Embedded computers—Software cost considerations

by JOHN H. MANLEY

Headquarters Air Force Systems Command
Andrews Air Force Base, D.C.

There is a special subset of the total set of electronic data processors that is growing in importance, especially to the Air Force. This class of computers possesses several unique features that cause software development to be very expensive with relation to the number of lines of code required to make them run. This paper describes some of these unique features and illustrates how they tend to drive software costs upward. Potential solutions to the high cost of software problem in this area are offered which, as the reader will find, are tentative at best since we have only recently begun to investigate in depth the intricacies of what we call “embedded computers.”

EMBEDDED COMPUTERS

Embedded computers are information processors which are integral to electromechanical systems such as automated production lines, modern aircraft, ballistic missiles, automated rapid transit systems, naval vessels, spacecraft, and the like. They are considered different than normal computers primarily in the context of how they are built and acquired for a using system. For the purposes of this paper, a computer can be considered to be of the embedded variety when it is:

1. physically incorporated into a larger system whose primary function is not data processing; and
2. integral to such a system from a design, procurement and operations viewpoint.

Furthermore, it is generally not desirable to manage the development and acquisition of an embedded computer and its associated software totally independent of other system parts. Therefore, they are normally acquired as an integrated subsystem or “configuration item” which, in our terminology, represents a well defined part of a larger system.

It is emphasized that this definition of embedded computers prevents them from being classified as “general purpose.” The author also feels that they should not be loosely classified as “special purpose” in the generally accepted meaning of this terminology. The fact that many special purpose computer systems exist that are actually unique general purpose machines, such as the STARAN and ILLIAC IV, seems to warrant the use of new terminology for those of the embedded variety. It is the combination of all the descriptive parameters taken together that makes embedded computers unique. Many of the problems with developing the software to make them function are also somewhat different as shall be discussed later.

The argument for partitioning out embedded computers for special consideration may not seem too important to many computer professionals, however it is vitally important that program managers who are responsible for their development and acquisition become more aware of the special problems that embedded computers can create for them, including their impact on the systems that contain them. Special management techniques are necessary for developing embedded computer systems that do not pertain to most general purpose and many special purpose processor systems.

Therefore, much of the following discussion is intended to support the author’s firm convictions that:

1. embedded computer systems are sufficiently unique to be placed in a class by themselves; and
2. special management techniques are required to oversee their development, testing, operation and maintenance.

DEVELOPING EMBEDDED COMPUTER SYSTEMS

The systems approach to developing any large scale system can be logically viewed as a consistent method of reducing highly complex problems into sets of relatively simple parts that can be dealt with individually. Subsystems and other “building blocks” or “black boxes” are defined and developed individually. Then, a system architect integrates the parts into a whole. This process permits a single program manager to keep the “big picture” in proper perspective, rather than forcing him to direct his attention to relatively minor aspects of the total system.

When a digital computer subsystem is treated as an individual building block, especially one that is intended to integrate the parts of a large scale system through informa-
tion transfer, many problems arise in developing its software that cause high costs. The computer system that is embedded into an aircraft in the form of a digital avionics subsystem must receive special attention from the program manager and not be treated simply as another configuration item that can be built in relative isolation from the rest of the system. The importance of this notion can be realized when we consider that an aircraft avionics subsystem includes airborne electronic sub-systems such as computers, sensors, display devices, control elements, and so forth; all of which must be integrated by means of electromechanical interface elements to assist the aircrew in managing aircraft resources.

The nature and extent of the problems in creating avionics computer subsystem software are still mostly undefined. For years the use of information generating devices in aircraft subsystems have been largely limited to analog types directed toward semi-autonomous subsystems involving communications, navigation, weapons delivery, electronic countermeasures, stores management, power plant monitoring, and so forth. Very sophisticated analog computers have been developed in the radar navigation and bombardment areas, for example, and routine use of on-board cathode-ray display devices for monitoring aircraft engine performance have been in existence on transport aircraft for well over twenty years.

Within each of these subsystem areas, however, there are interactions and the consequent need for more efficient information transfer. The logic of using digital devices throughout all subsystems is highly persuasive to satisfy the information transfer and overall integration needs. Therefore, a definite trend has developed toward the replacement of analog with digital equipment. However, the basic point is that this trend has introduced problems that were not foreseen and consequently are still in the process of being defined.

The currently accepted practice used to develop large scale automated systems such as aircraft equipped with digital avionics uses program management techniques. To plan the system development, the overall system is broken down into subsystems, sub-subsystems, and so forth, until a level of detail is arrived at that serves as a baseline specification for design engineers to begin detailed specification of individual parts. At some point in this decomposition process, the program manager's interest is fully satisfied by a detailed description of the inputs and outputs of various components without regard to their internal structure or their detailed interface requirements. The program manager's staff look at a much finer level of detail within their individual functional areas, but still not to the level required by those persons who will actually build the system.

The problem with using this normal systems approach with respect to embedded computer subsystems is that they are treated as a "black box" exactly the same as any other configuration item. In aircraft systems, the tendency is to define the size and weight of the computer hardware so that it will be compatible with the overall aircraft design, weight and balance as early as possible in the planning process. This specification is usually made well before software designers have developed sufficient detail in their area to really know how big a computer is required with respect to core and auxiliary storage. Concurrently, at the macro-subsystem level, tradeoffs are made to reduce the physical size and weight of the computer hardware to its absolute minimum to relieve pressures to develop greater aircraft lift capacities by enlarging the wings or engines. The software problem is greatly increased by these procedures whenever it is later determined that the hardware is inadequate for the software. This forces the use of lower-order basic languages, tricky programming to save space in core, and other devices that all lead to higher software costs.

Therefore, it follows that one of the basic problems involved with embedded computer systems lies in the difficulty of developing them using normal program management procedures. It should also be noted that automatic process control systems and other embedded computer systems, in addition to aircraft digital avionics, are all individually created and require their design specifications to be developed mostly from scratch. At the very least, useful documentation exists for only part of the development problem, and certainly not for the entire task since prior systems are not sufficiently the same to copy, especially in the computer software area. In addition, full complements of automation and computational equipment are generally lacking. It is difficult indeed to find an off-the-shelf computer that can do the job without major modification.

NEED FOR EXTRA CAPACITY

With any large complex automated system such as a spacecraft, early warning and control system, advanced aircraft, and so forth, that contains a controlling embedded computer, it is well understood that regular modifications will be made to add new functions and improve those which were included in the original design. As such modifications are made over time, the software must also be modified, usually requiring additional code. Since the programs in an embedded computer are generally stored in the system itself, the system computer memory can rapidly become saturated. Also, the additional functions often cause a system to become overworked to the point where the cycle time is not fast enough to keep up with the increased workload.

When the memory and speed capabilities are approached through the modification process, it is not normally possible to replace the computer with a more powerful one as can be done relatively easily in a conventional computer center. Since the hardware has been engineered into the overall system, and is therefore relatively fixed, the modifications must be forced to fit through non-standard, "tricky," programming practices which ignore the original strict control standards used during development. This results in
From the collection of the Computer History Museum (www.computerhistory.org)

generally costly and unreliable software modifications, yet another factor which drives up the life cycle cost of embedded computer systems.

The most logical solution to this problem is to add sufficient extra memory and speed capacity to the original specifications to allow for future modifications. This is an extremely difficult tradeoff for the program manager to make, especially where he is closely scrutinized for his efficiency in reducing costs to an absolute minimum. The recommendation is to educate the program manager and his technical staff in the dangers of cutting the memory and speed specifications too close during initial development. Precisely what is “too close” and how one determines just how much “extra capacity” to allow for is a subject for further research. What is important to point out, is that this is a rather unique problem for embedded computer systems that management must become familiar with to prevent excessive future software costs.

SUBSYSTEM INTEGRATION

Returning now to the avionics area, it was stated previously that many diverse aircraft functions must be coordinated by the embedded computer subsystem. For example, in our military aircraft we must integrate communications, command and control, traffic monitoring, power distribution, flight instruments, flight control systems, detection of an enemy, tracking an enemy, fire control systems, weapons management, and many others. Each of these functions are performed by analog and digital processors that operate on input data supplied by an extremely wide variety of sensors.

During the evolution from stand-alone analog devices to interacting digital systems in our aircraft development, it has been found that many relatively autonomous avionics subsystems are not fully compatible with each other. This has resulted in requirements being placed on the system architect, generally a prime contractor, to force various pieces of the system into being compatible. This practice results in an additional requirement for considerable interface software (and sometimes hardware) development. This adds to the overall complexity of the system software at higher cost, but does not in any way increase overall system capability.

Furthermore, since every newly developed aircraft is substantially different from the last, new mechanical components are acquired that perform substantially the same functions from plane to plane, but are sufficiently different to warrant different software to operate them or to use their developed output data. This is often the case because two processors that develop information from two different segments of the electromagnetic spectrum, for example, such as radar energy and infra-red energy, are usually developed by two different subcontractors. Present procedures do not generally demand investigation to see if these potentially common items could be operated in a redundant mode, with one serving as a back-up system for the other; or whether one could be eliminated by increasing the speed and capacity of the other to the point where it could handle both workloads.

Considerations such as this should be pointed out to the overall system architect by the program manager. We are currently investigating how to prescribe specific systems engineering procedures that will insure that similar items will be thoroughly investigated to see if they can be shared or used to provide redundant back-up capabilities. Such a procedure will ultimately decrease the total cost of software through a reduction in overall system complexity and diversity.

STANDARDIZATION

Since the development of embedded computer systems must be tailored to the function of the overall system they are contained in, standards for system development are taken as those that pertain to the overall system. For example, standards for building railroads would be applicable to an automated rapid transit system, aircraft standards are used for airplanes, and so forth. The difficulty here is that each of these systems use their own methods for developing subsystem software. Thus, even though there are many software development standards available, none are standard for all embedded computer system software. This is not only true across the classes of major systems, but even within classes of systems such as avionics.

Because universal standards do not exist, software developments are often characterized by poor programming practice and inadequate documentation and support software, all of which generally lead to higher than necessary life cycle costs. We are therefore looking carefully at innovations such as structured programming techniques which may be useful to partially standardize coding methodology. We are also in the process of analyzing our configuration control procedures and documentation standards to provide a basis for establishing embedded computer standards by 1975.

The preparation of reprocurement data and technical order handbooks also represents a significant portion of the development and acquisition cost of embedded computer systems. One must consider the usefulness of acquiring reprocurement data in view of its relatively high cost. Alternatives to our militarized versions of “tech orders” are also being examined since it might possibly be more cost effective to use commercial handbooks in some cases. Therefore, the goal of standardization of all data must be carefully evaluated and weighed against what it will cost in the long term.

On the other hand, the standardization of subsystems and components in our automated weapon systems appears to have a large potential for cost savings. Standardization permits reduced numbers of inventory items, such as computer programs, and larger production runs on hardware
components. Weapon systems and similar embedded computer systems should take advantage, whenever appropriate, of subsystems on hand or commercially available.

TIME-SHARING

For a fully-integrated system that utilizes an embedded computer for its control, time-sharing becomes an extremely critical problem for software designers. When time-sharing is recognized as a problem in general purpose computer shops, it is frequently because low priority jobs never seem to get into the processor for execution; or users operating remote consoles cannot get timely responses to their inputs. Identical problems in an embedded aircraft computer cannot be tolerated. For example, the central computer must continually poll various sensors to determine whether or not the electrical, hydraulic and other such subsystems are operating correctly. Simultaneously, the computer receives radar and radio sensor signals which must be processed into real time information such as current aircraft position, airspeed, groundspeed, and so forth. The question comes up during software design as to which particular functions are more important than others. Should an impeding engine failure take precedence over a radar input indicating the aircraft is flying too close to a mountain? These questions are extremely difficult to answer and generally consume more manhours to study and define than comparable problems in general purpose interactive and time-sharing processors. The net result—increased software costs.

MAINTENANCE AND CONFIGURATION CONTROL

Malfunction analysis in complex embedded computer systems presents additional unique problems which drive up the cost of software maintenance. Whenever mechanical system components exist that are controlled by computer-generated input signals, malfunctions appearing to originate in the mechanical components can also be caused by logic errors in the controlling software. For example, the inability of an automated subway train to stop precisely at the right spot alongside a loading platform might be initially diagnosed as a malfunction of the braking system, sticking relays, or some other mechanical defect. Maintenance technicians would probably take apart and inspect all suspected system components that might be causing the problem as their first order of business. Only after all mechanical possibilities are eliminated would the complex system control software be painstakingly checked for a bug. Since the preliminary detailed mechanical checks that are performed prior to the discovery of a software problem should properly be costed against the software account, the overall software cost for embedded computer systems is uniquely higher insofar as troubleshooting is concerned when compared to other types of computer systems.

In highly complex systems it is virtually impossible to completely test all possible logic paths during software development. Again and again new bugs are discovered whenever the system encounters a set of logical circumstances against which the software must function that had not been anticipated. The occurrence of such bugs in normal management information systems are aggravating since the system might be late producing a particular report, or crash as a worst case. Note carefully that when the software is being used to guide an Air Force bomber close to the ground at the speed of sound, such a bug can be disastrous.

To help reduce the occurrence of bugs in the sensitive control subsystems of major systems, it is necessary to devote considerably more attention to the original planning of the system and, in particular, to the use of a stringent configuration control program. Minor changes to the software during development to make fixes, or to accommodate changes to other system parts, must be carefully analyzed with regard to their impact on other subsystems and the system as a whole.

The real proof of software adequacy and reliability is achieved during the testing phase. Traditionally, software has been tested to determine whether or not it meets the original specifications. Even in operational tests, the major criterion is whether or not it performs to its design. In the case of testing large scale embedded computer systems, it is also necessary to do the following:

1. Perform detailed diagnostic analyses of mechanical components that have failed to determine whether or not software can be a contributing cause.
2. Develop and install automatic test equipment that records the operation of the overall system over time. When any new malfunction occurs, test data will then be available to aid in the diagnosis.
3. If at all feasible, provide one or more systems that are highly instrumented and are capable of duplicating malfunctions that only occur during actual operation and cannot otherwise be observed and analyzed. Such instrumented systems can provide the complex interface data between subsystems that is critical to accurate and timely troubleshooting.

Another major problem with embedded computer software with regard to configuration control is the requirement that it be relatively “tamper-proof.” The software that integrates and controls various subsystems in a complex weapon system, for example, must not be made too easily accessible to deter unauthorized personnel from tampering with it. For this reason, the most important programs are generally protected by such methods as storing them in read-only memories. This specialized requirement tends to make software maintenance relatively more costly than in general purpose computer systems since the latter systems' programs can be readily accessed in the computer room. It should also be pointed out here that protection against tampering would probably not be a critical necessity if it
were not for the complexity of the integration problem discussed previously.

PROGRAMMER SKILL REQUIREMENTS

Finally, there is one more potential problem area in software development and implementation that may be somewhat unique to embedded computer systems. Although it probably cannot be proved as a fact, it would seem that the basic *modus operandi* of software developers and implementers are a direct result of a specialist's viewpoint and not that of a true systems engineer. Programmers concentrate primarily on getting subroutines to run, making subroutines run together in a program, and getting programs to run precisely in the manner as dictated in the original specifications. The exact methods used to get the programs running to specification (which are frequently more art than science) are centered on the program logic itself and not necessarily on what each fix, patch, or other coding change might do to the overall system.

Although this is speculative reasoning, it might partially explain why many of our key personnel engaged in developing and implementing embedded computer systems keep asking for a "special breed" of programmer, one who is a hybrid between an electrical design engineer and a computer programmer. The experts in this field claim that the normal analyst/programmer does not have the proper outlook on "systems" that he should have. Consequently, the extra awareness must be learned on the job. This ultimately requires additional training expense and, hence, increases the overall cost of special embedded computer system software.

Therefore, we are also beginning to investigate precisely what special skills are required for personnel engaged in developing embedded computer systems. We hope to have some results in this area in the near future, at the very least a determination of whether or not this is a real problem.

SUMMARY

The arguments presented above have attempted to justify the requirement for special management treatment of a class of computers that are "embedded" in large, complex, electromechanical systems. This subset of electronic data processors are more difficult to program, and hence more costly to program, than most other types of computers due to their interactions with other parts of a larger system. More attention to planning their development and stricter configuration control of all modifications and changes are mandatory to prevent adverse side effects in the operation of the overall system due to software bugs. These factors, in addition to the others discussed, cause the overall cost of embedded computer system software to be driven to a level that is proportionately higher than comparable software used in normal data processing applications. For this reason, we are intensively investigating this special subset of computer systems to determine how to improve our acquisition management methods to hold their life cycle costs to a minimum.