MEMORANDUM
RM-6213-PR
JANUARY 1970

SOME INFORMATION PROCESSING IMPLICATIONS OF AIR FORCE SPACE MISSIONS: 1970-1980

Barry W. Boehm

PREPARED FOR:
UNITED STATES AIR FORCE PROJECT RAND

The RAND Corporation
SANTA MONICA • CALIFORNIA

55

MEMORANDUM
RM-6213-PR
JANUARY 1970

SOME INFORMATION PROCESSING IMPLICATIONS OF AIR FORCE SPACE MISSIONS: 1970-1980

Barry W. Boehm

This research is supported by the United States Air Force under Project RAND—Contract No. F44620-67-C-0045—monitored by the Directorate of Operational Requirements and Development Plans, Deputy Chief of Staff, Research and Development, Hq USAF. Views or conclusions contained in this study should not be interpreted as representing the official opinion or policy of the United States Air Force.

The RAND Corporation
1700 MAIN ST. • SANTA MONICA • CALIFORNIA • 90406

This study is presented as a competent treatment of the subject, worthy of publication. The Rand Corporation vouches for the quality of the research, without necessarily endorsing the opinions and conclusions of the authors.

Published by The RAND Corporation

PREFACE

The text of this Memorandum, prepared as a briefing for the Air Force Scientific Advisory Board meeting on "Military Preparedness in Space," indicates some critical information processing problems implied by Air Force space objectives (and also by other Air Force objectives) and recommends steps for attacking these problems. It should be of interest to military and civilian space planners, and to military information system planners in general.

This Memorandum has benefited greatly from discussions with J. P. Haverty and W. H. Ware of The RAND Corporation.

- FOR SOME COMPUTING CAPABILITIES, USAF MUST ACTIVELY PUSH R&D
- ON-BOARD COMPUTERS: BUY 50% - 100% EXCESS CAPACITY TO MINIMIZE TOTAL SYSTEM COSTS
- STS: SOFTWARE DEVELOPMENT IS ALREADY ON THE CRITICAL PATH
- SOFTWARE CERTIFICATION PROBLEM UNDEREMPHASIZED; CONSIDER DEDICATED SOFTWARE TEST FACILITY

SUMMARY

SUMMARY

Most military space operations during the 1970s will not strain the available information-processing capabilities. But there are some operations--real-time image processing, multi-sensor data analysis, decision-oriented displays, and others--for which the Air Force will not be able to reach "on the shelf" and find tools capable of doing the job. The USAF will have to settle for reduced capabilities in these areas, unless space-mission planners more thoroughly investigate their detailed information processing requirements and couple them more effectively to the USAF R&D program in information processing.

In this Memorandum (the text of a briefing), an analysis of observed software cost trends indicates that the overall cost of an on-board computing system is generally minimized by procuring computer hardware with at least 50 percent to 100 percent more capacity than is absolutely necessary.

The proposed Space Transportation System (STS) *will* strain the available information processing capabilities. Historical data on similar software projects indicate that six or seven calendar years are probably required to design, develop, and check out the software for a project of the magnitude and complexity of STS. To make sure that software does not slip the overall schedule, STS planners must begin to design the software *now*. Also, the USAF should push R&D on high-capability flight computers for STS.

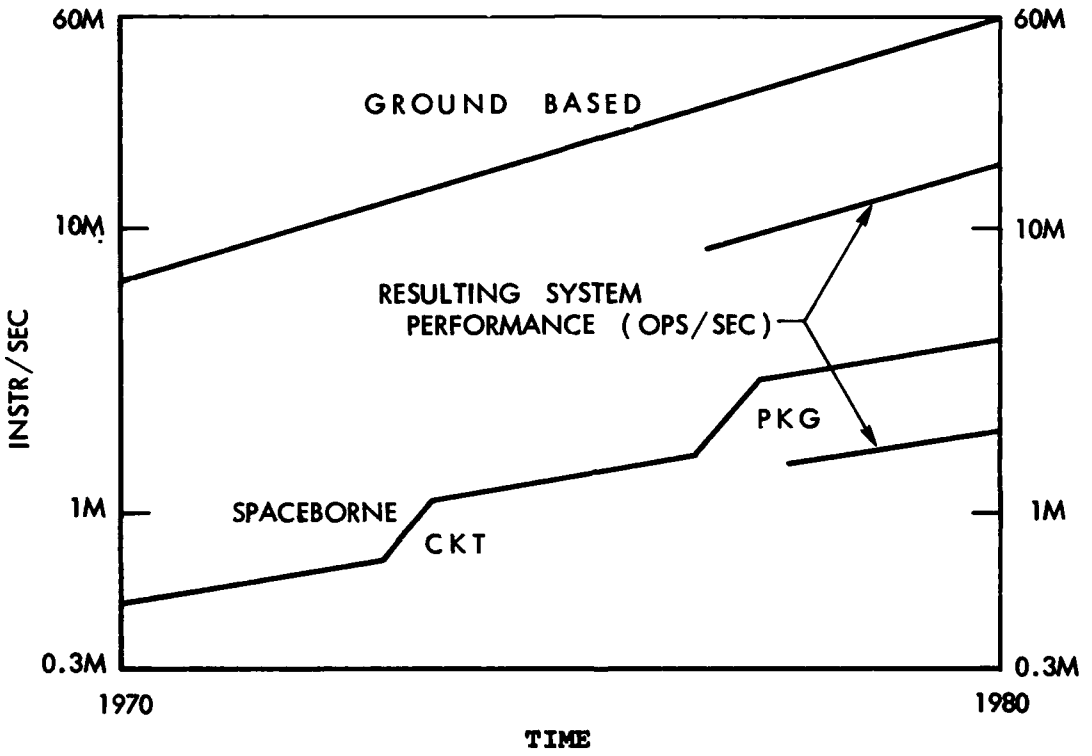
Preprogrammed space software in the seventies will allow many more user options. Serious consideration is being given to "programmer-astronauts" and real-time software modification to provide non-preprogrammed flexibility to USAF space-mission operations. But, coupled with our inadequate capabilities to check out and certify software, such measures could have disastrous consequences in escalating strategic crises or degrading critical defense

- FOR SOME COMPUTING CAPABILITIES, USAF MUST ACTIVELY PUSH R&D
- ON-BOARD COMPUTERS: BUY 50% - 100% EXCESS CAPACITY TO MINIMIZE TOTAL SYSTEM COSTS
- STS: SOFTWARE DEVELOPMENT IS ALREADY ON THE CRITICAL PATH
- SOFTWARE CERTIFICATION PROBLEM UNDEREMPHASIZED; CONSIDER DEDICATED SOFTWARE TEST FACILITY

SUMMARY

capabilities because of undetected errors in software options or modifications. The Air Force could markedly improve the situation by following its hardware development philosophy and establishing a dedicated facility for software testing and certification, along with operational procedures for using the facility during a major software project development.

CONTENTS



COMPUTER SPEED

HARDWARE TRENDS AND OPERATIONAL REQUIREMENTS

This Memorandum opens with some estimates of computer hardware capabilities that the Air Force can reasonably expect to find "on the shelf" during the 1970s. Following these estimates are discussions of 1) how best to use these capabilities when they are easily sufficient to do the job, and 2) some space operations for which these capabilities will be either barely sufficient or insufficient.

Amdahl [1] has indicated the probable computing capabilities that the current pace of technology will yield to support USAF missions in the seventies. Due primarily to advances in large-scale integrated circuit (LSI) technology, the speed of ground-based computers will increase from 6,000,000 instructions/sec to 60,000,000 instructions/sec between 1970 and 1980. Also, during this period the speed of spaceborne computers will increase from 500,000 instructions/sec to 4,000,000 instructions/sec due to advances both in LSI and circuit-packaging technology.

However, Amdahl's analysis indicates that--to the extent that transfers to and from high-speed memory are unpredictable (due to real-time interrupts, accesses to lower-speed memory, etc.)--the actual performance realized by the computer as a system will drop by a factor of four (to 15,000,000 operations/sec) for ground-based computing and by a factor of two (to 2,000,000 operations/sec) for spaceborne computing.

The advances in speed will be accompanied by advances in hardware reliability and reductions in cost, size, weight, and power requirements for spaceborne computers. For example, Amdahl estimates that the size of an 8,000,000-bit memory will decrease from 75 cubic feet in 1970 to 2 cubic feet in 1980.

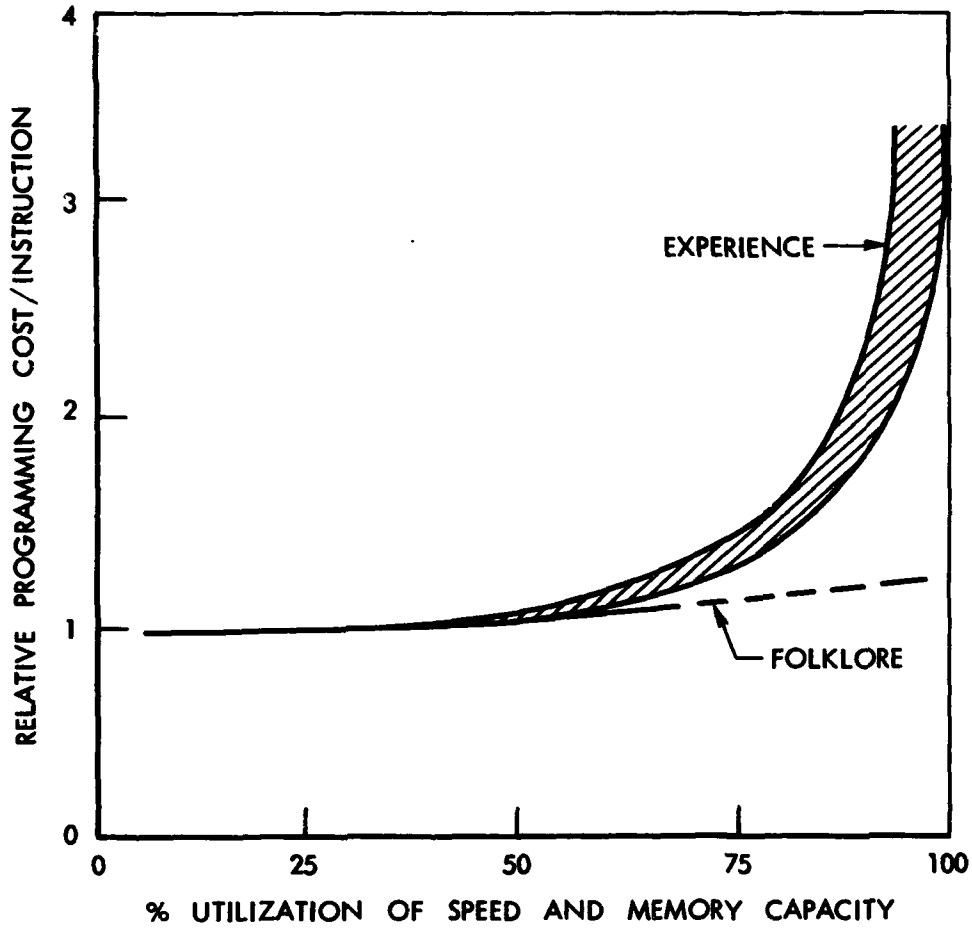
MOST MILITARY SPACE OPERATIONS
WON'T HAVE TO STRAIN
THE COMPUTER'S CAPABILITY

— BUT THIS DOESN'T GUARANTEE
THAT THEY WON'T

MOST JOBS WILL NOT HAVE TO STRAIN THE HARDWARE

The computing capacity available during the 1970s will easily accommodate most of the information processing requirements of USAF space missions. Current concepts of navigation satellite systems, life support and environmental monitoring for manned missions, and basic systems for attitude control and trajectory guidance and control easily fall within this category. (Some operations that will strain this computing capacity are discussed below.)

However, it is still very easy to produce a situation in which computing capacity is strained. One way is to procure a small computer to decrease hardware costs, when larger ones are available. Another is to absorb any excess computing capacity with marginally useful tasks. As the next figure indicates, both of these practices can gravely inflate software costs.



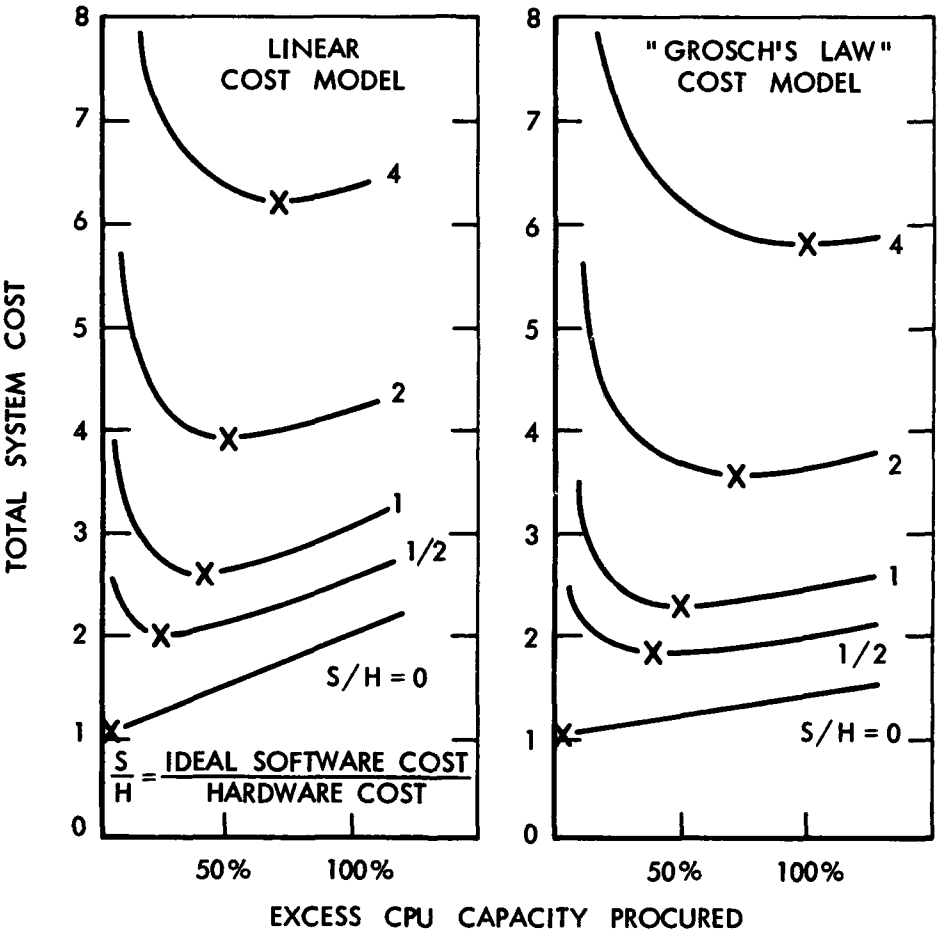
ON-BOARD COMPUTING: SOFTWARE COSTS

ON-BOARD COMPUTING: SOFTWARE COSTS

This figure shows how dramatically software costs increase as one strains the capacity of the computer's central processing unit. It is based on the experience of North American Rockwell's Autonetics Division in developing a large body of software for aircraft, missile, and spaceborne computers [2].

Why does the curve look like this, and not like the "folklore" curve? Primarily because when one is pushing the computer's capacity, slight gains in program efficiency can only be bought at the cost of logical complexity. Machine language must be used instead of higher-order languages like FORTRAN; several elements of data must be packed into a single machine word; and many tricky programming shortcuts must be employed with scaling, reusable portions of code, and the like. All of these make the program not only harder to write but also much harder to check out, modify, and coordinate with other operations.

Thus, the figure indicates that current folklore-based practices of specifying computer hardware by sizing the data-processing task and adding perhaps 15 percent for contingencies are highly inappropriate if software constitutes an appreciable portion of the cost of the data-processing system, or if the sizing is subject to any significant error. The next two figures quantify this conclusion.



ON-BOARD COMPUTING: TOTAL SYSTEM COSTS

ON-BOARD COMPUTING: TOTAL SYSTEM COSTS

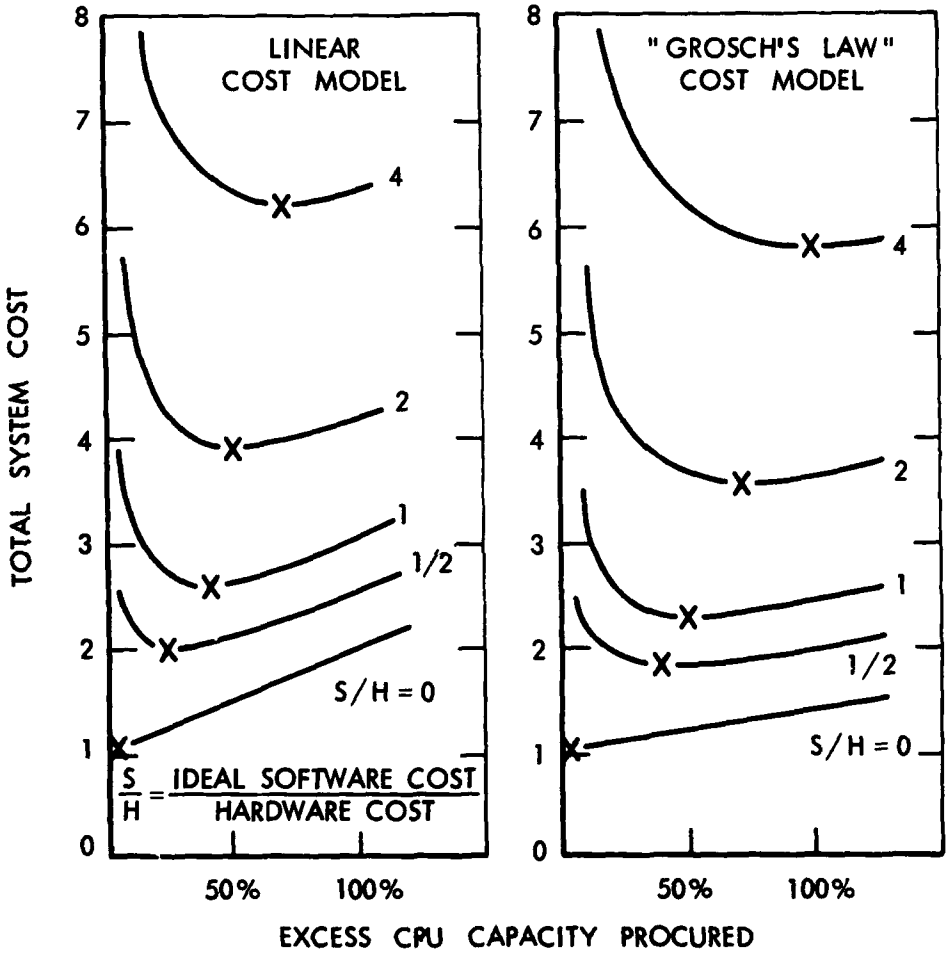
Suppose that one has sized a data-processing task and determined that a computer of one-unit capacity (with respect to central processing unit speed and size) is required. The figures on the opposite page show how the total data-processing system cost varies with the amount of excess Control Processing Unit (CPU) capacity procured for various estimates of the ratio of ideal software-to-hardware costs for the system.* The calculations are based on the previous curve of programming costs and two models of hardware cost: the linear model assumes that cost increases linearly with increases in CPU capacity; the "Grosch's Law" model assumes that cost increases as the square root of CPU capacity. Sharpe's data [3] indicates that most applications fall somewhere between these models.

It should be remembered that the curves are based on imprecise observations; they clearly cannot be used in "cookbook" fashion by system designers. But even their general trends make the following points quite evident:

1) Overall system cost is generally minimized by procuring computer hardware with at least 50 percent to 100 percent more capacity than is absolutely necessary.
2) The more the ratio of software-to-hardware cost increases (as it will markedly during the seventies), the more excess computing capacity one should procure to minimize the total cost.
3) It is far more risky to err by procuring a computer that is too small than one that is too large. This is especially important since one's initial sizing of the data-processing job often tends to underestimate its magnitude.

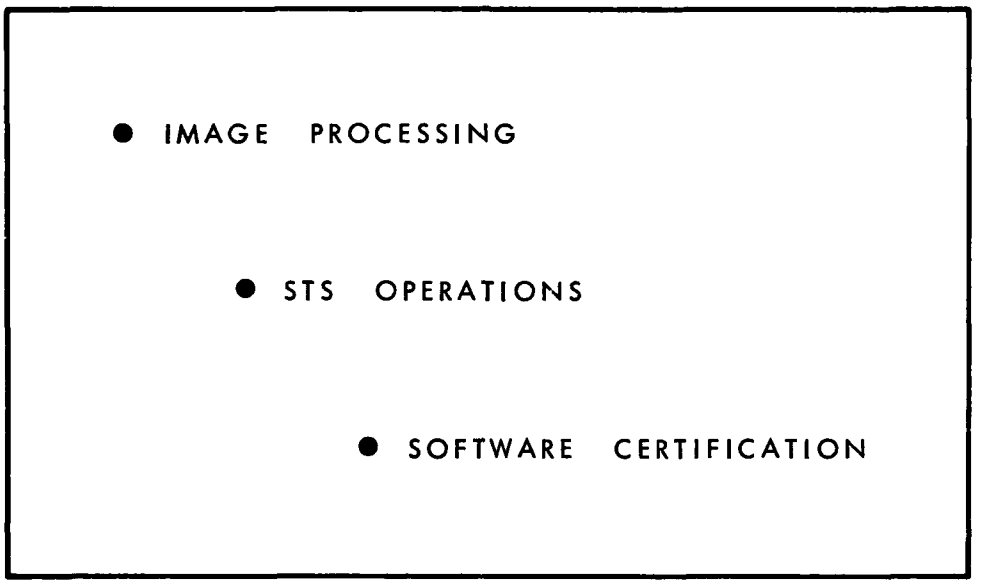
---

*"Ideal software" costs are those that would be incurred without any considerations of straining hardware capacity.



ON-BOARD COMPUTING: TOTAL SYSTEM COSTS

Of course, buying extra hardware does not eliminate the need for good software engineering thereafter. Careful configuration control must be maintained to realize properly the benefits of having extra hardware capability, as there are always strong Parkinsonian tendencies to absorb excess capacity with marginally useful tasks.



SOME CHALLENGING REQUIREMENTS

SOME CHALLENGING REQUIREMENTS

As indicated above, the computer hardware available during the seventies will allow system designers the "luxury" of procuring excess computer capacity for most space operations. However, some information-processing requirements of the military space program will present strong challenges, including the correlation and analysis of data from large numbers of independent sensors; the analysis and design of decision-oriented displays for military commanders; and the general problem of software engineering, or management of large software projects. This Memorandum, however, concentrates only on three representative topics: 1) real-time image processing, 2) Space Transportation System (STS) operations, and 3) software certification.

	OPERATIONS/ IMAGE
ALIGN, FILTER, DISPLAY	$10^8 - 10^{10}$
ENHANCE CONTRAST	$10^8 - 10^9$
CROSSCORRELATE WITH TEMPLATE	$10^7 - 10^{14}$
RETRIEVE, DISPLAY PREVIOUS IMAGE	$10^8 - 10^{10}$
DIFFERENCE TWO IMAGES	$10^7 - 10^8$
CROSSCORRELATE DIFFERENCE WITH TEMPLATE	$10^7 - 10^{14}$

IMAGE PROCESSING

(1 IMAGE ~ 10^7 POINTS, 10^8 BITS)

REAL-TIME IMAGE PROCESSING

The requirements for real-time image processing are implicit in many military operations involving such activities as surveillance, strategic warning, and damage assessment. The figure opposite indicates the rough number of computer operations required to perform a typical sequence of tasks on a rather modest image containing 10^7 points and 10^8 bits of information. (For comparison, the Mariner 6 and 7 photographs had about 10^7 bits [4]; high-resolution Luner Orbiter pictures have $5 \cdot 10^8$ bits; high-quality aerial photographs have up to 10^{11} bits [5].) The right-hand number in the operations/image column represents the number of operations required if no effort is made to conserve operations; the left-hand number reflects straightforward use of standard programming shortcuts for this type of task. The cross-correlation entries are an exception; there the range reflects whether one is cross-correlating with respect to a single point or with respect to the entire image of 10^7 points.

Thus, even using currently efficient techniques, one cannot perform these tasks in much less then 10^9 computer operations. On a 1980 computer, realizing $15 \cdot 10^6$ operations/sec (see figure, p. x), this would take about 67 seconds: functional for some activities, but not very satisfactory for military command and control. Even given the ideal computing power of $60 \cdot 10^6$ instructions/sec, the task would take about 17 seconds, and this on a relatively low-quality image. If the Air Force wishes to use images of high quality, or perform in a real-time environment more extensive image-processing tasks, it will be necessary to specifically define the image-processing functions required, and to actively stimulate research and development efforts towards performing them rapidly.

- SERIAL VS PARALLEL PROCESSING
- STORAGE AND RETRIEVAL
- GROUND VS SPACEBORNE
- AUTOMATED PATTERN RECOGNITION

IMAGE PROCESSING CONSIDERATIONS

IMAGE PROCESSING CONSIDERATIONS

Parallel processing might seem an ideal solution to the real-time image-processing problem because many of the image-processing tasks involve identical local operations among neighboring image points. Thus, to speed things up by a factor of N, one simply buys N parallel processors. However, recent experience on the Illiac IV experimental multiprocessor indicates that even jobs highly suited for parallel processing still carry a residual serial procesing burden of about 5 percent. This experience, if typical, limits the gains due to parallel processing to a factor of 20.

Over a period of years, an image archive could build up to a storage requirement of 10^{13}-10^{15} bits, with associated problems of defining an appropriate indexing and cataloging scheme for images and a related query language for users. Also, the space-time tradeoff question--either storing multiple copies of an image or storing a single copy and reprocessing it on demand--must be appropriately resolved.

One promising role for man in space involves the interpretation of image data and subsequent reconfiguration of sensors, and transmittal of summary information to the ground. This would greatly reduce the daily communications and ground-processing load; however, if during a crisis the ground operators want to be able to do their own processing, one would have to provide the communications and ground-processing hardware anyway. Also, another alternative for getting man's capabilities into space deserves further consideration: the use of remote-controlled teleoperators [6,7].

The Air Force should not expect any miracles of automated pattern recognition to sweep away the image-processing problem. If specific mission-oriented image-processing functions can be defined and given R&D priority, however, the USAF can expect the results to yield solid, if perhaps unspectacular, improvements in processing algorithms, which also can be called useful contributions to pattern-recognition research.

- ON-BOARD LAUNCH CHECKOUT AND CONTROL
  - OF STS VEHICLE
  - OF VARIOUS PAYLOADS
- WHOLE NEW LEVEL OF FLIGHT CONTROL

INFORMATION PROCESSING: STS INNOVATIONS

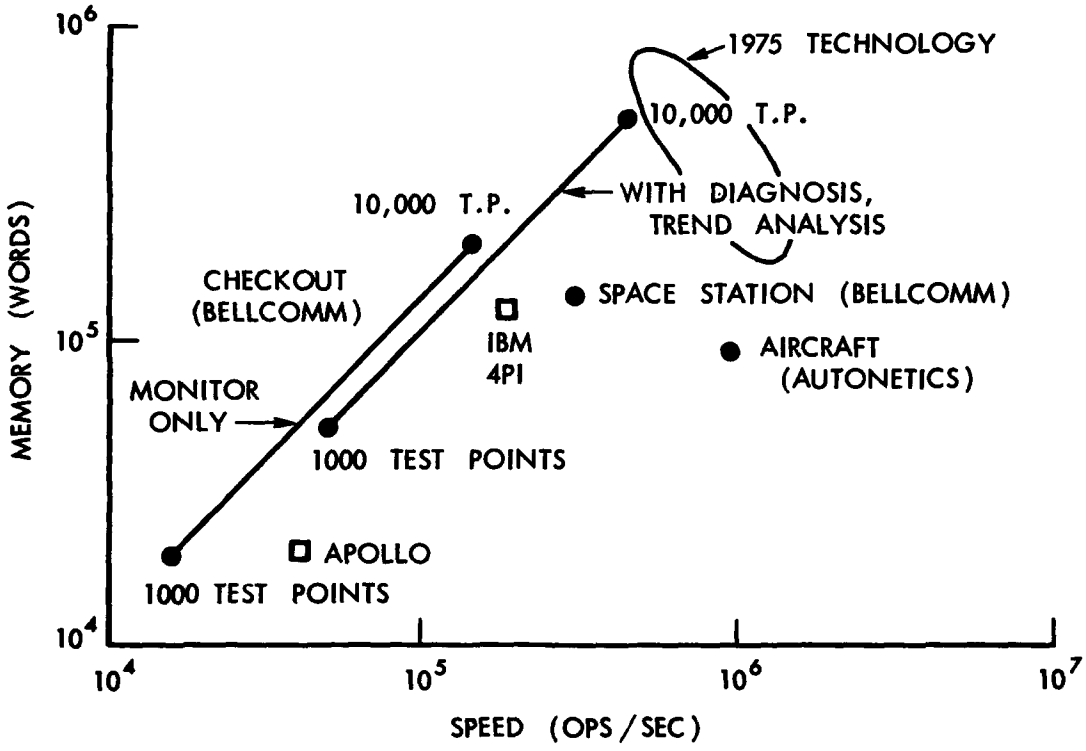
INFORMATION PROCESSING: STS INNOVATIONS

As currently conceived, the Space Transportation System (STS), or space shuttle, involves two major information-processing innovations.

The first is the use of an on-board computer to control the countdown and launch of both the STS vehicle and its interfaces with any number of diverse payloads (and perhaps the payloads themselves) in order to eliminate expensive ground checkout equipment and operations. Some precedents exist in airplane operations, but space booster operations will have many significant differences.

The second major innovation involves radical advances in flight control: the on-board computer solving heating and structural equations in real time as inputs to a control scheme that would minimize differences between the stress history of the vehicle and that of the nominal flight plan. This would permit a considerably lighter structural design and a correspondingly lower cost-per-pound of payload in orbit.

Will the available computer hardware support these innovations? At this time it is very difficult to say because little effort has gone into analyzing the information processing alternatives available to STS; e.g., centralized versus distributed launch checkout control, software versus special-purpose hardware for signal processing, or analytic versus tabular expression of flight-control functions. However, a general feel for the magnitude of the problem can be gained from the estimates in the next figure.



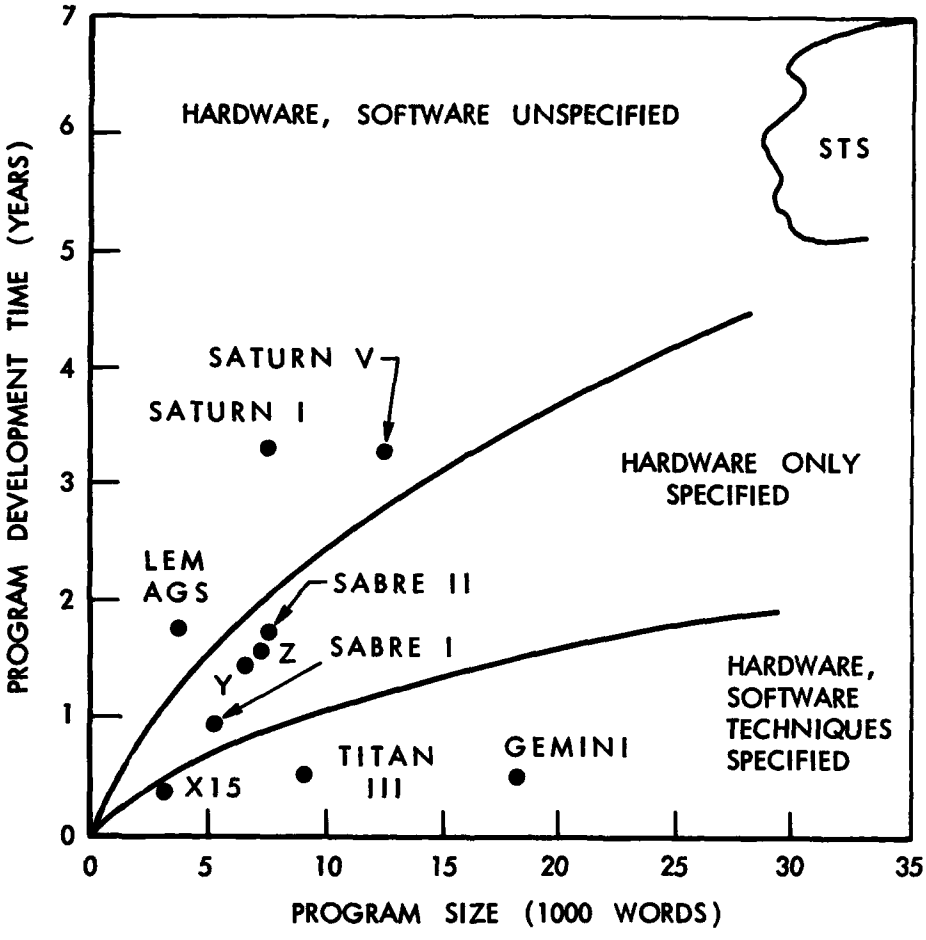
ON-BOARD COMPUTING REQUIREMENTS: MID-1970'S

STS: HARDWARE IMPLICATIONS

This figure shows some estimates by Bellcomm [8] and Autonetics [2] of on-board computing requirements for missions similar to those of STS. The Bellcomm estimates for checkout requirements depend on the number of test points monitored, and whether or not the computer performs diagnosis and trend analysis over and above monitoring. For reference, a system such as Apollo, with about 7000 test points, would require a computer memory of 250,000 words and a speed of 400,000 operations/sec to perform checkout monitoring, diagnosis, and trend analysis.

Even without considering the added demands of the innovations in flight control, the total load on the STS on-board computer will be heavier than any present-day flight computer (e.g., the IBM 4-PI) can support. Even the computers predicted for 1975 by Amdahl would be hard-pressed to handle the load. Thus STS will perforce be in the position of straining the on-board computer hardware capacity, a situation that was shown in the figures on pp. 4 and 6 to have serious consequences for software development.

A useful strategy would be for the Air Force to push the development of high-capability flight computers. If hardware manufacturers are provided incentives beyond those of the commercial flight-computer market (most of which can be serviced by relatively low-capability computers), it should be possible to improve on Amdahl's nominal estimates of future spaceborne hardware capability. Still, the complexity and indefiniteness of the STS information-processing task carries serious software implications, as seen in the next figure.



PROGRAM DEVELOPMENT TIME VERSUS SIZE AND DEGREE OF SYSTEM DEFINITION

STS: SOFTWARE IMPLICATIONS

The STS software will be quite extensive, the programs requiring considerably more than 30,000 words. Also, neither the hardware nor the software techniques for the STS on-board processing have been specified. Considering these attributes in the context of other on-board programming jobs reported in an SDC study [9], one can easily expect STS on-board program development to require five to seven years of calendar time. Thus, if one wants to fly an STS by 1976 or 1977, it is necessary to begin designing the software *now*.

Specifically, as much immediate effort is needed to resolve such questions as centralized versus distributed checkout control, and software versus special-purpose hardware for signal processing, as is needed for questions of propellant tanking, thermal protection, and the like. The Air Force must make sure that appropriate software contractors begin now to analyze STS information-processing problems in detail.

The above estimate, based on analogy and extrapolation, cannot be precise. If STS planners are not careful, however, to avoid the pitfalls of poor software engineering [10-12], or if the USAF has not developed and retained skilled software management personnel, the situation could become much worse. On the other hand, there are mitigating factors: the availability and increased feasibility of using higher-level programming languages; and the availability of floating-point arithmetic in on-board computers (some 30 percent of the Apollo programming effort was devoted to fixed-point scaling*).

---

*A percentage of this magnitude, coupled with the rising costs of software in general, indicates the necessity for stronger Air Force requirements on the inclusion of floating-point hardware in on-board computers.

- CRITICAL PROBLEM EVEN NOW

- EXTRA CRITICAL IN FUTURE
  DUE TO ADDED FLEXIBILITY,
  ADDED COMPLEXITY

SOFTWARE CERTIFICATION

SOFTWARE CERTIFICATION

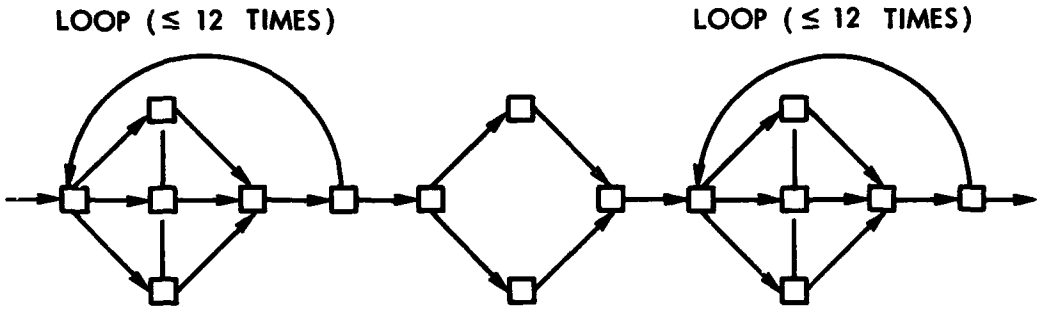
Having major projects like STS slip their schedules because of a software lag could be a serious problem. But there are software problems that could be far more serious: escalating a strategic crisis situation or degrading critical defense capabilities because of software errors.

Consider the following scenario: During a fairly tense situation, the Soviet Union sends up a large, new satellite. We decide to use a previously untried combination of sensors on an inspection satellite to take a look--an option available under the computer program that controls sensor sequencing. But under a certain unlikely combination of conditions, which was not checked out, this software option also activates a high-intensity electron beam used for warhead detection. This happens during the mission, and the electron beam kills a crew of six in the satellite.

In this or similar situations, a software error could quickly precipitate a dangerous strategic confrontation. Or, even worse, a software error could incapacitate a key component of our defense structure at a critical time.

But such things could quite easily happen, especially as software becomes more and more complex. The likelihood increases even further if we opt for real-time reprogramming, or programmer-astronauts, without a great deal more attention to real-time software certification.

Software certification is not easy. Ideally, it means checking all possible logical paths through a program; there may be a great many of these. For example, the next figure shows a rather simple program flowchart. Before looking at the accompanying text, try to estimate how many different possible paths through the flowchart exist.



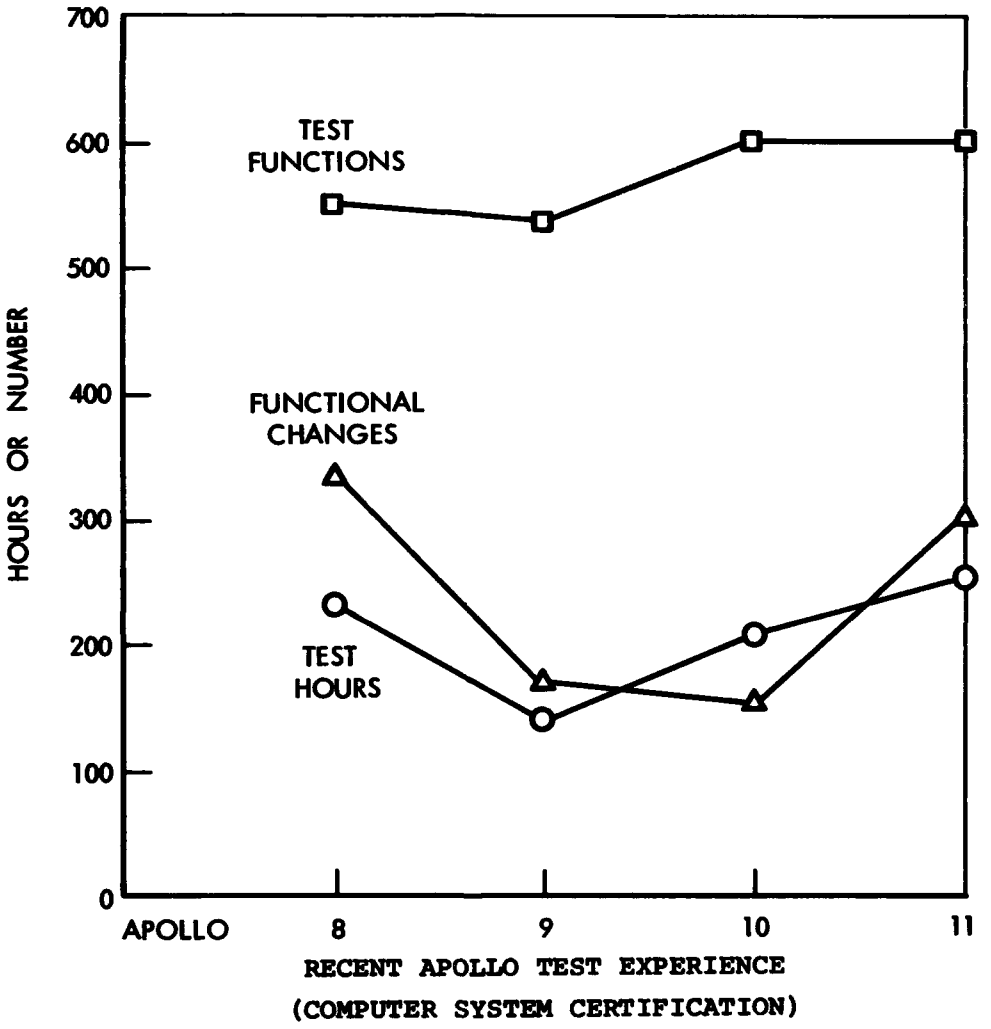
HOW MANY DIFFERENT PATHS THROUGH THIS FLOWCHART?

DIFFICULTY OF FULL SOFTWARE CERTIFICATION

Even through this simple flowchart, the number of different paths is about ten to the twentieth. If one had a computer that could check out one path per nanosecond (10^{-9} sec), and had started to check out the program at the beginning of the Christian era (1 A.D.), the job would be about half done at the present time.

So, how does one certify a complex computer program that has incredibly more possible paths than this simple example? Fortunately, almost all of the probabilitv mass in most programs goes into a relatively small number of paths that can be checked out.

But the unchecked paths still have some probability of occurring. And, even in the most thoroughly checked systems, software errors can occur. The next section discusses this problem and its effect on the Apollo program.

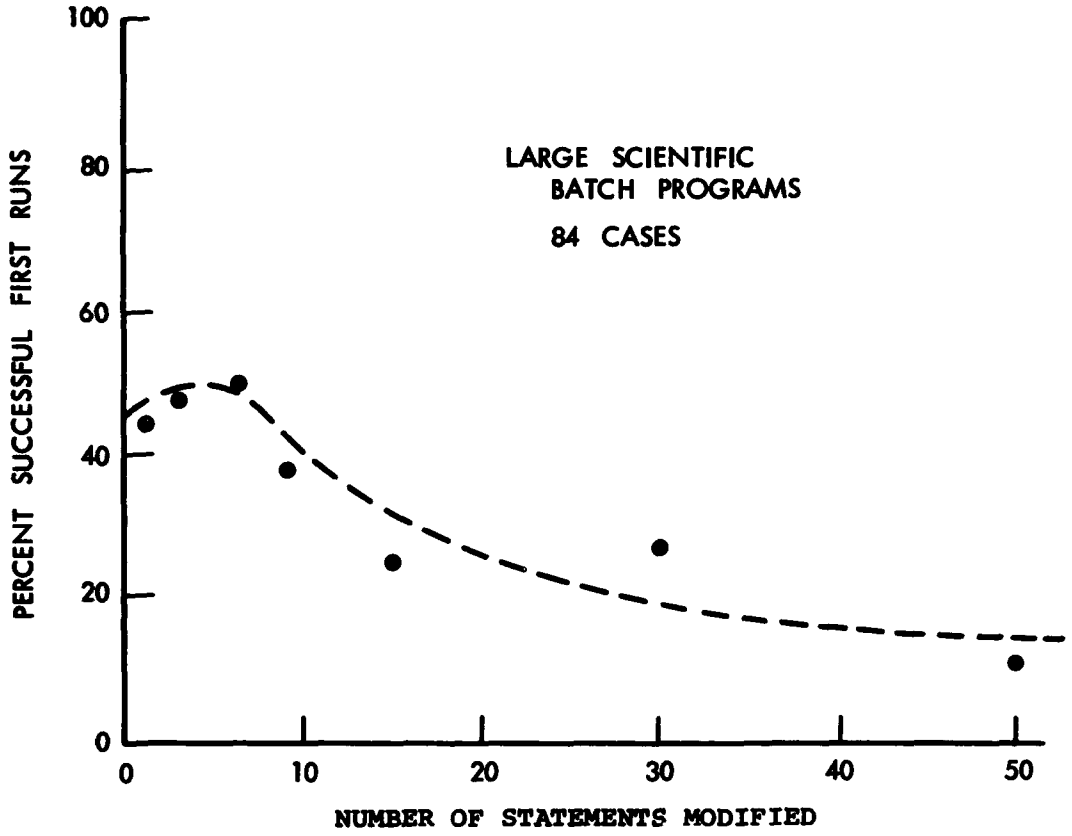

700
600
500
400
300
200
100
0
HOURS OR NUMBER
TEST
FUNCTIONS
FUNCTIONAL
CHANGES
TEST
HOURS
APOLLO
8
9
10
11
RECENT APOLLO TEST EXPERIENCE
(COMPUTER SYSTEM CERTIFICATION)

APOLLO SOFTWARE CERTIFICATION

As the figure indicates, the Apollo flights provide outstanding examples of thoroughness in testing computer systems [13]. Assuming that the testing system can certify one path through the program per millisecond, 200 hours of testing would check out 720,000,000 different paths. Yet there are many more paths, and sometimes one of them that produces a wrong result is encountered during an Apollo mission. On Apollo 8, an unforeseen sequence of astronaut actions destroyed the contents of a word in the computer's erasable memory--fortunately, not a critical error in this case. And on Apollo 11, the data flow from the rendezvous radar was not diverted during the critical lunar landing sequence, causing a computer overload that required Astronaut Armstrong to divert his attention from the process of landing the spacecraft--fortunately, again, without serious consequences.

Computer support of some military space missions will be at least as complex as that of Apollo, with two additional factors that will render the software certification problem even more difficult. First, the military systems will be far less organized about achieving a single objective; second, they will rarely have as much time to check out program modifications.

One might feel that the difficulties in checking out the Apollo computer system would disappear if there were only one or two functional changes rather than several hundred. To some extent, this is true--but consider the next figure.


100
80
60
40
20
0
PERCENT SUCCESSFUL FIRST RUNS
LARGE SCIENTIFIC
BATCH PROGRAMS
84 CASES
0
10
20
30
40
50
NUMBER OF STATEMENTS MODIFIED

CERTIFICATION AND SMALL SOFTWARE MODIFICATIONS

These observations on modifying computer programs indicate that small modifications have a better chance of working successfully than do large ones. However, even after a small modification, the chance of a successful first run is, at best, about 50 percent. In fact, there seems to be a sort of complacency factor operating that makes a successful first run less probable on modifications involving a single statement than on those involving approximately five statements--at least for this sample.*

At any rate, it appears that the problem of certifying software modifications does not disappear even for small changes. And, since there are generally many paths a program may take to and from the region of software that has been modified, the certification problem is still extremely difficult.

How can we improve the certification situation? The next figure gives some indications.

---

*The size and context of the sample preclude the results in the figure from being definitive for real-time military software; but there is no strong reason to believe that the basic trends of the data would markedly differ. But, surprisingly, no such data seems to be available on real-time military software. A USAF program to capture such data as this on software development, modification, and checkout, would not be very expensive--and certainly quite valuable from a software planning standpoint.

- SIMULATION LANGUAGES, HARDWARE
- MEASUREMENT TECHNIQUES
- INTERPRETIVE PROGRAM EXECUTION
- DEDICATED CERTIFICATION ORGANIZATION

SOFTWARE CERTIFICATION: POSSIBLE IMPROVEMENTS

SOFTWARE CERTIFICATION: POSSIBLE IMPROVEMENTS

One common way of certifying aerospace software is to simulate it, along with some part of its operating environment, on another machine--generally a more powerful ground-based computer. Usually, much of the simulation is programmed in machine language, and thus is characteristically difficult to create, modify, and understand. A language especially designed for simulating computer systems could markedly improve the situation. Several promising candidates are being developed or refined, including SIMUPOL [14], IBM's CSS [15], and ECSS, being developed by Rand for NASA/ERC [16].

Another common way of certifying an on-board computer system is to use a general-purpose computer to create stimuli for the on-board system and to evaluate its responses. However, major benefits might result from incorporating special-purpose extensions to the checkout computer; for example, parallelism for efficiently checking independent operations or carrying along multiple evaluation functions, or associative processing for identifying when the contents of key registers take on critical values.

Certifying a computer system implies being able to measure what it is doing. Yet many systems are still designed or implemented simply to maximize performance or to minimize response time, with little or no attention to facilitating measurement. Only later does the necessity for measurement arise, resulting in costly retrofits and poorer performance. Certainly, the SAGE experience [17] indicates the folly of such an oversight.

If a program is being modified on-line, it should be possible to exercise immediately the modified program by executing it interpretively with respect to several sets of nominal and critical input parameters. Thus, the on-line programmer (or his testing associate) could perform

- SIMULATION LANGUAGES, HARDWARE
- MEASUREMENT TECHNIQUES
- INTERPRETIVE PROGRAM EXECUTION
- DEDICATED CERTIFICATION ORGANIZATION

SOFTWARE CERTIFICATION: POSSIBLE IMPROVEMENTS

a good deal of checkout on a modification before incorporating it in the program. For a discussion of other certification techniques see Ref. 18.

The USAF has certainly found the advantages of a dedicated hardware testing organization sufficient to justify considerable expenditures on hardware test facilities. A dedicated software-testing organization would offer comparable advantages; these are enumerated on the next page.

- USAF HARDWARE TESTING FACILITIES
  - CENTRIFUGES
  - VIBRATORS
  - VACUUM CHAMBERS
  - ENVIRONMENT SIMULATORS
  - •
  - •
  - •
- USAF SOFTWARE TESTING FACILITIES
  - NONE

DEDICATED SOFTWARE TESTING FACILITY: ADVANTAGES

Among the advantages of a dedicated testing facility are:

Continuity. The testing staff does not disband at the end of each project. Being process-oriented rather than project-oriented, it can build on accumulated testing experience and develop special tools to make testing more thorough and efficient.

Motivation. The testing staff member is working *for* the client, not *against* his colleagues who developed the system. There is no institutional bias toward calling the system a success.

Muscle. The testing organization generally has the ear of the client, and can influence the system development specifications to make sure that testing considerations are appropriately included.

On the hardware side, the Air Force realizes these advantages with the inertial guidance test facility at Holloman AFB, Materials Lab at Wright-Patterson AFB, the environment simulators at Arnold AFB, the electronics-testing facilities at Hanscom AFB, and many others. But how many dedicated, project-independent software-testing facilities does the USAF have? None.

	ANALYSIS AND DESIGN	CODING AND AUDITING	CHECKOUT AND TEST
SAGE	39%	14%	47%
NTDS	30	20	50
GEMINI	36	17	47
SATURN V	32	24	44

COMPUTER PROGRAM DEVELOPMENT BREAKDOWN

RELATIVE IMPORTANCE OF SOFTWARE TESTING

The data presented here come from Rand and SDC studies [9,19] indicating that these examples of command-control and spaceborne programming projects are typical of such projects in general, with regard to the percentage breakdown of software development. Thus, during the 1970s the Air Force can expect to spend almost half of its software budget for military space operations on the checkout and test phases of computer-program implementation: two to three times as much as it will pay for having the programs coded.

How much will this be in dollars? This is highly uncertain; but for a rough comparison, suppose that the total computing bill for military space operations during the seventies is about the same as NASA's computing bill for manned space-flight operations during the sixties, which Nehama estimates at about two billion dollars [20]. Assuming, generously, that hardware and software costs will be about equal, we arrive at a budget of about $500,000,000 for software checkout and tests associated with military space operations in the coming decade.

Given a cost of this magnitude, it would not be difficult to justify a dedicated software-testing facility on strictly an economic basis. But this would be inappropriate. It must be justified primarily on a military preparedness basis. The nation's defense can little afford to be constrained by inflexible software. But even less can the nation afford to be drawn into strategic crises, or have its defenses weakened, because of errors in software options or modifications.

AIR FORCE SHOULD TAKE THE LEAD
IN DEVELOPING
IMPROVED TECHNIQUES AND FACILITIES

FIRST STEP: A DEFINITIVE STUDY OF
AIR FORCE CERTIFICATION PRACTICES

- CURRENT PRACTICES
- PROMISING NEW APPROACHES
- FEASIBILITY OF DEDICATED
  SOFTWARE TESTING FACILITY

SOFTWARE CERTIFICATION: A FIRST STEP

The software certification problem confronts the Air Force as seriously in other major areas--strategic and tactical command and control, and air defense--as it does in space operations. Of course, it is also a major concern in the operations of NASA, the U.S. Navy, U.S. Army, and other agencies.

But, since the greatest concern with the problem falls to the Air Force, it is at once the opportunity and the responsibility of the Air Force to provide leadership in developing improved techniques and facilities for software certification.

An important first step would be to compile a definitive study on software-certification practices in the Air Force, including:

1) Existing techniques and facilities in the Air Force;
2) Existing techniques and facilities in other agencies;
3) Promising new approaches to software certification;
4) Distribution of Air Force software research effort: does 45 percent to 50 percent go towards testing-oriented research?
5) Nature and feasibility of a dedicated Air Force facility for software testing.

- FOR SOME COMPUTING CAPABILITIES, USAF MUST ACTIVELY PUSH R&D
- ON-BOARD COMPUTERS: BUY 50% - 100% EXCESS CAPACITY TO MINIMIZE TOTAL SYSTEM COSTS
- STS: SOFTWARE DEVELOPMENT IS ALREADY ON THE CRITICAL PATH
- SOFTWARE CERTIFICATION PROBLEM UNDEREMPHASIZED; CONSIDER DEDICATED SOFTWARE TEST FACILITY

CONCLUSIONS

CONCLUSIONS

Most military space operations during the 1970s will not strain the available information-processing capabilities. But there are some operations--real-time image processing, STS on-board computing, multi-sensor data analysis, decision-oriented displays, and others--for which the Air Force will not be able to reach "on the shelf" and find tools capable of doing the job. The USAF will have to settle for reduced capabilities in these areas unless space-mission planners more thoroughly investigate their detailed information-processing requirements and couple them to the USAF R&D program in information processing.

Overall cost of an on-board computer system is minimized by procuring computer hardware with at least 50 percent to 100 percent more capacity than is absolutely necessary. The optimal excess hardware capacity will increase as the ratio of software-to-hardware costs increases during the seventies. Furthermore, it is far more risky to buy too little excess hardware capacity than too much. However, buying extra hardware does not eliminate the need for careful software-configuration control thereafter.

The proposed Space Transportation System (STS) *will* strain the available information processing capabilities. Historical data on similar software projects indicate that six or seven calendar years are probably required to design, develop, and check out the software for a project of the magnitude and complexity of STS. To make sure that software does not slip the overall schedule, STS planners must begin to design the software *now*. Specifically, the Air Force should be devoting as much contractor effort to such questions as centralized versus distributed launch checkout control and software versus hardware for signal processing as is currently being devoted to thermal protection and propellant-tanking alternatives. Furthermore, the USAF should push R&D on high-capability flight computers for STS.

- FOR SOME COMPUTING CAPABILITIES, USAF MUST ACTIVELY PUSH R&D
- ON-BOARD COMPUTERS: BUY 50% - 100% EXCESS CAPACITY TO MINIMIZE TOTAL SYSTEM COSTS
- STS: SOFTWARE DEVELOPMENT IS ALREADY ON THE CRITICAL PATH
- SOFTWARE CERTIFICATION PROBLEM UNDEREMPHASIZED; CONSIDER DEDICATED SOFTWARE TEST FACILITY

CONCLUSIONS

Preprogrammed space software in the 1970s will allow many more user options. Serious consideration is being given to "programmer-astronauts" and real-time software modification in order to provide non-preprogrammed flexibility to USAF space-mission operations. But, coupled with our inadequate capabilities to check out and certify software, such measures could have disastrous consequences in escalating strategic crises or degrading critical defense capabilities because of undetected errors in software options or modifications. The USAF could improve the situation by following its hardware-development philosophy and establishing a dedicated facility for software testing, along with operational procedures for using the facility during a major software project's development. The USAF should establish a study group to report on current certification practices and the feasibility of such a dedicated facility.

REFERENCES

1. Amdahl, G. M., "Information Processing Technology: 1970-1980," presented at USAF SAB Meeting, Vandenberg AFB, November 1969 (in process).
2. Personal communication from A. O. Williman and C. O'Donnell.
3. Sharpe, W. F., *The Economics of Computers*, The RAND Corporation, R-463-PR, August 1969. (Also, Columbia University Press, 1969.)
4. Leighton, R. B., *et al.*, "Mariner 6 and 7 Television Pictures: Preliminary Analysis," *Science*, Vol. 166, October 3, 1969, pp. 49-67.
5. Gruman, E. L., *et al.*, "The Flow of Data in Advanced Manned Missions," Bellcomm, Inc., March 17, 1967.
6. Johnson, E. G., and W. R. Corliss, *Teleoperators and Human Augmentation, an AEC-NASA Technology Survey*, NASA SP-5047, December 1967.
7. Interian, A., and D. A. Kugath, "Manipulator Technology: Ready for Space Now," *Astronautics and Aeronautics*, September 1969, pp. 24-32.
8. Baechler, D. O., and P. S. Schaenman, "Functional Requirements for the Spaceborne Computer System for a Mid-70's Space Station," *Proceedings, 1969 AAS Annual Meeting*.
9. Manus, S. D., "SDC Recommendations for Spaceborne Software Management," *Proceedings, First Spaceborne Computer Software Workshop*, 1966, pp. 345-360.
10. Naur, P., and B. Randell (ed.), *Software Engineering*, NATO Conference Report, January 1969.
11. Kay, R. H., "The Management and Organization of Large Scale Software Development Projects," *Proceedings, 1969 Spring Joint Computer Conference*, pp. 425-434.
12. Boehm, B. W., *Current Trends in Aerospace Computation and Some Implications*, The RAND Corporation, P-3484, November 1966. (Also, *Proceedings, First Spaceborne Computer Software Workshop*, 1966, pp. 12-24.)
13. Schnurbusch, J. V., "Verification of the Apollo Real-Time Mission Program," *Proceedings, AIAA Aerospace Computer Systems Conference*, September 1969.
14. Sherman, W. F., "SIMUPOI: A Language for the Production of Digital Simulators," *Proceedings, AIAA Aerospace Computer Systems Conference*, September 1969.
15. *Computer System Simulator/360: Program Description and Operations Manual*, IBM Document Y20-0130-1, 1967.

16. Neilsen, N. R., *ECSS: Extendable Computer System Simulator*, The RAND Corporation, RM-6132-NASA (in process).

17. Hosier, W. A., "Pitfalls and Safeguards in Real-Time Digital Systems with Emphasis on Programming," *IRE Transactions on Engineering Management*, Vol. EM-8, June 1961, pp. 99-115.

18. Rubey, R. J., and B. H. Dulac, "Software Tools for Certifying Operational Flight Programs," *Proceedings, National Space Navigation Meeting*, 1967, pp. 164-177.

19. Haverty, J. P., *The Role of Programming Languages in Command and Control: An Interim Report*, The RAND Corporation, RM-3292-PR, September 1962.

20. Nehama, I. D., "Information Processing," presented at USAF SAB Meeting, Vandenberg AFB, November 1969 (in process).