with this approach, "Practice has shown that the issues uncovered in this ["storm"] are the issues which require the most attention throughout the prototyping period."[2] Prototyping is, in some sense, testing in advance. Therefore, just like testing, prototyping is subject to missing out where scenarios are not tried. Here one never finds the last requirement, as in testing one never finds the last bug.[3] Involving end users in the process is not just a matter of asking them what they like. All too often they will tend towards preserving their existing work environment. Somehow in the rapid prototyping process end users must be lead beyond what is familiar to new approaches that will improve their capabilities. Typical of the many attempts to address the software problem, rapid prototyping must be used with the

---

1 Peter R. Mimno, at seminar, "CASE Tools, Comparison and Review, "CASExpo, Anaheim, Calif., 1988.

2 Andrea Weiss, private communication, August, 1989.

3 Frank Maginnis, private communication, August 3, 1989.

understanding that it will have a limited impact. Its application in practice must be managed with care. One must know how to stop the prototype phase cleanly, identifying the requirements that have been learned, and not succumb to the temptation of accepting undisciplined prototype code as the basis for the final program.

A rather vague concept that has received much attention in recent years is the reuse of mature (that is, relatively reliable) software. The core idea is obviously correct: to the extent you reuse reliable software, your software will be reliable.[1] The hype associated with software reuse mostly concerns the automatic cataloging of software and what it is meant to do. Automatic cataloging of software is an unrealistic promise that has been made by some in the artificial intelligence community. Clearly, if it were possible to define for a computer exactly what you wanted, and if there were a catalog of what existing software could do, all categorized well, then the problem would be solved. You would just click your mouse, and, voilà, the software you needed would be ready to reuse, in perfectly (and relevantly) documented form.

The idea behind reusing software is more subtle than the hype indicates. Two software-engineering approaches, one dating back to 1970, are important to the reusability goal. These are *information hiding*, a concept suggested by David Parnas in several early papers[2], and, what can reasonably be considered its descendant, *object-oriented programming*. Parnas's deceptively simple idea is to have small teams of coders work on software modules. These modules communicate only through extremely well defined interfaces. The modules contain deliberately hidden information, one or more secrets that define how the module operates. Other modules – and teams – know the interface definition but need not and should not know any but their own secrets. The secrets may, for example, refer to hardware-specific matters that are in this way isolated to single modules. In normal programs a change of hardware can have consequences throughout a huge software package. Parnas's secrets restrict consequences to single modules. Since consequences are kept local, information hiding allows easy evolution of requirements and change of software. It also is effective in allowing reuse of these very well defined and consequence-isolated modules. Information hiding encourages compartmentalized, well defined tasks, which are easier to catagorize for possible reuse. It does nothing to assist in the dream of automatic cataloguing. But at least, if a module is remembered, it is likely to be reusable.

---

[1] See, for example, *Proceedings, Workshop on Reusability in Programming*, (Newport, RI: ITT Programming,1983); and Will Tracz, ed., *Software Reuse: Emerging Technology* (Washington D. C.: IEEE Computer Society Press,1988).

[2] D.L. Parnas, "A Technique for Software Module Specification with Examples," *Communications of the Association for Computing Machinery (ACM)*15 (1972): 330-36, "On the Criteria To Be Used in Decomposing Systems into Modules," *Comm. of ACM* 15 (1972):1053-58, and "On the Design and Development of Program Families," *IEEE Transactions Software Engineering* SE-2 (1976):1-9.

Object-oriented programming has gained notice in recent years, much of it as a result of the development of the Smalltalk language by the Xerox Palo Alto Research Center (PARC).[1] Smalltalk was subsequently discovered by Steve Jobs on a legendary visit to Xerox PARC. It was then applied to the famous graphical human interface of Apple's Macintosh computer. Objects consist of data and one or more methods (like program procedures) that operates on the data. One communicates with objects by passing messages between them telling them what to do. The well-defined message-passing interfaces and the consequence-isolated methods of the objects are certainly within the spirit of the information hiding philosophy. A class of objects is defined once; it may have many objects that are instances of the class (such as the multiple windows on a Macintosh). Classes, with their own specialized traits, may inherit general traits from other classes (for example, the traits of a basic window may be inherited and augmented by scrolling capabilities). One reuses what is needed and never has to describe anything twice. This is true software reuse, building on what has been done before. A large object-oriented program is built up – and tested – object by object, in an evolutionary way that clearly reduces bugs. The ease of changing objects carries with it a disadvantage from a project management perspective, the difficulty of controlling change.

In addition to Smalltalk, the original object-oriented programming system (OOPS), there are now object-oriented versions of the programming languages LISP (used in AI) and C. One version of object-oriented C, called C++, was developed by Bjarne Stroustrup at AT&T's Bell Labs, the originator of C, and is showing encouraging signs of standardization and widespread acceptance.[2] Objective C, developed by Brad J. Cox, was intended to demonstrate a technology that would allow, in principle, adding objects to other important languages, most importantly FORTRAN.[3]

The DoD has put a tremendous investment into an effort to produce a new language called Ada. The DoD's regulations now require its contractors to use this huge and complex language, with a decreasing rate of exemptions. Ada has many modern software-engineering ideas in it, and apart from its complexity and size it represents, though many dispute this, a step forward over existing heavily used languages. Ada is commonly considered to be object-oriented. Object-oriented programming has many aspects, and Ada misses out on some that are central, including straightforward inheritance.

Ada is a strongly typed language; thus programmers must define the type (such as integer, floating point, text string) of a variable before using it. This philosophy is meant to reduce errors

---

[1] A good introduction is Mark Stefik and Daniel Bobrow, "Object-Oriented Programming: Themes and Variations," *AI Magazine*, 6, 4 (winter 1986): 40-62.

[2] Bjarne Stroustrup, *The C++ Programming Language* (Reading, Mass.: Addison Wesley, 1987).

[3] Brad J. Cox, *Object-Oriented Programming, An Evolutionary Approach* (Reading, Mass.: Addison Wesley, 1987).

since the compiler will catch misuse of a variable. Unfortunately, rigorous insistence on strong typing conflicts with allowing type changes at run time. *Dynamic typing*, also called *loose binding*, permits objects to do similar things to different types of data in ways that appear similar to the programmer or user. The information as to type is hidden with the data. Binding of type to data is usually done at compilation. Dynamic typing is important in deferring binding until a program is run. This has the effect of making more convenient a broader application of the inheritance mechanism. For many problems, dynamic typing greatly expands reusability and reduces errors. The debate over the advantages of strong and loose typing is ongoing. Clearly, despite DoD's goal, Ada cannot be the single language for everything.

I have intended, in the preceding extended discussion, to give a flavor of what is unique about "the software problem" and why intuition and experience gained from other technology areas can be very misleading. Most important is to understand the extreme difficulty of making big advances in the productivity of writing software or the reliability of the result. There are no simple solutions. One needs to exercise considerable judgment in walking the knife edge between anarchy and the potential for sloppy unreliability, on one side, and excessively bureaucratic philosophy constraining creativity and changeability on the other. The reliability of military software relates directly to the sensible use of techniques and methodologies of modern software engineering.[1]

The emphasis here has been on describing how the problem is being attacked. This is one good way of viewing this rather intangible subject and sensing how difficult the struggle to manage software development really is. Looked at from another direction, when computer systems are successfully applied, as they so often are, this serves to whet the appetite for applying them to more complex problems. A prime cause of the software problem is that our ability to imagine ever more ambitious systems is not limited. What is limited is our ability to understand and express, and then to produce and test, what we imagine, precisely enough to make our dreams work properly in reality.

Because of the difficulties involved, producing reliable software in quantity makes great demands on the available pool of talent. The limited size of this talent pool must be addressed, even before the question of budgets and deficits, in determining whether the United States has the resources for any proposed computer-related project. For example, software talent that is tied up in a particular defense effort is not available for developing civilian systems to help maintain a competitive posture in international trade. Most seriously, attempting more projects than the talent pool can support is a prime cause of ineffective and failure prone systems in both the military and civilian sectors.

---

[1] Gene Layman and J. R. Robins, "Open Architecture for Modern Command and Control Systems," *Signal*, (August 1988): 83-89.

Artificial Intelligence and Expert Systems

Artificial intelligence has attracted great interest and much publicity, often driven as much by wishfulness as accomplishment. The essential goal of AI is to apply the speed of computers to intelligent problem solving. Attempts to reach this goal have highlighted the essential differences between computers and the human brain. All computers follow the traditional von Neumann concept of a stored program based on a sequence of individual instructions that (a) fetch numbers, (b) perform arithmetic (+, -, x, +) or logical (and, or) operations on those numbers, (c) store results, and (d) determine succeeding instructions by sequence, branch, or conditional branch. Though some computers may be optimized for inference activities, they are all essentially *numerical* calculating engines. Intelligent problem solving is not primarily numerical but tends to involve pattern recognition and matching.

AI perhaps may be described best as the effort to provide automated assistance to humans addressing nonnumerical problems, including such challenges as language translation, chess playing, speech recognition, vision, reasoning and deduction, theorem proving, and learning. A traditional working definition of AI, "the study of mental faculties through the use of computational methods," traces back to John McCarthy's original proposal for a summer study in 1956 to be based "on the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it."[1] The history of the last three decades indicates that the goal of fully autonomous AI has succeeded only to the extent that McCarthy's original conjecture is true. When problems, and the means of solving them, can't be precisely defined or the requisite conventional computing power applied, autonomous AI has run into presently insurmountable obstacles.

Because truly autonomous AI is so difficult to achieve, AI research has increasingly been directed toward providing tools to assist humans. The most common successful application of AI ideas has been in expert systems, which may be queried by humans to dispense expert information. So-called *mixed-initiative* decision aids are particularly important in the military context. These allow humans to interact, or not, with the computer at any point in the decision process. The intention is to allow each member of a human-computer team to handle what it is best at: the human, the global issues; the computer, the detailed doctrine. (Such aids are discussed later in this section). Also important has been a spin-off from the AI community – good *human interfaces*, techniques that effectively couple the nonnumerical human brain to the computer. (The importance of improving human interfaces, and using good ones, is the subject of the next section.)

---

[1] Eugene Charniak and Drew McDermott, *Introduction to Artificial Intelligence* (Reading, Mass.: Addison Wesley, 1987), 6, 11.

AI has proved successful in allowing autonomous computers to accomplish, with available capabilities, tasks that have heuristic (rule-of-thumb) descriptions. The heuristic description has little, if anything, to do with the way a human brain does a task. It is a description adequate to instruct a brute-force, sequential, computing engine. This technique has worked well for chess: only top players, with considerable effort, can generally beat the best chess-playing computers. The computers search out and score large trees of possible options as they select best moves. Grand masters, in contrast to computers, tend to look at patterns on the board, before checking small search trees. (The oriental game of Go has a significantly larger combinatorial complexity than chess, and its play therefore, moves beyond the present threshold of what a brute-force computer can successfully challenge.) Using heuristics and computational speed, AI has been able to achieve very simple speech and pattern recognition, syntax analysis, and simple or repetitive aspects of theorem proving after humans have organized the proof (the four-color theorem was proved by computer).

AI has encountered significant difficulties with problems that require a "common sense data base". Examples of areas where such a data base is critical are language translation and its prerequisite, natural-language understanding. Human beings carry around with them a lifetime of experience we call common sense. (We know, for example, that telephones are generally loose on tables or desks in buildings, but are sometimes attached to walls or outside pay booths, or to the dashboards of cars, in which they work only in or near cities and cost more per minute, and so on.) The resulting knowledge of context and varying relation and form is what allows humans to "understand" language, as well as to parse it. The latter is closer to what computers really do than "understanding".

Without a common-sense data base, it is impossible for a computer to act intelligently beyond the very restricted domain for which it has information, expert and otherwise. Efforts are in progress to produce a common-sense data base that would be available to systems trying to "understand" natural language and for limited-domain expert systems to access when inferences led outside their domains. It is not clear at what threshold of completeness a common sense data base would start to become useful nor what scale of resources would be required to get there. Nonetheless, the effort in this area, led by Doug Lenat, is extremely important to all problems requiring artificial understanding.[1]

The goals of autonomous AI have also proven unattainable in areas where brute-force computational approaches require more power than the hardware can provide. Most difficult are problems involving recognition of sophisticated patterns. These include complex voice and scene patterns as well as lengthy logic and reasoning. As conventional computer hardware becomes more

---

[1] Doug Lenat, Mayank Prakash, Mary Shepherd, "CYC: Using Common Sense Knowledge to Overcome Brittleness and Knowledge Acquisition Bottlenecks," *AI Magazine*, , 6 (winter 1986): 65-85.

and more powerful, progress results automatically in these areas. However, full scale success, particularly in tasks involving open-ended logic and reasoning, is orders of magnitude away.

The basic reason for all this difficulty lies in the attempt to use computation to imitate what the brain does. The brain does not compute. In order to compute, one must first analyze, as McCarthy suggested in 1956. But try to analyze how you discriminate between two human faces, a cat face and a human face, or two leaves on a forest floor. To the extent you can analyze a problem, you can compute it. If the analysis is complex, the number of computational steps may become enormous and out of reach.

Before returning to the possibility of doing things more the way brains do, let's look at the important accomplishments in providing expert-system and mixed-initiative tools to assist rather than replace humans. Expert systems are rule-based programs for tasks requiring expertise. They tend not to use deductive logic, but rather abductive reasoning. The word *abduction* describes the unnatural process of generating explanations, cause from effect. Normal logic proceeds by deducing effect from cause. Abduction involves such heuristic techniques as plausible inference and the weighing of evidence. Most important, for lack of a broad understanding of the world shaped by common sense, expert systems must be limited to narrow and isolated domains.

Expert systems generally are run on *inference engines*, specialized programs on von Neumann computers optimized to do the logical inference operations (`if ... then...`) that expert systems use most heavily. Famous examples of the effective use of expert systems are diagnosis of breakdowns of railroad locomotives, and, in the case of Digital Equipment Corporation's VAX series of computers, determination of appropriate customer configurations. Expert systems appear to present a significant debugging problem when they get large. This implies that an inherent size limit may come into play in the future.[1]

The process of obtaining knowledge from a human expert (on locomotives, for example) and putting it into a form that an expert system can refer to goes by the loaded name, *knowledge engineering*. Not surprisingly getting experts to agree, when there is more than one, is a significant problem. It is an interesting, perhaps semantic, question whether expert systems dispense knowledge or doctrine. One can imagine that when the use of medical diagnostic systems becomes widespread, any medical doctor who ignores the questions and conclusions of the systems will risk malpractice suits.

In the military context, as one might expect, much of what is going into expert systems is, in fact, doctrine. For example, MOVES (Maneuver Planning Expert System) is a major subsystem of the AirLand Battle Management (ALBM) program, which supports planning at the corps and

---

[1] David Weiss, private communication, August, 1989.

division levels. The system's planning procedures are modeled "as a starting point after the doctrinal planning process described in various Army field manuals and taught at the Command and General Staff College"[1] Such tools are potentially very valuable because they could relieve planners, heavily stressed by combat, of the *cognitive load* of remembering all the details of planning procedures. The machine could lead the error-prone, but sometimes creative, human through details and doctrine up to the level where the planner could concentrate on looking at the larger picture and selecting flexibly between alternatives.

One might wish for decision-making aids in combat to broaden the availability of commanding genius. But, imagine, for a moment, subjecting someone like George Patton to knowledge engineering. Such genius is usually neither stimulated nor identified until a battle is joined. Present capabilities do not permit rapid knowledge engineering, certainly not on the time scale required for modern warfare.[2] They do allow for input of a commander's guidance as do standard, non-computerized, U.S. military-planning procedures.[3] Such input is a familiar means for creativity, if not genius, to enter the planning process at the last minute. However, expert systems, if they live up to their promise, will be able to incorporate advanced consideration of many more contingencies than a printed field manual can possibly cover. Clearly identified competent and knowledgeable military leaders must invest time in the slow process of "teaching" expert systems, as much as possible, about situations that go beyond what is covered by traditional doctrine.

Since computers are now producing a tremendous flow of data both for behind-the-lines logistics and for battlefield activities, it is not only reasonable, but essential, to ask their assistance in managing and interpreting the data.[4] This includes assistance in routing and interpretation of intelligence and warning (I&W) data, fusing information coming from dispersed sensors, sources, and computer systems, and aiding commanders in evaluation of threats and options. Assistance in interpretation and evaluation represents a clear-cut application of mixed-initiative expert systems as decision making aids This extends with increasing sophistication a long standing military application of computers that predates expert systems. Though essential, such applications are potentially dangerous, the more so as the time scale for action gets shorter and the volume of data becomes more overwhelming.

---

[1] Gregory Stachnick and Jeffrey Abram, *Army Maneuver Planning: A Procedural Reasoning Approach* (unpublished paper, Advanced Decision Systems, Mountain View, Calif., 1988). This paper refers to Department of the Army manuals, FM 101-5, *Staff Organization and Operations* (1984); FM 100-5, *Operations* (1986); FM 100-15, *Corps Operations* (1987).

[2] Zvi Lanir, Baruch Fischhoff, and Stephen Johnson, "Military Risk-Taking: C3I and the Cognitive Functions of Boldness in War," *Journal Strategic Studies* 11 (1988), 96-114. This paper discusses the problem in combat situations of allowing "military boldness" supported by "calculated risk" computerized decision aids.

[3] Stachnick and Abram, *Army Maneuver Planning*, 3.

[4] Numerous articles on C^3I in *Signal* magazine over the past few years have described this activity.

When information is dispensed clearly (see the discussion on human interfaces) and when humans are allowed full control of the decision process in mixed-initiative systems, decision aids can help enormously to reduce battlefield confusion. The result is to reduce the possibility of mistakes that could lead to crisis escalation. Conversely, such systems, by providing erroneous or misleading information, have the potential of destabilizing a situation. One cannot overemphasize the importance of getting the right information into these battlefield expert systems and making sure it is disseminated so that no mistake in understanding is likely. These tasks require a much heavier investment of a commander's time in "downloading" knowledge, policy, and doctrine into a system and, most critically, being sure they are presented in an unmistakable way. How decision aids are used and understood in the field will determine how the forces will act. It would certainly appear that commanders at very high levels should find it of immediate interest to check in on the rapid-prototyping sessions that define the tools the planning teams in units under their command will use. These vital matters deserve much more attention than they are getting.

Some more radical research may also prove practical. An example is research into the concept of distributed decision aids spread out over a modern networked computer system. It should be possible to distribute information in such a way that, even in the ultimate fog of war, when communication links are down, the expert system will be able to suggest, on the basis of available information, what neighboring allied (and even enemy) commanders are likely to do.[1] At the simplest level, knowing the procedural tools, and at least some of the data, available to other commanders will allow a simulation of their planning processes and forecasts of their likely actions.

It is less reasonable to promise that systems will be able to do fully automatic, natural-language threat analysis of I&W traffic (though some effective sorting of traffic is now possible)[2]. Nor can the systems be expected to scan automatically raw voice, intercepted traffic, or imagery.[3] What is possible are systems that will aid I&W analysts and improve their efficiency. Such systems will be important in dealing with the large increase in data, particularly imagery, from new satellites. Digitized imagery displayed on high-resolution workstations, combined with expert systems and well-configured data bases, will provide an environment that should increase analyst efficiency by a factor of two or so.[4] Such a gain in

---

[1] Bruce Hamill and Robert Stewart, "Modeling the Acquisition and Representation of Knowledge for Distributed Tactical Decision Making," *Johns Hopkins APL Technical Digest* 7 (1986), 31-38. See also M.Athans, *Command-and-Control (C2) Theory: A Challenge to Control Science*, (Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, Mass.) .

[2] Bruce Loatman and Stephen Post, "A Natural Language Processing System for Intelligence Message Analysis," *Signal*, (Sept. 1988): 41-45.

[3] Michael McCown, " Knowledge-Based Intelligence Systems," *Signal*, (April 1988): 101-103.

[4] Eamon Barrett, briefing, Lockheed Corporation, Mountain View, CA, July 1989.

productivity could prove critical, for example, to arms control progress. Verification depends more and more on an increasing flood of images and other data supplied by "national technical means". Improvements of more than an order of magnitude in the efficiency of handling I&W data are simply not on the immediate horizon; the barriers are very similar to those that affect software reliability and productivity. Wishing for more won't get it.

The expectations from military AI research were largest and the wish list was most unrealistic in 1983, when the Defense Advanced Research Projects Agency (DARPA) initiated the Strategic Computing Program (SCP) with the mandate to "develop a new generation of computers that can SEE, HEAR, TALK, PLAN, and REASON [*sic*]."[1] The Strategic Computing Program was named and announced within months of naming the Strategic Defense Initiative. SCP represented a conscious decision to "prove AI works." Related was the Reagan administration's effort to push DARPA toward supporting applied research and away from its decades-long role as the dominant, extraordinarily successful, mentor of basic research in computer science.[2] The key demonstration projects were to be the Autonomous Land Vehicle, the Pilot's Associate, the Naval Battle Management System, the AirLand Battle Management System, and the Image-Understanding Program. The subgoals stated by fiscal year, were, to put it mildly, unreasonably optimistic. For example, subgoals for fiscal years 1992 and 1993 included: "Recognition of [spoken] Sentences. Independent of Speakers, from a 10,000-Word Vocabulary with Natural Grammar under Moderate Noise and Low Stress Conditions"; "Perform [visual] Reconnaissance in a Dynamically Changing Environment"; and "Interactive, multi-user acquisition, analysis, and explanation system which provides planning support and substantive understanding of streams of textual information."[3]

Historically, AI has been oversold in the military context because its promises resonate with the prevailing perception of military needs by decision makers.[4] As recognition of overselling has grown, expectations have become more realistic,[5] at least among practitioners if not among all high-level military and political decision makers. At DARPA the pendulum has swung, perhaps overswung, in the other direction. Shadows confused the autonomous land vehicle (ALV)

---

[1] Quoted by S. Ingvar Åkersten in Allan M. Din, ed., *Arms and Artificial Intelligence*, (New York: Oxford University Press, 1987), 87.

[2] See, for example, Andrew Pollack, "America's Answer to Japan's MITI", *New York Times*, March 5, 1989; and Jonathan Jacky, "The Strategic Computing Program," in Bellin and Chapman, *Computers in Battle*, 171-208.

[3] Quoted by Åkersten, in Din, *Arms and Artificial Intelligence*, 94.

[4] It is not just in the defense sector that AI has been pushed with overambitious goals. The Japanese, normally thought to be more judicious in such matters, fell victim to the same syndrome eight years ago when they announced their "fifth generation," predominantly AI, computer-research project. This project was cited as a threat to U. S. leadership in the environment in which DARPA's SCP was born. By now the goals -and accomplishments - of this project have been considerably scaled back. See, for example, Howard Ullman, "Machine Dreams," *The New Republic*, July 17 and 24, 1989, 12.

[5] See, for example, Edward Taylor and Daniel Snell, "Artificial Intelligence in Command and Control," *Signal*, (April 1988): 25-29.

even on paved roads, and it never was able to move rapidly and decisively cross-country as planned. This project has been canceled quietly. Some argue strongly that the DARPA computing leadership is now decidedly AI-unfriendly, even toward research and development (R&D) into realistic mixed-initiative tools.

Yet, the Strategic Computing projects did push the technology forward. Even if it takes brute-force computing, one should not discount the potential importance of even small advances in AI. The mentality for publicity, intrinsic to SCP projects, which were born with loud public promises about applicability, exacerbated technical problems in several of the projects. The promises were glitzy and required demonstration. The ability to show off ALV feats in dramatic fashion was almost an explicit project requirement, and a serious misdirection of resources and emphasis resulted. One should never oversell the potential or the successes of AI, or any other advanced R&D work for that matter. One shouldn't overreact to failures either, certainly not without a good understanding of what caused them.

Today the long-term hope for achieving true (autonomous) artificial intelligence would appear to have switched to connectionist approaches to simulating the capabilities of the brain. This new and extremely active field of endeavor is usually referred to generically as *neural networks*, though practitioners have a variety of specialized meanings for the term. The activity represents the first true breakout from the von Neumann computer concept. Macroscopically, neural networks mimic the widespread networked interconnection of neurons and synapses in the brain. Microscopically, as any neurobiologist will tell you (some sneeringly), the connections in the network of the brain's neurons are more numerous by many orders of magnitude, with multiple rather than binary levels, than those of any imaginable electronic neural network.[1]

Conceptually, electronic neural networks involve interconnected operational amplifiers (though many present-day implementations use digital logic). The output of each amp is fed through reinforcing and inhibiting (complemented) connections to many or all other amps. The weights of these connections may be adjusted. Input data is represented as levels feeding into some of the amplifier "neurons." After the network has had a chance to relax to an equilibrium state, the output of certain neuron amps can be interpreted as the "answer." If you get your weights and reinforcing/inhibiting connections right, the network will look at the whole picture, so to speak, and relax to the right answer. By feedback of correct answers (from a human looking at the same data), the weights can be automatically adjusted in a procedure that corresponds to a type of learning by the network. This ability to "learn" is one of the most exciting factors that encourage optimism about neural networks. Another important factor is that the discrete and repetitive nature of neural network circuitry is particularly appropriate for the technology of very large-

[1] Jacob T. Schwartz, "The New Connectionism: Developing Relationships Between Neuroscience and Artificial Intelligence," *Dædalus*, (winter 1988): 123-42.

scale integration (VLSI). In fact, several designs for neural network chips are in progress at this time. The effort to develop chips is proceeding in parallel with conventional computer simulations of neural network topologies.

Neural networks have clear potential for imagery recognition and interpretation and of recognition of patterns in other applications. Physically, neural networks created by VLSI should couple well to pixel sensors, for example, to allow direct interpretive readout. The excitement and publicity about neural networks appear to rest on a somewhat stronger conceptual foundation than do earlier flurries of promises about autonomous AI developments. This research, nonetheless, is in its infancy. Full-blown, revolutionary applications are far off, and not, by any means, guaranteed. However, processing capabilities that are more brain-like will remain key to AI's efforts to address big wishes. Despite the extreme simplifications made in copying what happens in real brains, neural networks may provide a sufficient breakthrough for real progress where pattern recognition is required. It remains to be seen whether the hopeful vision many see in connectionist approaches is an oasis or mirage.

Human Interfaces

If the information flowing from a computer does not arrive correctly at the appropriate human brain (or vice versa), the human-computer decision-making system has not benefited from the computer's participation. The two partners in the activity, so to speak, have different personalities. The human brain is visually oriented and pattern-sensitive – and numerically slow. The computer is numerically oriented and computationally powerful – and insensitive to patterns and whole scenes. The computer works on small parts in terms of numbers. The human sees whole patterns and functions in terms of graphics. For the human-computer partnership to function effectively, the two must communicate through an effective interface.

It is difficult to overstate the importance of human interfaces. Not only is the productivity of the human-computer system at stake but also, if wrong information is received, the result may be destructive. In the military context, wrong information can be seriously destabilizing. Computers can produce wrong information because of errors in a program or input data. (The famous "garbage in, garbage out" metaphor indicates wide appreciation of the problem). But even when computer information is correct, errors occur all too often at the human interface. Some of these errors have resulted in widely publicized disasters. The disasters are usually attributed to "computer error" or "human error" but in fact should be attributed to errors in the interface between computers and humans.

The reactor problem at Three Mile Island (TMI) quickly grew out of control because operators could not cope with information displayed on a large wall of panel meters and status indicators. Utility companies had used such displays in conventional plants, where the displays had been acceptably matched to the information levels, time scales, and consequences of potential failures. An inadequate human interface design was an important contributor to the ultimate scale of the Three Mile Island disaster.[1]

In January 1989 a 737 jetliner crashed on the M1 expressway in England. Even though fire indicator showed a serious problem with one engine, the plane should have had another working engine with sufficient thrust to land safely. Soon after the accident, it became clear that the pilot had shut down the wrong engine. The *Times* of London on January 19 headlined "Warning dials may hold secrets of M1 air crash." The display (figure 1) was a new style electronic readout, but its graphics had been patterned after those of electromechanical dial panels traditionally used in 737s.

---

[1] See, for example D. Canter and J. Powell, *Summary of Report to Insurance Technical Bureau* (Guilford, United Kingdom: University of Surrey, 1983) (quoted in James Thompson, *Psychological Aspects of Nuclear War: A Statement of the British Psychological Society* (New York: Wiley, 1985), 45.

An indicator for the good engine showed severe vibrations (correctly, since the engines were mechanically coupled), which perhaps caused the pilot to confuse the engines despite the fire indicators' pointing at the correct location of the problem. Whether or not this confusion, in fact, caused the crash, it is clear from the layout of the displays that in a difficult situation confusion could all too easily occur. The left/right "vib" engine dials are adjacent and away from the fire indicators (not shown). Indicators are kept in fixed locations, as had been required for mechanical meters. Problem indicators – and option selectors – are, thereby, not clustered in one place. Such clustering would nearly eliminate the type of error thought to have occurred. In part, inertia and tradition among pilots, in advising what they want in displays, led to the use of graphics that do not take full advantage of modern technology for electronic displays.

When the U.S. Navy played back the tapes from the *Vincennes* command center, it was clear that the Aegis control system had not failed on the cruiser. It had provided full and accurate information as requested by the command center crew. Nor had the the ship's captain acted wrongly on the information that had reached him. Yet Iran Air 655 was shot down, in error. "In the *Vincennes* incident," a psychologist testified to Congress, "the highly sophisticated equipment performed flawlessly while breakdowns occurred in the machine-to-commander link."[1] Something happened between the computer and the decision-making human that corrupted the information flow. Manifestly, the human interface failed.

It is inappropriate to attempt here an analysis of the second-by-second chronology that led to the accident. Much of that chronology, and the Navy's analysis of the incident are available in the open literature.[2] Somehow, the attribute "UNKNOWN – ASSUMED ENEMY" was attached to a commercial airliner as it was taking off from a mixed-use airfield and was never corrected in the critical and tense minutes that followed. Somehow the Aegis operators, information to the contrary on their screens, persisted in the misattribution. The ship was already engaged in a surface battle with Iranian gunboats, bullets bouncing off the steel. A gun failed, and the Vincennes heeled over in a full-speed, full-rudder turn to direct a working gun at a real enemy. Books fell off the command center shelves; lights dimmed. The command center operators – and some indicators that caught their attention – reinforced the fatal misattribution in a psychological phenomenon known as *scenario fulfillment.*

The information was on screens, on corners of screens – well, it was at least available to be called up if anyone asked. Apparently, nowhere was *all* the fatal information associated with the firing decision popped up to one unmistakable display, isolated from extraneous information,

---

[1] Robert Heimreich, "On the Subject of the USS *Vincennes* Downing of Iran Air Flight 655," testimony before the U. S. House of Representatives Committee on Armed Services, October 6, 1988.

[2] William Fogarty et al., *Formal Investigation into the Circumstances Surrounding the Downing of Iran Air Flight 655 on 3 July 1988: Investigation Report*, (Department of Defense, Washington, D. C. 1988).

for evaluation by an officer. Just as in the case of the 737, before the M1 crash, the information had to be fused in human brains from many isolated bits of display information. The technology allows better these days.[1]

Human failings were certainly a key factor on the Vincennes. Related to scenario fulfillment, another human failing, which has been studied in the laboratory, is thought to have played a role. This failing goes by the name *perceptual hysteresis* . When confronted with the same set of indicators for evaluation of a threat, analysts evaluate the threat differently depending on how those indicators turn on (see figure 2). If a small number of indicators turn on at first at high confidence, followed by more at high confidence, assessment of threat is lower than if many indicators come on together at low confidence and then increase together in confidence. The result is the folded threat response surface of Fig. 2, with a region of high threat perception from which it is difficult for people to retreat. Such perceptual trapping is suspected in the *Vincennes* scenario. [2]

Psychologists have cataloged many other types of human error in the subjective assessment of probability (see Table 1). The important lesson to be learned from the *Vincennes* incident, as well as the M1 crash and TMI disaster, is not that there are human failings. It is that the failings are amplified by stress and that good training alone will not eliminate them. To blame accidents on human errors is simplistic and unproductive. In an excellent review of the psychology and technology of distributed decision making, Fischhoff and Johnson note the common "misunderstanding of the role of people in person-machine systems, somehow reflecting a belief in the possibility of engineering the human side of the operations just as one might hope to engineer the mechanical or electronics side." They add, "A common sign of insensitivity is use of the term 'operator error' to describe problems arising from the interaction of operators and system."[3] The focus must be on making decision aids, and their human interfaces, so powerful that they have a chance at driving critical information through human perceptual barriers.

A long-standing, apparently inherent resistance to paying adequate attention to good human interfaces is not unique to the military. This resistance is one aspect of a universal lack of emphasis on clarity in communication, a lack dramatically apparent in any literature from the defense sector. Communication and friendly computers seem not to be a macho priority. Like technical writing, designing good human interfaces makes demands on an uncommon mix of

---

[1] One shouldn't be too critical of the Aegis interface. Though Aegis represents nearly the latest in naval systems, it is based on early 1970s technology, and its human interface was probably ahead of its time. It takes years to bring a weapon system through the development and procurement process.

[2] Alexander Woodcock, "Perceptual Hysteresis: You Cannot Get There From Here," *Signal*, (July, 1988): 51-57 (an article that anticipates the *Vincennes*); and "Deadly Threat: A Nonlinear Analysis," *Signal*, (March, 1989): 67-72.

[3] Baruch Fischhoff and Stephen Johnson, "The Possibility of Distributed Decision Making," *in Political-Military Decision Making* (forthcoming).

aesthetic, humanist, and technical sensibilities. Defining requirements for a human interface, as well as permitting extensive rapid prototyping, runs counter to the usually rigid nature of the competitive procurement process.

Other sources of resistance contribute. Experts exhibit a bit of technical territorial imperative and take some comfort in difficult interfaces at the boundary of their domains of expertise. There are probably also those with make-work attitudes who don't want things too easy for the troops. Some issues are legitimate, such as concerns that human interfaces could get in the way in a time-critical situation. This issue must be addressed in the design process. By definition, a good interface will reduce the time for human tasks.

Influenced by distributed workstations and personal computers (PCs) like the Macintosh, much more attention is now being paid to human interfaces by defense-system designers. As noted earlier, rapid prototyping involves the end user of software in the evolving requirements of a project during design. Rapid prototyping has been particularly effective in attracting high-level users to consideration of human interface questions. Starting at the interface is a painless way to encourage top-down, system design practices that have the best chance of meeting real requirements. It encourages procurement organizations to overcome institutional compartmentalization and get real end users deeply involved in an evolving design process. This news is good but probably not good enough.

A strong tendency remains to change traditional displays slowly, if at all, because of concerns about interfering with quick, habitual responses. A predilection also remains for displaying too much information on too many screens. One strategic project is now specified at over 300 screens containing "everything we had before" plus newly available information. Somehow, in the flood of so much information, the critical essentials must be synthesized and presented to the decision maker. The lack of such synthesis seems to be the common thread in the famous incidents cited here and elsewhere. Developers and end users must continue to address how to synthesize critical information in the development of effective strategic and tactical battle-management systems.

The complexity and power of many modern systems permits little tolerance for errors and little time to make – and correct – operational decisions. Significant cross-disciplinary research should examine human interfaces and how they can reduce the effects of human failings in decision making and threat evaluation under high pressure of stress and time. The research should involve psychologists, multimedia communication specialists, graphic artists, and end users from military and civilian organizations that require good human interface tools. The headlines from places like Three Mile Island, the Persian Gulf, and scenes of air crashes, certainly cry out for this kind of research.

Network Security and Software Infiltration

Despite a long history of demanding specialized computers, the defense sector now cannot ignore the revolution in networked workstations that is clearly under way throughout the business, technical, and military worlds. Workstations provide localized tools with the possibility of good human interfaces and excellent graphics. Their cost is extraordinarily low at each performance level. Mass-produced, open, commercial, networked desktop computers and their diverse software base, represent what may be the nation's most important area of technological leadership. The military today recognizes that it can't beat the cost, performance, and ease of use of these tools simply by producing its own small quantities. As a result, mass-produced, commercial desktop computers, their *public* software, and their *public* network technologies are rapidly spreading through the military and are being used to handle information and data at all levels of secrecy.

At the same time, a series of highly publicized incidents in the civilian world demonstrate the vulnerability of such computer systems. The public has been startled by PC and network viruses and worms. Commercial and government computers have been threatened by international hacker groups, with ominous names like the Chaos Club of Hamburg or with alleged links to the KGB[1], who break into sensitive (though apparently not secret) computers and by supposedly unassociated programmers who presumably by accident bring huge networks to their knees. It is no wonder that network security and software infiltration these days makes those involved with defense computing very nervous.

The National Security Agency (NSA) has organized the National Computer Security Center (NCSC). The NCSC has prepared a classification scheme for "trusted computer systems," described in the so called Orange Book[2] (see Appendix). Many familiar, widely used operating systems are (or could be) certified at the C2 level, including VAX VMS version 4.3 and higher, CDC NOS, and UNIX systems in general. Very little exists at higher levels of protection. IBM is *aiming* at B2 for its "secure XENIX" system and for its new environment of integrated software for development. B3 operating systems must have secure domains "small enough to be subjected to analysis and tests," and A1 systems must suffer proof of security through formal specification and verification techniques. Honeywell's Secure Communications Processor (SCOMP) is an experimental system at the A1 summit of the NCSC classification. Only twenty copies of SCOMP have been installed because of its limited scope and poor performance.

---

[1] John Markoff, "West Germans Raid Spy Ring That Violated U.S. Computers," *New York Times*, March 3, 1989; see also Markoff's article in *The New York Times* , March 4, 1989.

[2] *Department of Defense Trusted Computer System Evaluation Criteria, DOD 52.28-STD* (Department of Defense, Washington, D. C., 1985).

The goal of secure operating and network system software at the A1 level requires that a formal specification be written that defines precisely what the software will do. It is possible to prove formally that design of the code generated from such a specification, in fact, does what the spec says it should. The problem lies in writing a correct spec. Despite a limited scope, the formal specification for SCOMP requires the equivalent of over twenty dense pages of mathematical spec notation. This amount must be somewhere near (some evidence suggests it is beyond) the maximum that humans can produce with reasonable assurance that it is error-free. Note that even at this A1 summit of security, there is no requirement to prove that the code implemented is equivalent to the verified design!

Formal specification languages result in code about ten times longer than the spec. Program source code and specification languages present the same scale of difficulties in assuring their correctness. Formal methods, therefore, essentially extend by an order of magnitude the size of assuredly correct software segments, from about 1,000 lines to about 10,000. Though this may be extendable somewhat by a hierarchical application of formally proven modules, one soon gets into the problems of measurable software reliability discussed in an earlier section. A further problem is that as the size of formal mathematical proofs increases they quickly reach a point where confidence in the correctness of the proof itself is at issue.

Since general operating systems, like VMS or UNIX, typically involve hundreds of thousands of source lines of code, it is not reasonable to expect the whole of such a system to be verified formally at the A1 level of security. The trick is to restrict the problem to key security barriers that control access to data and to and from the network. If these barriers can be kept localized and small enough, it should be possible to prove them formally. This requirement of localizing secure points puts a strong demand on the overall architecture of the operating system and underlying hardware, a demand which may compromise other requirements. At the least, the process requires rethinking and redesigning the architecture of existing operating systems and computer families that have gained widespread popular acceptance. The changes required may well encounter insurmountable marketplace resistance. It is no wonder, then, that there has not been a stampede to develop the highest standards of "trusted computers systems" for networked civilian workstations. The military is caught between its need for secure systems and the enormous advantages of the technologically superior, though fundamentally insecure, distributed-workstation products available in the private sector.

Unclassified information seems to indicate that secret data on secure networks is not vulnerable to even the most talented hacking. Such networks are physically isolated, with no connections to public networks or the telephone system. When data is allowed to leave a secure,

often Faraday-caged,[1] site, it is tightly controlled and heavily encrypted. Networks mixing secret and open workstations are not presently in use. The NSA is addressing the technological challenge of securing the secret part of such a mixed environment. It is evaluating hardware, called Blacker Boxes, which is designed to isolate secret parts of networks and to screen data crossing the network fence.[2]

In general, then, the problem of distributed computing and network security is being addressed through approaches focusing on hardware, software, and consciousness raising. Public operating systems can be expected to evolve slowly but steadily to more secure forms. Ultimately, sometime in the twenty-first century, these systems will reach NCSC A1 levels. All these developments are good from the viewpoint of military security, and encouraging for those concerned about crisis escalation during the confusion that could result from misguided or malicious attacks on military networks or operating systems.

One big worry remains. Networks and computers – apparently at all levels of security – are now vulnerable to "Trojan horses" left by hostile infiltration of software-development groups. Software (or firmware) may be entered and changed through a variety of devious techniques, which may, for example, trigger self-destructive actions at a later time. The infiltrators may leave just a few lines of code as a "hook" for future access. Even formal methods are not an absolute barrier; such a hook could be installed in the depths of a very mathematical, lengthy, formal specification, or someone could attempt to modify the code after it had been verified.

For sensitive code related to strategic control, checking of software by two persons is common. For some nuclear weapon software, a second team has been used to disassemble code. Other conceivable, and expensive, countermeasures, can be used against infiltration of particularly critical code during development or maintenance. One such measure is the use of the *n*-version software technique described earlier, with frequent voting by the versions. The likelihood of successfully infiltrating one highly classified and secure software development group is very low. The likelihood of infiltrating two or more – and implementing hidden sabotage that behaves in the same way – is negligibly small.

A second countermeasure against infiltration is to use a strict audit trail of secret function check sums during development and maintenance of software. A simple check sum is the sum of the numerical values of all words, in a data transmission, for example. If the number is the same at both ends of the transmission, one can be reasonably satisfied the data got through correctly. The idea is to maintain a secret function of values of all words in the binary image of a program (and the formal specification) and to audit that this function value stays the same except under

---

[1] A Faraday cage is fully enclosed on all sides in a conducting metal shield so that no electromagnetic signals can escape.

[2] Ronald Elliott, "The Integrated Tactical Data Network," *Signal*, (March 1989): 53-56.

carefully supervised circumstances, when small incremental changes to the code are made with several people present. This, or a similarly onerous mechanism, could provide the necessary assurance that no unauthorized tampering has altered the code and specifications since their inception. *n*-version software techniques may already be in use for such purposes. Strict audit trails of check sum like functions of programs or specifications apparently are not in use, but in critical situations should be.

The potential consequences of the successful infiltration, unlikely as it may be, of a software development project are no less overwhelming than those of any other major breach of military defense. The vulnerability is widespread. The Soviets, for example, are exposed since their computers are primarily clones of U.S. machines. Some of their computers run on copies of standard operating-system software.[1] Viruses are known to spread like wildfire through Soviet PCs.[2] As the Soviets increase their use of networked workstations, the problem is likely to become more severe for them. They should certainly worry that Trojan horses may reside in the commercial software they have "borrowed" or been permitted to buy.

The use of computer sabotage is not restricted to countries with the capabilities of the superpowers. Such sabotage is available to any Third World country that has access to technically educated nationals or friends. A country with terrorist or other malicious inclinations can use sabotage equally easily against regional or northern enemies. A recent personal computer virus that affected users in North America and Europe was initiated in Pakistan. Though it was not a realistic threat in the military context, the attack serves nonetheless, as a reminder that sophisticated software capabilities increasingly exist in places one normally would not consider technologically advanced.

This global ability to attempt attacks on computers should be sobering. Even if our country can "trust" and protect its critical military software and systems, other countries may not be able to protect theirs. The result is a potential threat to stability, at least in regional conflicts, from deliberate, controlled attempts to sabotage C^3I. Throughout history it has been widely accepted as good military practice to attack command and communication systems, except for communication links required for retreat or surrender. Does this attitude remain acceptable in a world that is tightly coupled in terms of security? Where the horizon of instantaneous communication now covers the globe? Where the short time scale of escalation to even regional nuclear weapons could result in disastrous consequences? Such questions should be the subject of another study.

---

[1] A visit to a Soviet computer center is like a science fiction trip back in time to early U..S.. installations. See, also, Seymour Goodman, William McHenry, Ross Stapleton, "General Purpose Computer Systems in the Warsaw Pact Countries, " *Signal*, (December 1984), 87-101.

[2] Gary Chapman, personal communication, August 10, 1989.

Conclusions and Recommendations

I have covered here a number of topics that have something to do with computers and defense, but otherwise might seem unrelated at first glance. The common thread is as important as it may be subtle: computer systems are intimately involved with human beings and their decision making. In the military context computer systems, improperly conceived or applied, could thus amplify the fog of war to a degree never before encountered — and in an era where such confusion can lead to escalation with enormous consequences. The problem requires understanding of two related general issues: the reliability of human-computer systems and the question of what can be expected reasonably of computers and their software, given the technology, talent and budgetary resources available for their design, production, and testing.

To use computers with any ignorance of reliability is to saunter irresponsibly into the battlefield computer fog bank. The issue of reliability is indeed being addressed, with great thoroughness, in well identified and quantifiable areas like communications. However, as this report has tried to demonstrate, there is a strong need for more attention and a deeper understanding by decision makers where questions of reliability are less obvious and more difficult to quantify. This study has pointed, in particular, to development of software and firmware, network and system development security, and the human interface channel between the computer tool and the human.

Appraisals of what is possible now appear somewhat more realistic. In the early 1980s outrageous claims were being made about what computers could do in artificial intelligence and missile defense. Unfortunately, the tendency continues for many in the civilian and military worlds to confuse technological wishes with real options. A serious problem results for policy making in science and technology. Researchers often intensify the problem with exaggerated, fund seeking, statements to the media. Typical are overblown promises after significant but embryonic advances and announcement of discoveries before they are substantiated. (The recent public excitement over warm-temperature superconductivity and the "discovery" of significant energy production in cold fusion come to mind.) It is important for policy makers to remember that society's needs and wishes are not strongly correlated with what technology can deliver. Sometimes it is possible to meet wishes. Often it is not.

Similarly, resource limitations are not just budgetary but also involve the extent of the nation's (or the world's) talent pool. Complex projects require talented personnel who are trained and knowledgeable at all levels including management and execution. Managers familiar with large system projects know that the key to success is to contain the complexity from the start. Complexity is difficult to quantify, but it is clear that a project team can successfully get its arms around just so much complexity. The nation's defense systems, including those that are operational and those under development, taken together form an even bigger system, in this

sense. The number and variety of individual subsystems certainly are important factors in the measure of complexity of this super-system. The defense and political leadership must understand the complexity of the overall defense system so that it can be effectively staffed and managed. The numbers of weapons and defense computer systems should be reduced to levels where one can be confident of having adequate talent resources available to manage them at acceptable levels of reliability.

This line of thinking takes one to even more serious concerns. Defense systems are a subset of national activities in technology. Talent that goes into technology for defense is not available for technology activities in other fields that are critical to the nation's technological viability and its ability to compete in the international economy. Management of talent resources has widespread implications that cry out for attention.

The following are specific recommendations regarding policy research:

- Underlying this report has been the conjecture that increasing the fog of war is destabilizing. The consequences of this conjecture are so important that the subject should be put on a firm research foundation. Specifically researchers need to analyze carefully the impact that unreliable, ineffective, and infiltrated computer systems can have on stability in times of crisis. Study should also be directed at questions associated with the appropriateness of and the possibilities and mechanisms for international action to prevent destructive penetration, by means of software or networks, of one country's military computers by those of another country.

- The general level of understanding of the capabilities, limitations, and costs of computer projects by decision makers must increase. A well-staffed center should provide politically neutral technical advice about computers and their application. The subject should also be addressed with explicit support for subgroups under such government advisory entities as the Assistant to the President for Science and Technology at the White House and the Congressional Office of Technology Assessment (OTA).

- Policy makers must examine the quantity and quality of resources in personnel talent resources, and the competing demands for these resources by high-tech projects in the defense and civilian sectors. This difficult effort must involve more than just a count of graduate degrees in electrical engineering; it must get at a measure of the availability of key technical talent and its match to project requirements. Again, the effort could be organized under the auspices of an organization at the level of the White House science advisor or Congress. The armed forces should adjust promotion practices to encourage those with computer expertise to stay in the military.

- The government and DoD should return to intensive funding of basic rather than applied research in computer science. Specifically, support should cover work on connectionist approaches, software-engineering tools and methods and project management, human interfaces, and on improving the time to input knowledge to and the verification of expert systems.

- The DoD must invest much more of the time of its most experienced and knowledgeable military officers in the continuing and evolutionary definition of system requirements, and in the knowledge engineering of expert systems. The DoD should research means to increase such involvement through policy, incentive, and the use of attractive tools and procedures. In the same spirit, it should encourage, as part of the procurement process, the extensive use of rapid-prototyping techniques involving operationally experienced officers.

- Priority in funding should go to cross-disciplinary research into human interfaces, including research into how human interfaces can reduce the effects of human failings in decision making and threat evaluation under high stress and pressures of time. The research should involve psychologists, specialists in multimedia communication, graphic artists, and end users from military and civilian organizations that require good human interfaces.

Acknowledgments

I would like to thank Sidney Drell for his encouragement and guidance about the direction this research could take. He also introduced me to many knowledgeable and helpful people who made it possible for me to pursue this subject.

I thank David Weiss, Charles Zraket, Charles Hebel, Eamon Barrett, and James Marsh for their hospitality at the Software Productivity Consortium, Mitre Corporation, the XEROX PARC, Lockheed Missiles and Space Corporation, and Advanced Decision Systems, and for arranging meetings with their colleagues. I have had very useful and informative discussions with Gideon Akavia, Daniel Bobrow, Ashton Carter, Bruce Denby, Paul Edwards, Jim Greenwood, Tim Harrison, Charles Hebel, Barry Horowitz, Ted Jarvis, Allan Jaworski, John Knight, Doug Lenet, Frank Maginnis, Jim Marsh, Duncan McQueen, Elisabeth Paté-Cornell, Ted Postol, John Quilty, John Ruddy, Barry Sussman, Peter Tasker, David Weiss, and Charles Zraket. I particularly thank Bill Agresti, Gideon Akavia, Eamon Barrett, Gary Chapman, Sidney Drell, James Marsh, William Perry, John Quilty, John Ruddy, Andrea Weiss, David Weiss, and Charles Zraket for carefully reading the manuscript and making useful suggestions. I also thank Miriam DeJongh and Pat Mascione for their efforts in editing and preparing the manuscript. Despite the help from all these people, any errors of fact or interpretation in this paper are entirely my responsibility.

This work was carried out while I was on sabbatical from the Fermi National Accelerator Laboratory, which provided part of my support under contract with the U.S. Department of Energy. I thank the Carnegie Foundation for the other part which came through a Carnegie Science Fellowship at the Center for International Security and Arms Control at Stanford University.

Table 1

Human Failings: Subjective Assessment of Probability of Biases[1]

Representativeness	Availability	Adjustment and anchoring
Insensitivity to prior probability of outcomes	Biases due to the retrievability of instances	Insufficient adjustment
Insensitivity to sample size	Biases of imaginability	Biases in the evaluation of conjunctive and disjunctive events
Misconceptions of chance	Illusory correlation	Anchoring in the assessment of subjective probability distributions
Insensitivity to predictability		
The illusion of validity		
Misconceptions of regression		

[1] Summarized from D. Kahneman, P. Slovic, A. Tversky, *Judgement Under Uncertainty: Heuristics and Biases*, (New York: Cambridge University Press, 1982).

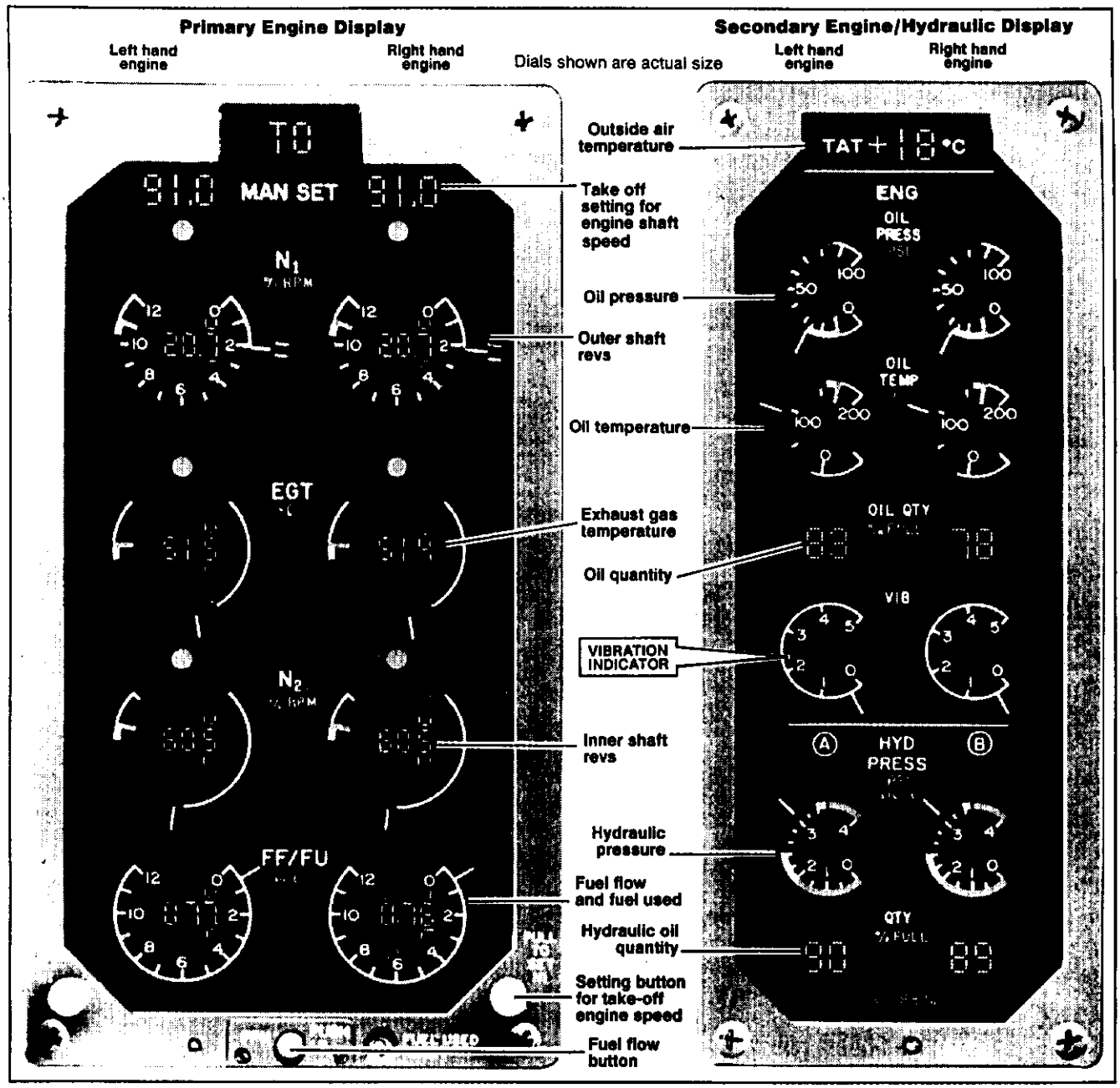

Figure 1 - Electronic display panel as seen by pilots in the Boeing B-737 that crashed on the M1 expressway in England in January 1989. Although electronic, the display graphics simulates electro-mechanical displays to which pilots have been accustomed. This system does not take advantage of the capability of computerized displays to cluster problem indications and options in one place. (Reprinted from *Times* [London], January 19, 1989)

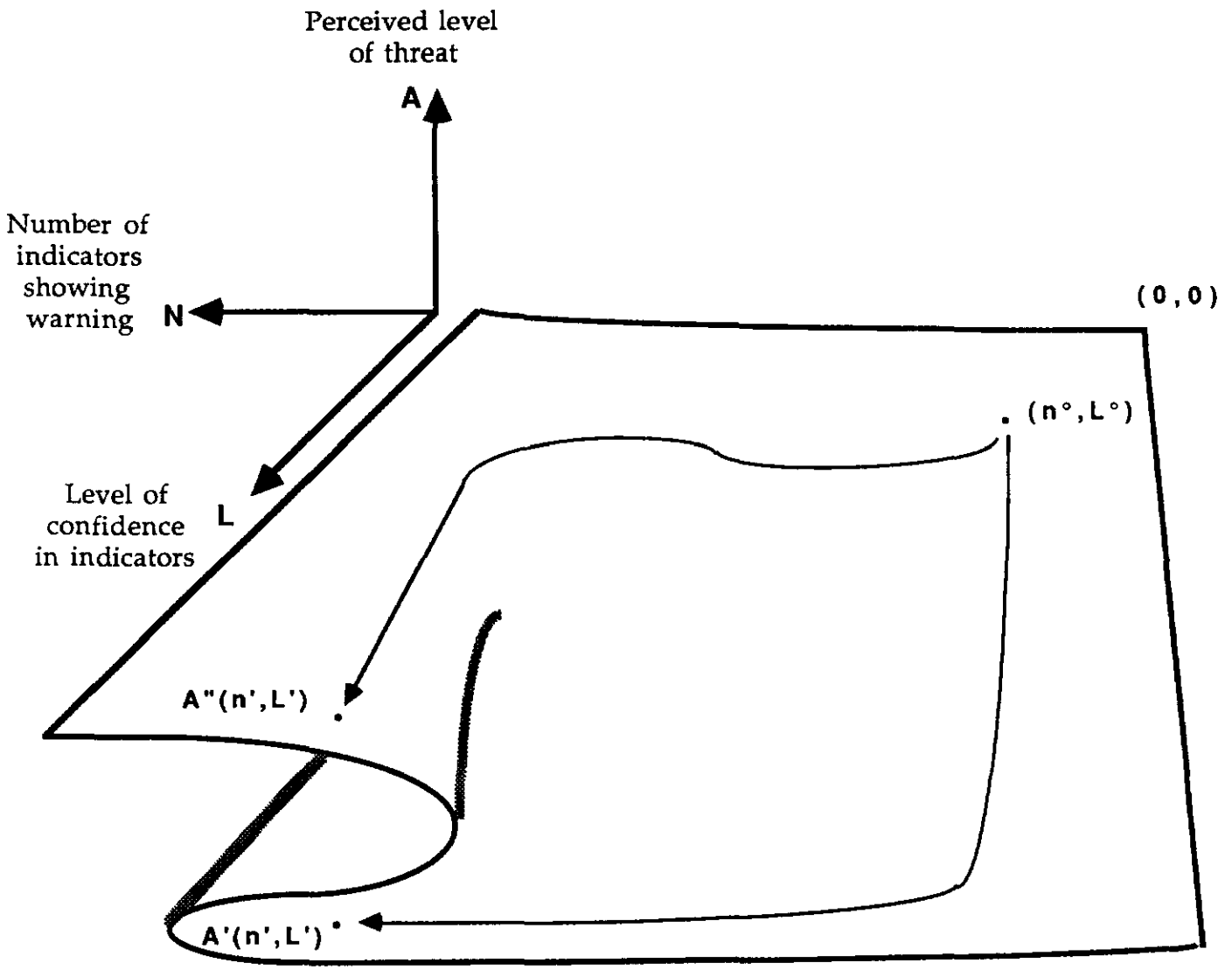

Figure 2 - Perceptual hysteresis in threat assessment. The folded threat assessment surface shown here is an example of how multivalued assessment functions of the two variables, number of active threat indicators and level of confidence, can develop. In this figure the analyst assessment (A) is measured on the vertical axis. The number of indicators (N) and level of confidence (L) are measured on the two horizontal axes. It is possible for an analyst to come to different assessments (A',A") at the same values of n' and L' depending on the path travelled from the quiescent initial condition (n°, L°). If confidence increases after indicators turn on, experiments indicate that the assessment level (A") will be higher than if the number of indicators increases after confidence in them is already higher (A') This is an example of ambiguity in perception caused by "preconditioning." (Adapted from Alexander Woodcock, *Signal*, (July, 1988): 51-57 and (March, 1989): 67-72.)

Appendix

NCSC Classification of Trusted Software Systems[1]

The classes of systems recognized under the trusted computer system evaluation criteria are as follows. They are presented in the order of increasing desirability from a computer security point of view.

Class (D): Minimal Protection

This class is reserved for those systems that have been evaluated but that fail to meet the requirements for a higher evaluation class.

Class (C1): Discretionary Security Protection

The Trusted Computing Base (TCB) of a class (C1) system nominally satisfies the discretionary security requirements by providing separation of users and data. It incorporates some form of credible controls capable of enforcing access limitations on an individual basis, i.e., ostensibly suitable for allowing users to be able to protect project or private information and to keep other users from accidentally reading or destroying their data. The class (C1) environment is expected to be one of cooperating users processing data at the same level(s) of sensitivity.

Class (C2): Controlled Access Protection

Systems in this class enforce a more finely grained discretionary access control than (C1) systems, making users individually accountable for their actions through login procedures, auditing of security-relevant events, and resource isolation.

Class (B1): Labeled Security Protection

Class (B1) systems require all the features required for class (C2). In addition, an informal statement of the security policy model, data labeling, and mandatory access control over named subjects and objects must be present. The capability must exist for accurately labeling exported information. Any flaws identified by testing must be removed.

Class (B2): Structured Protection

In class (B2) systems, the TCB is based on a clearly defined and documented formal security policy model that requires the discretionary and mandatory access control enforcement found in class (B1) systems to be extended to all subjects and objects in the ADP system. In addition, covert

---

[1] From *Department of Defense Trusted Computer System Evaluation Criteria, DoD, 52.28-STD,* (Department of Defense, Washington, D. C., 1985).

channels are addressed. The TCB must be carefully structured into protection-critical and non-protection-critical elements. The TCB interface is well-defined and the TCB design and implementation enable it to be subjected to more thorough testing and more complete review. Authentication mechanisms are strengthened, trusted facility management is provided in the form of support for system administrator and operator functions, and stringent configuration management controls are imposed. The system is relatively resistant to penetration.

Class (B3): Security Domains

The class (B3) TCB must satisfy the reference monitor requirements that it mediate all accesses of subjects to objects, be tamperproof, and be small enough to be subjected to analysis and tests. To this end, the TCB is structured to exclude code not essential to security policy enforcement, with significant system engineering during TCB design and implementation directed toward minimizing its complexity. A security administrator is supported, audit mechanisms are expanded to signal security-relevant events, and system recovery procedures are required. The system is highly resistant to penetration.

Class (A1): Verified Design

Systems in class (A1) are functionally equivalent to those in class (B3) in that no additional architectural features or policy requirements are added. The distinguishing feature of systems in this class is the analysis derived from formal design specification and verification techniques and the resulting high degree of assurance that the TCB is correctly implemented. This assurance is developmental in nature, starting with a formal model of the security policy and a formal top-level specification (FTLS) of the design. In keeping with the extensive design and development analysis of the TCB required of systems in class (A1), more stringent configuration management is required and procedures are established for securely distributing the system to sites. A system security administrator is supported.