S-3A Avionics: Software Revolution Forerunner

L. Frank Morgan*
Lockheed-California Company, Burbank, Calif.

A review of the S-3A Project as the forerunner of software-integrated avionics systems leads the author to conclude that the guideposts of technical change are signalling a revolutionary change in the system development process itself. Advanced avionics systems are viewed as software rather than hardware based, requiring a development process which takes advantage of the greatly increased flexibilities offered for designing to a fixed cost and schedule. Examples are cited from the successful S-3A program of "software-first" influences which have a highly beneficial impact on total system design and method of implementation.

Introduction

FOR two decades the avionics industry has witnessed the significant and steady-growing influence of digital computers and associated software in the design and development of advanced aerospace systems. Thus far, computer hardware (both analog and digital) for avionics applications has established an excellent record in terms of performance, reliability, and over-all cost effectiveness. Avionics systems designers today, therefore, do not hesitate to draw liberally upon the cornucopia of digital computer hardware which now characterizes the electronics industry around the world. Curiously though, when the avionics requirement almost demands a centralized multiple processor system whose resources are flexibly allocated in real time by a common executive program, red flags of deep concern are thrown up from nearly every quarter to steer the design in some other direction. Despite the obvious and closely associated progress in avionics software development, it remains the great inhibiting factor in integrated avionics.

Somehow, the U.S. Navy's S-3A Air Vehicle for ASW rose to challenge all the red flags and to successfully integrate a complex array of electronics using a prodigious amount of central software. To those who, like the author, have seen a number of large-scale real-time software efforts begin in the same direction only to fall back after great expense to a simpler or "distributed" approach, the S-3A software development clearly represents a significant breakthrough.

To the extent that the S-3A approach and basic architecture are representative of where things are headed in integrated avionics, the author foresees a rapid and drastic change in the avionics development process for future systems. A simple projection of the growing influence of software on avionics development says that software will inevitably become pivotal for both the design and development of future systems.

In observing what others are doing and saying along parallel lines, one finds an increasing number of responsible people (and technical papers) in the field of avionics recognizing similar trends; however, it is not clear that a concensus exists as to what these trends mean.

A paper by B. List[1] of the Air Force Avionics Laboratory at Wright Field described these trends more broadly, yet succinctly, and with impressive insight. Emphasis is given here to the larger effects of what Mr. List referred to as the "Software Revolution" upon what must be an attendant revolution in our approach to avionics systems design and development.

A paper by B. Boehm[2] of the TRW Systems Group is an outstanding example of the kind of revelations which are sorely needed concerning the impact of software on new-development projects, in general, which emphasize the use of digital computer technology. Boehm's research leading to a suggested "Software-First Machine" is right on target with a principal conclusion of this author as a field practitioner.

For List and Boehm, and for the many others who have pointed to the growing influence of software and its implications, the now-culminating S-3A Avionics development project conveniently offers a real-world case in point.

S-3A Avionics—Key Design Concepts

The S-3A (Fig. 1) is a twin-turbofan aircraft designed for carrier-based antisubmarine (ASW) missions. Its empty weight is 26,300 lb, approximately 10% of which is payload electronics. Gross takeoff weight is 41,000 lb. Catapult takeoffs and arrested landings require the aircraft to withstand shocks of 6 to 8 g's with duration up to 2 sec. It is manned by a crew of four: Pilot, Copilot, Sensor Operator, and Tactical Coordinator (Fig. 2).

The avionics system in the S-3A is integrated around a general-purpose digital computer (Figs. 3 and 4). The computer gathers data on a request/acknowledge basis from the various flight control, navigation, and sensor subsystems; formats the data and presents it, both automatically and on demand, to the crew members on multipurpose programable cathode ray tube displays. Inputs from the keyset panels at each crew station are processed by the computer to provide equipment and software sequencing and mode control. The computer also maintains disposables inventories, computes and provides weapons

Fig. 1 S-3A air vehicle in flight.

Received October 23, 1973; revision received August 12, 1974.
Index category: Computer Technology and Computer Simulation Techniques.
*Senior Research and Development Engineer.

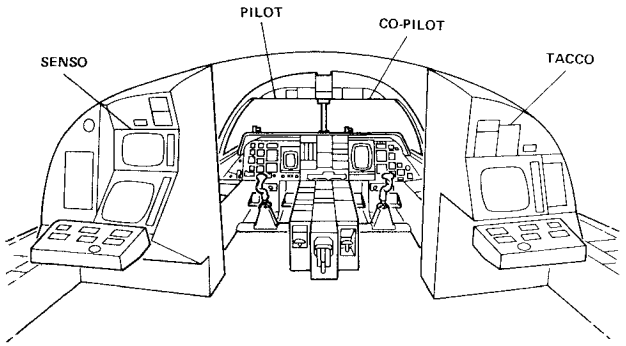

Fig. 2 Crew positions.

release commands, keeps mission records, and at crew-selected option, drives a number of dedicated flight instrument and special status panel indicators. The aircraft may be safely flown without the central computer on board, since offline controls are provided for necessary manual or automatic piloting, navigation, and communications.

Extensive built-in test equipment (BITE) is a major feature of the avionics hardware and software design. The fleet line-maintenance concept is based upon onboard checkout and fault isolation to the weapons replaceable assembly level. Base-level (carrier- and shore-based) uses the Versatile Avionics Shop Test (VAST) system for fault testing each replaceable assembly to the individual pluggable module level. Maintainability commensurate with an over-all aircraft turnaround time of 45 min has been a ruling factor in avionics design. The entire avionics system can be readiness tested in approximately 18 min using onboard software (Fig. 5) to interrogate BITE while sequencing the equipment through all functional modes.

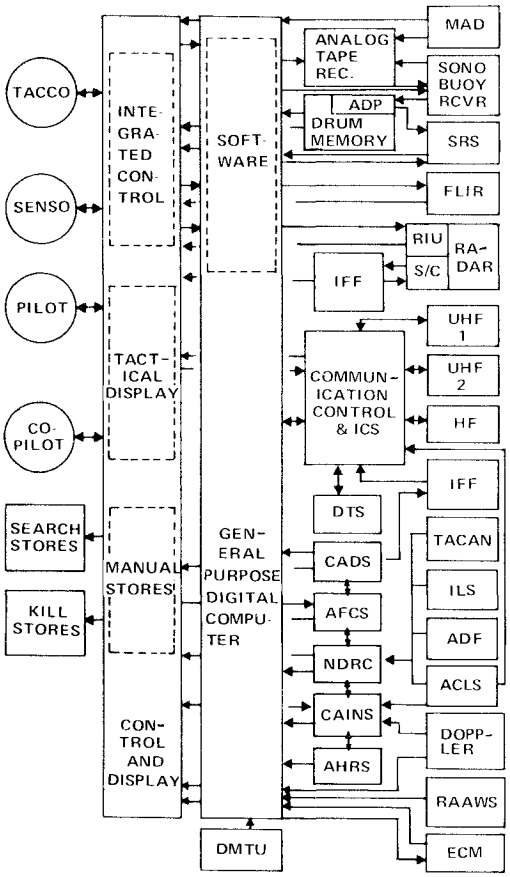

Fig. 3 Avionics system.

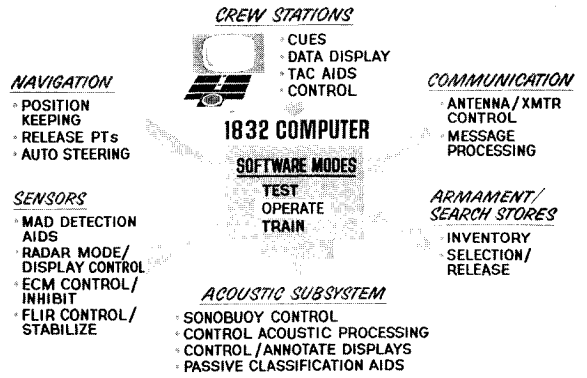

Fig. 4 Avionics software functions.

The S-3A avionics suite is comprised of several major subsystems. System design is predicated on autonomous mission operations despite the small crew size. This is made possible by automating multisensor operation and control for the collection, processing, compaction, and display of large quantities of data as required for on-the-spot decisions. The availability of advances in solid-state sensor designs and the computer-integrated system approach permitted many functions to be more effectively committed to software than to black box hardware and/or crew members as in earlier aircraft. Compactness and versatility therefore were pivotal concepts for the avionics; a fact which is readily apparent in viewing the actual cabin interior. State-of-the-art programable CRT-displays and multifunction keyset controls are prominent at each crew station, and the central computer which supports these dynamic man/machine interfaces is equally prominent in the aft-cabin avionics bay.

The central computer, as well as many other major system components, was a new-development item. The computer system is a fully dualized and cross-strapped system, as shown in Fig. 6. Operational software has been designed to take full advantage of the advanced features of the computer, including the ultrafast serial-channel capability (approximate 167,000 words/sec maximum transfer rate) to support a dynamic (drum auxiliary memory) overlay approach to the majority of real-time tasking. The operating system executive is designed for symmetrical central processor loading based on a "two highest priority tasks working at any given time" concept.

S-3A Development Approach

The development approach to the S-3A system is of special interest here for two primary reasons: 1) it is a successful current-development project, and 2) the extent of new development includes not only a more-than-usual number of advanced hardware designs, but a large-scale,

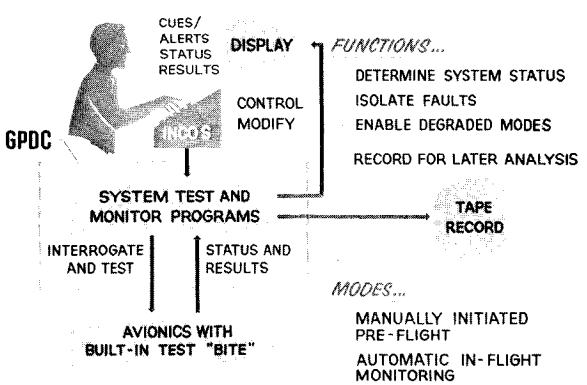

Fig. 5 On-board test operations.

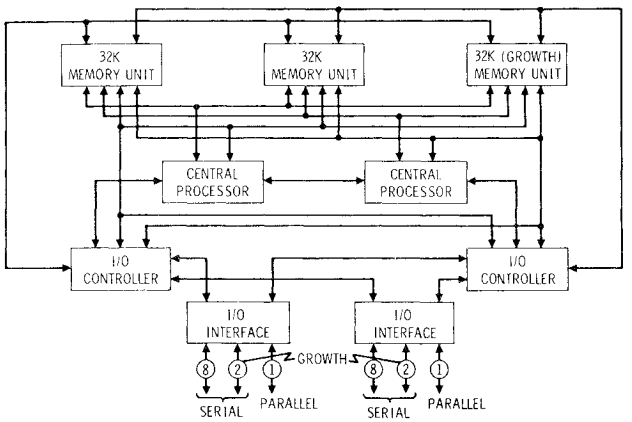

Fig. 6 Central computer.

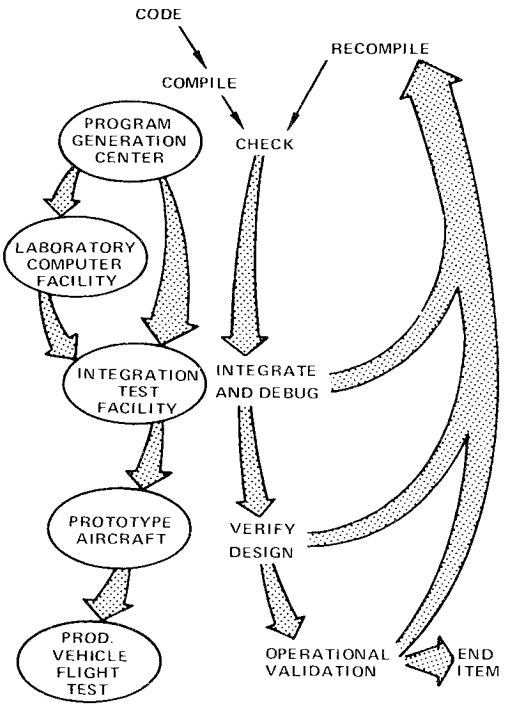

Fig. 7 Software system development.

multiprograming, dual-processor computer/software subsystem as well.

Much of the program's success is directly attributable to the continuity and learning gained from the previously developed, land-based, P-3C ASW aircraft system. It was during this prior, also highly integrated, avionics development program that the key customer and contractor personnel experienced fully the accentuated importance which must be given to the software aspects of the system. As a consequence, and because the S-3A was to be considerably more software-centered than the P-3C system, emphasis was devoted early to the acquisition of the proper software skills. This led further to the type of development planning visibility needed to place Boehm's[2] "software-first" concept into a least partial practice. A "top ten" list of techniques applied to the software (really, the system) design and development task of the program includes:

1) Allocation of substantial excess-growth margins for central-processing speed (750,000 equipment adds per second, realizable) and memory resources (98,000 words of 32 data bits and 4 parity bits).

2) Adoption of an ultrafast (6 MHz), single-twisted-pair-per-device, internal data multiplexing scheme to ensure high-response margins and reliability for an asynchronously operated, request/acknowledge method of input/output.

3) Multistaged, overlapped test and integration facilities suitable for debugging and validation of progressively combined hardware/software elements (Fig. 7). These facilities are described in Ref. 6.

4) Contingency-adaptive development planning.

5) Use of higher-order programing language (Navy's CMS-2).

6) Use of laboratory-mockup-coupled, avionics simulation tools.

7) Early identification of a "design to cost" software requirement category.

8) Detailed technical audits (with keen interest) at management level.

9) Tailored software documentation.

10) Realistic memory targeting and continuous tracking.

Early development planning included carefully analyzed, potential contingencies. Working constraints and priorities were identified against a number of specific hazards, and alternate plans were devised for effective workarounds. Several of the analyzed contingencies did arise and alternate plans effected accordingly. This planning also forced a degree of work phasing which more readily permitted 1) cost-effective adaptation to inprocess changes affecting software design, and 2) timely recognition and elimination of efforts which became nonproductive as a consequence of changing conditions.

Because of tight memory-usage and timing constraints, use of a higher-order language for operational computer programming entailed some technical risk; however, both customer and contractor felt that cost/schedule advantages outweighed the risk and adopted the use of the Navy-supplied CMS-2 language. Through concerted efforts to make the language work for S-3A, such usage proved of significant over-all advantage to the program while providing the customer with a standardized and more easily maintainable end-product.

Advantage was taken of the fixed-digital-format character of the majority of equipment interfaces by early simulation of these interfaces in a laboratory computer. Through dynamic and realistic exercising of computer programs prior to delivery of actual hardware, initial laboratory and flying test bed integration tests were considerably enhanced and software debug cycles shortened.

Through extensive early definition of software requirements, program module allocation to the level of individual crew-station functional capabilities were available to form a highly visible basis for joint customer/contractor review during the initial design period. Functions were categorized by operational need, priority, and memory usage. This permitted early identification and "holding" of design-goal-type capabilities. Thus a sizable "bucket" of design-to-cost capabilities was identified and separated. Most of these capabilities were subsequently incorporated as established time/budget/priority constraints permitted.

A system for special auditing of software detailed designs was devised and implemented. Audits of carefully selected areas by avionics general management allowed practical and experienced judgment at the total system level to be brought to bear in assuring workable, efficient software designs. Through detailed knowledge and understanding of software as well as hardware designs at management decision levels, potential interface problems and ineffectual design efforts were largely avoided.

Navy requirements included extensive, formal software

Table 1 S-3A software development scope

Software category	Application: Function or system element	Application: Machine	Supplier	Approx task magnitude: Size (32 bit words)	Approx task magnitude: Dev. span (months)
Ground support (onboard, central computer software dev. only)	Complete (CMS-2)	642B	Navy/ CSC	Government furnished	
	System Tape Generation	UYK-7	Univac	60K	12
	Lab. simulation	1230	Lockheed	60K	12
	Integration test	1832	Univac	387K	24
	Data reduction	360/91	Lockheed	70K	24
Onboard, subsystem-distributed preprocessing	Navigation	CAINS	Litton	Government furnished	
	Acoustics	1094/AYS	Sanders	23K	36
	ECM	CM 416	IBM	2K	30
Onboard central processing and control	Mission programs	1832 (AN/AYK-10)	Univac	176K	48
	Test programs			112K	
			Total Contractor Furnished	900K	48

documentation. Through the detailed planning completed during the first year of the contract, the degree of requirements granularity achieved permitted the elimination of wasteful "blanket" coverages and a more tailored application of detailed documentation requirements.

Shortly after contract go-ahead and monthly thereafter, the memory usage of each major software module was tracked against target allocations in a manner similar to that established for aircraft weight control (Fig. 8). Dynamic memory usage targets were established well below system capacity, taking into account expected development-growth factors. Since over-all software development costs strongly correlate with the number of actual memory cells required to store the programs,[5] the early establishment of tight targets and rigorous controls for this parameter acted significantly to reduce total software costs.

Cost Trade "Handle" for Software

One of the principal reasons why software tends never to be delivered on time and within cost is that it too often becomes the "infinite sponge" for absorbing difficulties encountered with hardware. Because the cost/schedule impact of in-process change to software design is typically a much more nebulous quantity[2,4] than for hardware, the former frequently loses the battle in post-go-ahead system tradeoffs. As a part of the contingency-adaptive planning mentioned above, quantitative software change impact guidelines were established shortly after go-ahead for the S-3A project as a means of avoiding this situation.

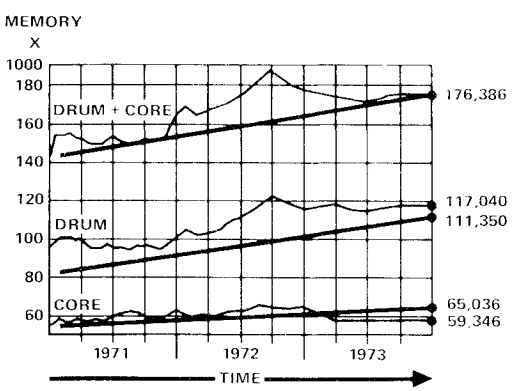

Fig. 8 S-3A mission program control and tracking curves. Note: Heavy lines represent development-growth targets.

Subsequent to the functional design freeze date (about six months after go-ahead), design change trades involving software impact estimates drew upon the following guidelines: 1) A system-capacity "use cost" of $250 per added "instruction-execution-per-second" (IPS). 2) A development cost of $100 per added instruction delivered. 3) An incompressible schedule-delay impact of one calendar-hour per added instruction.

Thus a hardware/software trade involving a potential in-process change to software calling for a new periodic task which executes every 200 msec and whose size is estimated to be 500 instructions would have a potential cost/schedule impact of:

$$\left(\frac{500}{0.200}\text{IPS} \times \$250\right) + (500 \times \$100) = \$675,000$$

$$(500 \times 1\text{ hr}) = 4\text{ weeks net schedule slip}$$

The system capacity "use cost" in this instance is absorptive, since such a periodic task would represent a significant fraction (⅓ of 1%) of the total system capacity (for the S-3A system the total software/computer subsystem "use" value was arbitrarily set at 175M, a rough approximation of the total computer subsystem hardware cost for the projected number of aircraft).

It should be remembered that such a trade comparison presumes that performance or other system design factors are not involved and that the function could be performed by the hardware with no significant loss in over-all system effectiveness.

For those who might wish to try the "use cost" per instruction per second rule of thumb in their own projects, it may be found by dividing the estimated total costs of computer subsystem hardware for the total number of systems to be built, by the effective system capacity in instruction executions per second. In most instances, just

the more concrete software development cost/schedule impact estimates based purely on the number of added instructions are sufficient to cause the majority of hardware problems to result in hardware fixes. It is only where the added number of instructions is small but the execution rate high that it is sometimes difficult to see the proper trade balance without invoking the software "use cost."

It should be noted that having quantitative tools for software vs hardware tradeoffs is no guarantee that software will not have to absorb a considerable number of hardware (or system) oversights or deficiencies. In some instances, schedule considerations alone may mitigate against hardware redesign even though the cost of software absorption of the problem may be shown to be higher. Because of its flexibility, software often becomes the best means available for purchasing precious time. In these instances, it is possible to convert time to cost at the total project level, so that cost remains the principal driver. In any event, it appears inevitable that software in these systems is, almost by definition, the area in which the principal system design "slack" must be vested and drawn upon during later stages of development. It is precisely this aspect of software-based systems that introduces the greatly increased design-to-cost potential for future avionics systems.

Magnitude of Software Task

To place these development activities in proper perspective and to indicate the over-all magnitude of the S-3A software task, it is necessary to note that the principal operational end-item software, i.e., the on-board central processing and control programs, are only the tip of the iceberg. As shown in Table 1, the total complement of contractor-furnished software is approximately 900,000 instructions. Although data which would allow precise separation of all software-related costs are not available, it is conservatively estimated that the number exceeds 30% of the avionics system total (nonrecurring costs only).

Nearly two thirds of the total software volume and one half the cost were attributable to support software or what might be termed "tooling" in hardware terminology. These numbers are conservative when compared with those of other projects[2,4] where the compiler and associated operating system development costs are included. Had the S-3A system been able to use an already existing pool of avionics simulation and test tools, these costs would have been reduced even more substantially. Had such tools been available and used to optimize hardware design prior to completion of breadboard testing if not procurement of components, the total system cost may have been different indeed. While the S-3A avionics development process followed more or less conventional lines for hardware, the "nearly first" impetus given to software at the onset kept oft-poor history from repeating.

Conclusions

The major conclusions of this paper as listed below are likely to produce a measure of controversy; as a consequence they are supported by clarifying discussion in a brief Appendix.

1) The typical command and control electronics system of the future will tend toward a more or less standard architecture, as depicted in Fig. 9.

2) Once in the customer's logistic pipeline, that portion of the typical system which includes the central computers, input/output multiplexing, standard interface units, CRT/keyset, and operating system software will become common to a number of mission-peculiar payloads.

3) In the instance of air and spaceborne systems, basic

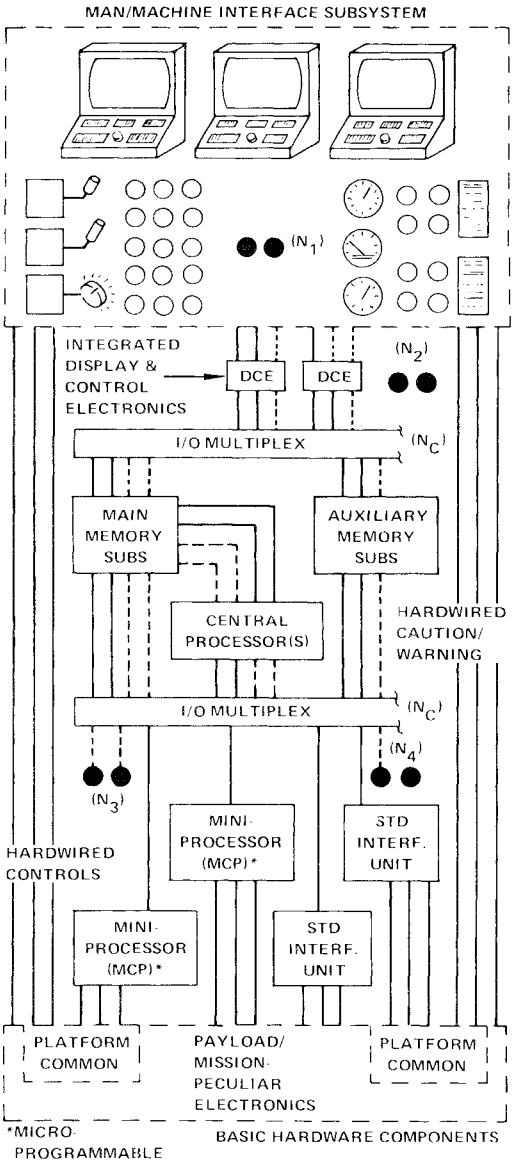

Fig. 9 Future integrated avionics—typical architecture.

flight control electronics and CNI (Communications, Navigation, and Instrumentation) will be included in this larger avionics core and become common to many widely varying payload applications (a stated major objective of the current space shuttle project).

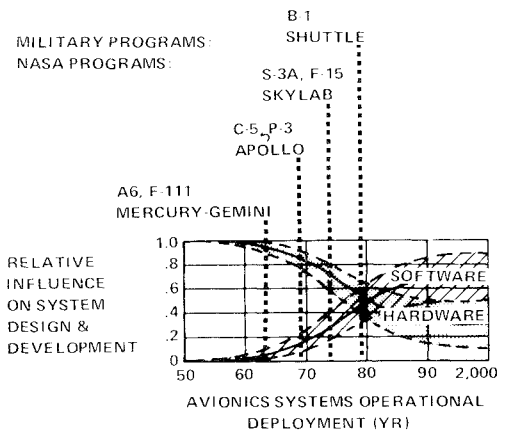

Fig. 10 Growing influence of software on avionics system design and development.

4) "Total System Simulation" will become a universal and indispensable tool for system design and development.

5) A crossover point is being approached in terms of hardware vs software influence on system design as illustrated in Fig. 10.

6) The principles of major development program management, technical approach, and key personnel skill mix will undergo substantial revision in coming years as a direct consequence of the above trends.

7) Eventually, a major new system will be successfully implemented (perhaps by the end of the century) by first developing operational software, crew procedures, and man/machine interfaces, then specifying and implementing the major hardware elements.

Appendix

Through many discussions with associates in recent years, the author is sensitive to the considerable controversy with which the conclusions of this paper are received by the majority of those who presently guide the direction of avionics development projects. The following brief arguments and clarifications are offered to those who would seek either to better understand the conclusions or to lend support to their own, similar convictions.

Conclusions 1 and 2 vs "Centralization-Fear Syndrome"

The architecture depicted in Fig. 9 corresponds to a "centralized" or "federated" computer approach, contrary to the currently popular trend[1] toward "distributed" computers. The key conceptual distinction lies in shared vs dedicated computational resources as allocated to the various subsystem processing tasks. Previously, computer resources, particularly memory, were relatively expensive and consumptive of power and weight; consequently, their employment led naturally to "integrated electronics" applications wherein a central (shared) processor was inserted to form a system output and control apex for the principal man/machine interfaces. As the momentum toward "integrated systems" increased, centralized multiprocessors were adopted (as for S-3A) because of the need for greater "apex" processing capacity and reliability. Consequently, the pure hardware functions at the subsystem level experienced serious encroachments by the central computer system. Hardware suppliers who traditionally dealt with functionally complete assemblies, including the man/machine interface, have been visibly affected. This encroachment combined with the changing economics of solid-state digital hardware has more recently produced a huge inertia in the way of integrated avionics progress. This problem has been excellently put by others[3] in terms of "doing business as usual" or "what are you doing in my area of responsibility?"

As was visible in each of the first four references, the electronic hardware supplier may now struggle more successfully to absorb that portion of the digital computer application which is pertinent to his specialty or subsystem. The present trend toward a distributed array of peripheral processors lends credence to Boehm's[2] "centrifugal tendency" but for reasons entirely different from those which he suggests in his paper. It is a phenomenon which has repeatedly surfaced in the system definition studies of recent years and one which the author refers to as the "Centralization-Fear Syndrome."

An effective expression of the syndrome lies in the frequently asked, poorly answered, question "why not avoid the complexities of centralization in view of past difficulties with 'super software'?" Aside from the obvious efficiencies to be gained, an important advantage of centralization which can be crucial to the development phase of the system is often overlooked. Centralized systems offer the opportunity for extensive, automatically sequenced system self-test capability. Though amenability to built-in (software) test capabilities is recognized for the software-computer subsystem itself, it is rarely exploited at the total system level. This fact is particularly bothersome in view of the increasing availability of digital BITE at the hardware component level. The potential for central, programable control (and data capture) of component/subsystem/system configuration and operation under simulated flight-dynamic conditions is difficult to overemphasize.

It is well established that more than half the cost/time of new developments is vested in tools and manpower necessary for system integration and verification testing. Even so, it is understood that software-heavy systems present a near-impossible verification problem because of the inherently limitless number of distinct logic-paths which are possible. A principal cause for poor software development experience with complex, centralized systems is the failure to take advantage of the equally prolific possibilities for system-level self-test and self-synthesis; for the S-3A project, this was not the case. As a consequence, test sequencing through software/hardware configurations and modes necessary for verification of large numbers of test cases was accomplished without resort to lengthy manual procedures or the involvement of large numbers of support personnel and special test equipment. Further, the momentary ability to reconfigure, disable, or "no-op" a failing equipment or process from a single control point frequently proved indispensable to meaningful test continuance.

The experience of the S-3A in this regard suggests that future avionics systems need to have such capabilities exploited to the fullest. In fact, it is possible that the absence/presence and quality of this type of central computer ability to rachet the system through a large number of normal/degraded sequence paths may hold the key to system integration success per planned budget and schedule. It is entirely possible that the required depth of verification testing simply cannot be achieved without perceptive exploitation of the dynamic control/stimulation/measurement potential afforded by a centralized system.

To return then to the instigating question of "why not avoid the complexities of centralization?" one's answer might be: "For some systems, a central software capability is justified for system integration and verification purposes alone—even though fears of complex operational software may force inclusion of redundant hard-wired controls and instrumentation for critical equipment, and, perhaps disablement of many software control capabilities once verification is complete."

Conceptually, centralization such as depicted in Fig. 9 entails a more complex software design and test verification of a nearly infinite number of functional paths, all of which converge at an apex ideally suited to automatic monitoring, test, and rapid reconfiguration. Distributed systems, on the other hand, require complex hardware designs and test verification along a large but finite number of functional paths which converge only through manual procedures and a substantial overlay of hardwired instrumentation.

If centralization requires super software, then distributed systems require super hardware and procedures. Once made error free, the one is relatively intangible but replicated cheaply and perfectly; the other is much more tangible but is expensive to replicate and without hope of perfection. Unfortunately, one tends to reduce hardware manufacture; the other tends to proliferate it.

Conclusions 3 and 4: Running the Gauntlet

This tug-of-war between "distributed" (to hardware) and "centralized" (to software) is somewhat analogous to

the emergence of the cerebral cortex over the more primitive midbrain. The software revolution in our more advanced manmade systems, hopefully, will not take so long. There is really nothing new about the kind of transition needed. Such a rapid-change gauntlet was run by the aircraft industry following WWII.

Boehm's "software-first machine" already exists and is used routinely in the design and development of aircraft structures and associated flight vector controls (i.e., in the design of electronics platforms). More importantly, perhaps, such synthetic design/development (using computers and software) is going on continuously within major aerospace companies as a consequence of routine allocation of independent research and development funds. The reasons for not having universally extended this design approach to include payload or mission-peculiar electronics systems (i.e., total system design) are clear; since, by definition, the potential variations of need, rules for design, and sources of funding for payload systems have been much less certain than those for multipurpose aerospace platforms. Besides, the software used in simulation was seldom required to fly.

To the extent that aerospace electronics systems (including flight-vector controls) become universally integrated around software-activated digital computers, software/computer subsystems tend to become part of the multipurpose aerospace platform. Then, just as a payload black box must be packaged and installed to fit within and withstand the flight dynamics of the platform, so must it meet the interface requirements of the software/computer subsystem. More precisely, the payload-peculiar functional requirements are used first to select and refine some such platform which has already been largely evolved, then actualized from available, component technology to match the platform specification envelope.

Conclusion 5: Hardware/Software Crossover Point

To recognize the growing importance of software is a necessary first step, but it is not sufficient. It must be further recognized that a crossover point is being reached in the curves of hardware and software influence on the system design and development process (Fig. 10). It is becoming increasingly necessary to select and design hardware based on software constraints, rather than the other way around. In deference to the hardware-oriented reader, it is admitted that software cannot be realized without hardware, whereas the opposite is not true. The hardware-oriented reader is reminded that this is precisely the reason why software-based systems must be approached from a software-first viewpoint. Otherwise, the tendency for hardware designs to occur in a relative vacuum may prove insuppressible and software objectives unrealizable.

The true import of the trend depicted by the curves of Fig. 10 is being felt and communicated in an ever-increasing number of avionics projects.[4] The crossover point is shown as occurring at the beginning of the next decade because of the frequency with which the author is encountering such expressed thoughts in the current literature. The unintentional, but very real, tyranny of hardware which has prevailed throughout the sixties and seventies will surely give way as systems such as the shuttle and B-1 projects reach operational reality.

Conclusions 6: Project Management Changes Needed

Granted that, for future projects, software becomes pivotal, how should the development approach differ from that of hardware-based systems? A few suggestions are listed. 1) Elevate role played by software elements of the organization. 2) Insist on collocation and close teamwork of software and hardware engineering personnel, especially at subsystem and system design levels. 3) Invest in plant facilities and IR&D appropriate to software development in proportion to its growing importance. 4) Select and encourage the development (from both software and hardware personnel resources) of "systems" engineers and managers whose knowledge and skills encompass software as well as hardware understanding.

At the technical level, software-based systems are simply approached from a "software first" viewpoint in certain critical areas as opposed to the conventional, other way around. Instead of choosing hardware concepts and then discovering what kind of a system can be synthesized from them, the idea is to use the greatly increased potential for "self synthesis" which software-based systems offer.[2]

Conclusion 7: Future Development Approach

It is suggested that integrated electronic systems of the future may first be completely synthesized using only computers, software and man-interface mockups (e.g., a facade of control room or other vehicle/platform interior configurations). Actual operational software and manual procedures would then be iteratively developed as various hardware black-box components of the system are math-modeled and optimized (computer-aided design). When the system is thus exercised through sufficient real-world scenarios to verify that a given design produces the desired end-process, specifications for realization of actual hardware would then be released for procurement.

In this idealized approach, one of the more (if not most) difficult subsystems would already be largely actualized. To use an analogy, it would be as though the "essence" of a new system were created first and then allowed, within limits, to create its own corporeal structure. Even if by the time the software were created the project were to be cancelled or totally reconstituted, the work already done would be more readily salvageable than if hardware had been done first or in parallel.

References

[1]List, B. H., "DIAS, A Major Crossroad in the Development of Avionics Systems," *Astronautics and Aeronautics*, Volume 11, No. 1, Jan. 1973, pp. 55–61.

[2]Boehm, B. W., "Software and Its Impact: A Quantitative Assessment," *Datamation*, Technical Publication Co., Barrington, Ill., May 1973, pp. 49–59.

[3]Ruth, J. C., "DIAS: The First Step," *NAECON '73 Record*, pp. 14.

[4]Trainor, W. L., "Software—From Satan to Saviour," *NAECON'73 Record*, pp. 22.

[5]Farr, L. and Zacorski, H. J., "Factors that Affect the Cost of Computer Programming: A Quantitative Analysis," TM-1447/001/00, AF Contract 19(628)-3418, Aug. 1964, System Development Corp., Santa Monica, Calif.

[6]O'Laughlin, B. D., Graham, E., and Christiansen, J., "S-3A Development Test," AIAA Paper, St. Louis, Mo., 1973.