Appendix F

Software Architecture

Content

F1.0 Tab 1: Importance of Architecture in DoD Software

Barry M. Horowitz, Ph.D.
ESC-TR-94-208, September 1994

F1.1 Preface

DoD's automated systems are likely to face more varied military threats than in the past that will require the ability to make system changes rapidly. In addition, defense budgets are likely to continue to decrease. It is important then, for both mission effectiveness and cost savings, that these systems be built with as much flexibility as possible to incorporate new capabilities and new technology. This paper proposes that an increased focus on digital system architecture can markedly improve system flexibility as well as yield significant cost savings. The author, Dr. Barry M. Horowitz, is President and Chief Executive Officer of The MITRE Corporation.

F1.1.1 Acknowledgements

Many people at MITRE contributed data and ideas to this document. Special thanks are expressed to Judith A. Clapp, Dr. Richard J. Sylvester, and Gerard R. Lacroix.

F1.2 Introduction: A New Direction for DoD Software Acquisition

Our world is changing. The military threat to the United States posed by the Soviet Union for nearly fifty years is diminished, but there are new threats from rapidly evolving Third World countries that require rapid changes to military systems. Crises such as the recent events in the Persian Gulf highlight the need for flexible systems that can be changed quickly to meet the military's unanticipated challenges. In addition, the defense budget continues to be reduced — the government has less money to spend on systems.

The answer to this dual challenge — to make systems more flexible and to reduce the cost of defense systems — lies in the design of the digital system architecture, which includes the composition of hardware and software components, the structure that interconnects them, and the rules by which they interact. All too often, both government and industry focus too narrowly on achieving the initial requirements for systems and give little thought to being able to adjust to what the system may be required to do five or ten years later, or to what happens as hardware may no longer be supportable or advanced technology may become available for incorporation into the system.

Architecture design is the key to achieving the cost savings and operational flexibility inherent in digital systems. If the system is properly structured, then hardware components can be added or upgraded without expensive changes to the rest of the system. A good architecture allows a system designed to counter one threat to counter a different threat through localized modifications to the software that change the functional capability of the system or allow it to interoperate with other systems. What is more, under the right circumstances, these changes can be made very quickly.

F1.3 DoD Software: More Important — and More Expensive

F1.3.1 The Value of Software

Software provides modern defense systems with a flexibility that cannot be achieved in any other reasonable way. Operation Desert Storm provides several excellent examples.

Patriot is an Army corps-level missile system primarily designed to counter aircraft. Given the inherent capability of the missile itself, the designers gave some thought to employing it to shoot down incoming enemy missiles. The Scud attacks during the war, however, focused everyone's attention on this threat with much more urgency. Patriot's designers developed a new software package that increased the Patriot's effectiveness to counter the Scud threat. When radar tracks began to show that the Scuds were breaking up on re-entry, the designers further tuned the package to recognize and attack the Scud warhead, and not the debris that accompanied it. Without the modified software, Patriot would have been less effective. Yet the designers were able to implement this capability quickly and at a surprisingly low cost. No new missiles or radars were required. The software improvements could go to the war region in a briefcase.

Another example of this flexibility also involves Patriot, though at the much higher level of command, control, communications, and intelligence (C3I). To improve Patriot's ability to react to the Scud attacks, which proceeded at much higher speeds than the targets normally confronting Patriot, US space satellites were redirected to watch for Scud launches. When a launch was detected, the satellite relayed the targeting cues over a satellite link to the appropriate Patriot battery, leading to a successful interception. Minor software modifications permitted a network to be set up.

There are other, less dramatic examples of the value of software's inherent flexibility that came out of Desert Storm. Navy attack aircraft had been set up for years to attack Soviet targets, either at sea or on land. A cassette data tape provided the attack computers with the information they needed to launch their stunningly accurate attacks on targets that had only recently been identified.

Software was also the key to the effectiveness of Air Force jamming aircraft. Programmed for operations against Soviet-bloc radars, the jammers were faced with a mixture of Soviet, French, British, and Italian equipment. Software changes enabled the equipment to perform its task against this new threat far more quickly — and less expensively — than could have been done otherwise.

Precisely because of its flexibility, the DoD is buying more software in its systems and implementing functions in software that had previously been performed in hardware. Figure F-1 shows this trend in a number of systems. For example, the latest version of the Cobra Dane radar system uses more software than did the original release, and the new Milstar terminal uses more software to perform more functions than did its predecessor, AFSATCOM. Desert Storm demonstrated that the flexibility software offers us is real and of great value to the military. It will become more so if we continue to have crisis scenarios that are a lot harder to predict and cause us to apply our systems in unplanned ways.

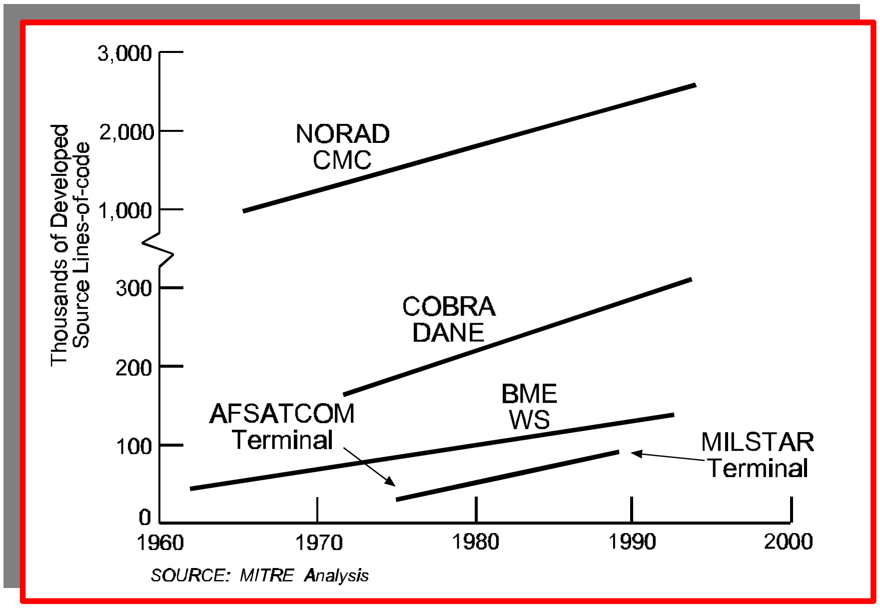

Figure F-1. Growth of C3 Software Size

Looking at the distribution of software maintenance activities is itself illuminating. About two-thirds of the software maintenance effort for a system is typically spent on modifying the original system to provide new capabilities and to add new technology, at least twice the effort spent on making repairs. Figure F-3 confirms this data for an Army command and control system. Taking these two sets of data together suggests that about 45% of the effort spent on software is used to change the system after it has been delivered.

F1.3.2 The Cost of Software

Since the DoD has been buying more and more software, its total expenditure on software has been increasing, and software is expensive. With shrinking military budgets, we have to find ways to use more software and yet reduce its cost. Typically, two-thirds of what is spent on software is believed to be spent after the system becomes operational, during the maintenance phase, as illustrated in Figure F-2.

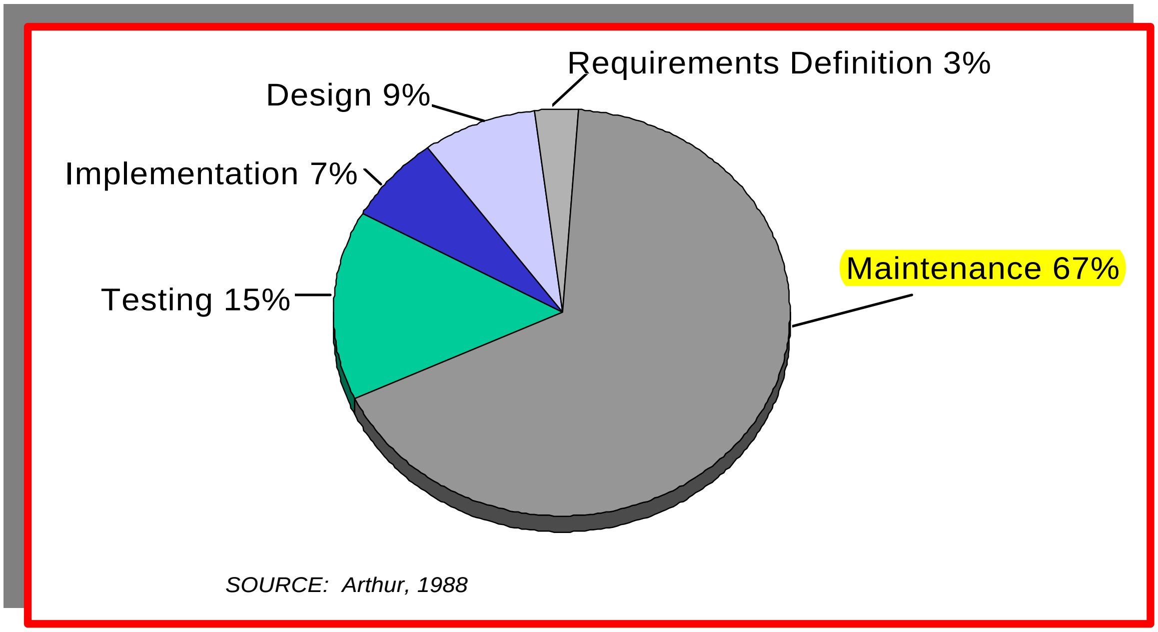

Figure F-2. Software Life Cycle Cost Distribution

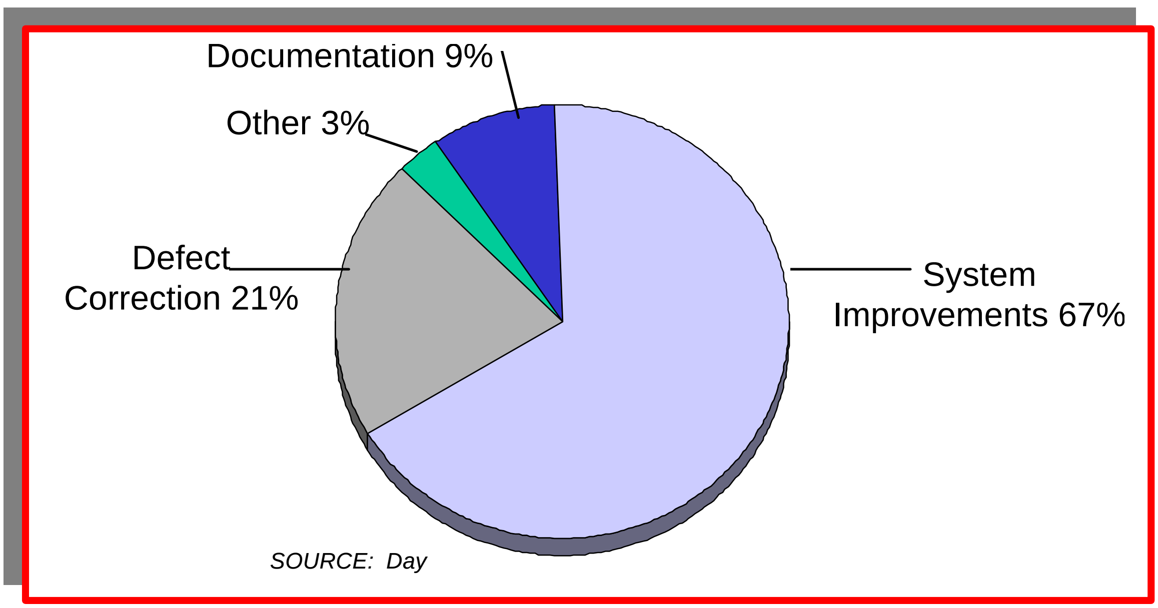

Figure F-3. Software Maintenance Activities

Experience also shows that we often spend part of the system development effort making changes in response to changing or better understood requirements. We probably spend more than 50% of our software effort to change the capabilities of a system over its developmental and operational lifetime. If we can design software systems to take only half as much effort to modify, we can reduce the life cycle cost of the entire software system by 25%. When applied to the total amount the DoD spends on software, this improvement can yield enormous cost savings. While it is difficult to determine accurately how much the DoD spends on software, MITRE staff made a rough analysis that indicates the total amount to be approximately $30 billion per year (see Figure F-4). If we can in fact reduce the life cycle cost of software by 25%, the total savings will range between five and eight billion dollars every year.

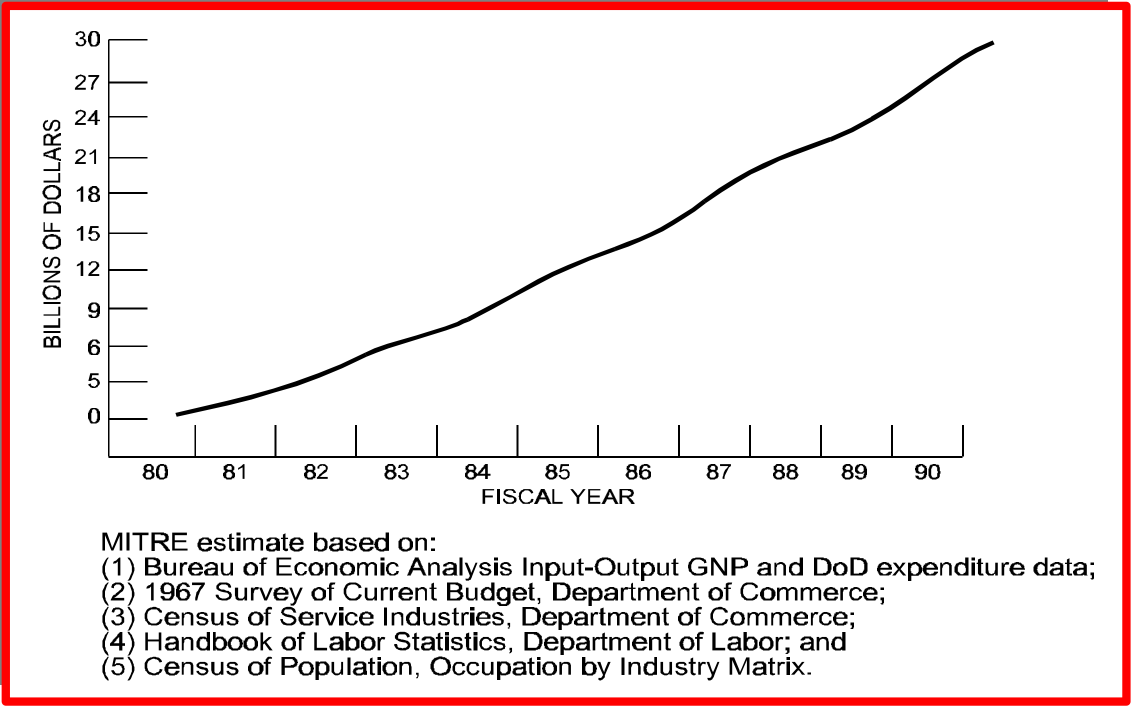

Figure F-4. DoD Software Expenditures

The example in Figure F-5 illustrates how these savings might be possible. Three thousand lines of new code were required to be added to a system of 50,000 lines. When the changes were made, the cost, time, and number of defects found in the delivered system were measured. Then, the structure of the software was improved, and the changes were again made. It cost only half as much to modify the structured software and it took less than half the time. As an added benefit, there were about one-eighth the number of errors in the structured software.

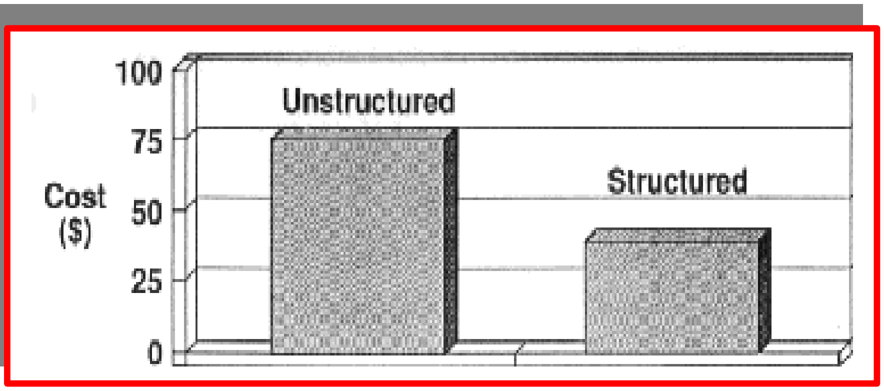

Figure F-5A. Structure Versus Cost to Change

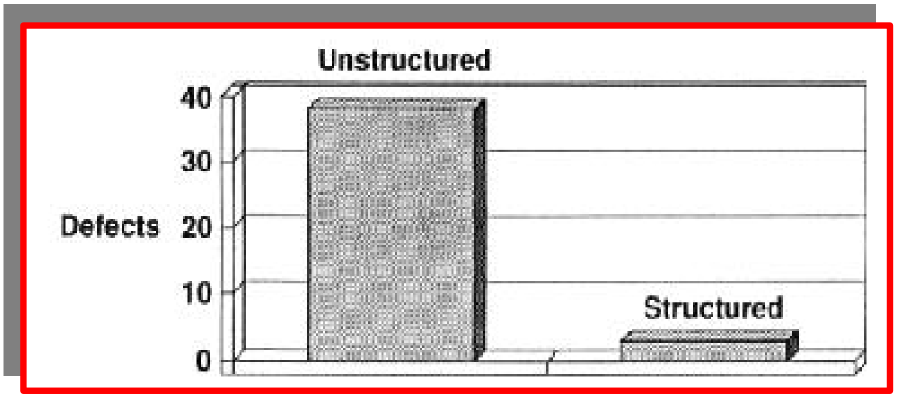

Figure F-5B. Structure Versus Defects

Another indication of increases in productivity that may accrue from well-structured software is shown in Figure F-6. The points on the graph represent software size and productivity for development of some systems programmed in Ada. One of those systems, the Command Center Processing and Display System Replacement (CCPDS-R), was developed with special attention to designing a system architecture and tools to facilitate its modification. The original system was then significantly modified to produce two new versions. Productivity data for the two modified versions of the system are shown in boxes in Figure F-6. The high overall productivity is due in part to the architecture that accommodated these changes and in part to tools that facilitated making changes. Further benefits were realized because the architecture made general system services more accessible and consequently the application modifications were smaller than they might otherwise have been. The productivity data were adjusted for the reused and tool-generated software. This is even more impressive when the usual negative relationship between productivity and system size is taken into account.

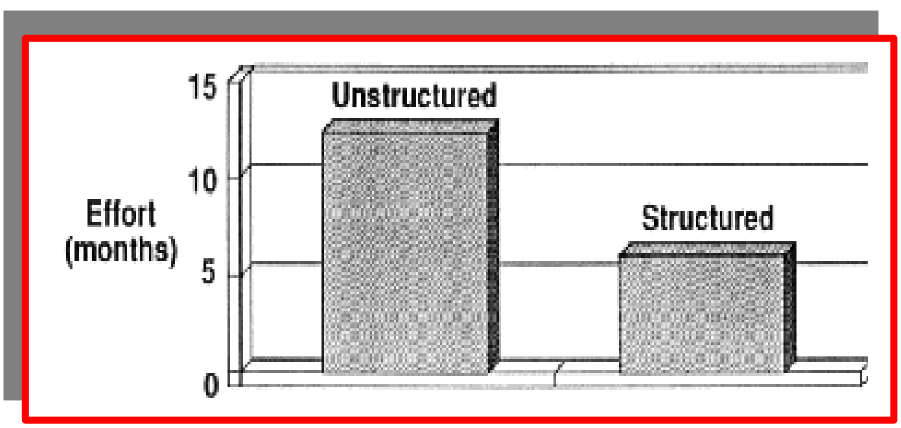

Figure F-5C. Structure Versus Time to Change

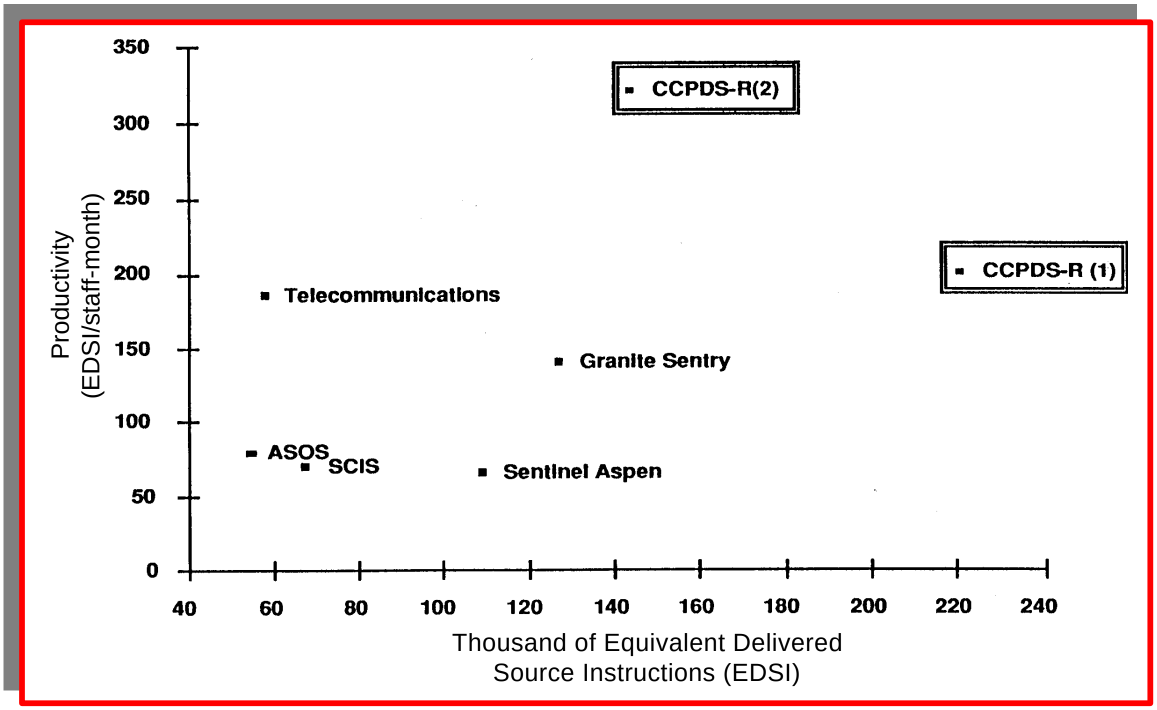

Figure F-6. Ada Development Versus Modification

While important, the dollar cost of making changes to the system is only one concern; time is another. Operation Desert Storm provided many examples of how well the flexibility of software served the allied cause, but there were also cases where we were not able to exploit software as we would have liked. Requests for changes to systems were made early in the campaign, and estimates were provided that said it would take 18 months to make the desired changes. This was obviously unacceptable, and the military found it hard to understand why it should take so long, given that software is supposed to offer great flexibility.

Software does provide flexibility, but it must be designed from the start with an architecture that allows it to do so. Furthermore, everyone concerned must preserve the integrity of the architecture; otherwise, flexibility can be lost through the process of change. As an example, Figures F-7A and F-7B are plots of the time it took to create each release of an IBM operating system and the number of modules affected in each release. The graphs show a progression; that is, it took longer and longer to modify the system as the system grew older. This was due to the growing complexity of the system — more and more modules had to be changed for each new release. The software structure degenerated, which made it more difficult to determine which modules had to be repaired. In addition, the pattern of regression testing had to be more extensive since so many parts of the system had been affected by the modifications. This complexity and uncertainty translates into more time and money, and this process begins a vicious circle — modifying the system makes the next modification even more difficult, time-consuming, and expensive.

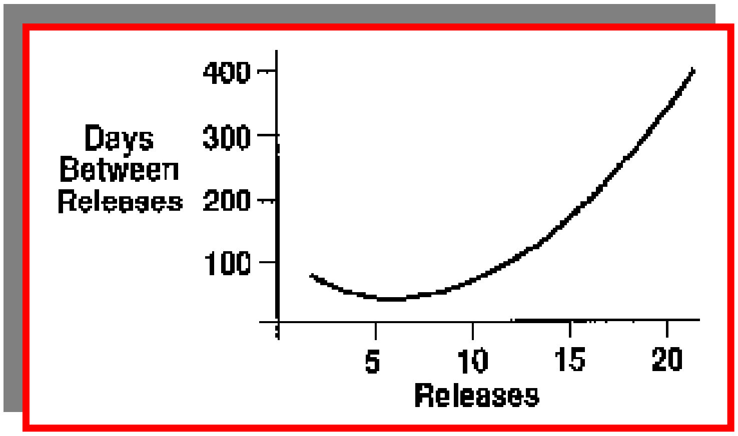

Figure F-7A. Release Interval Versus System Age

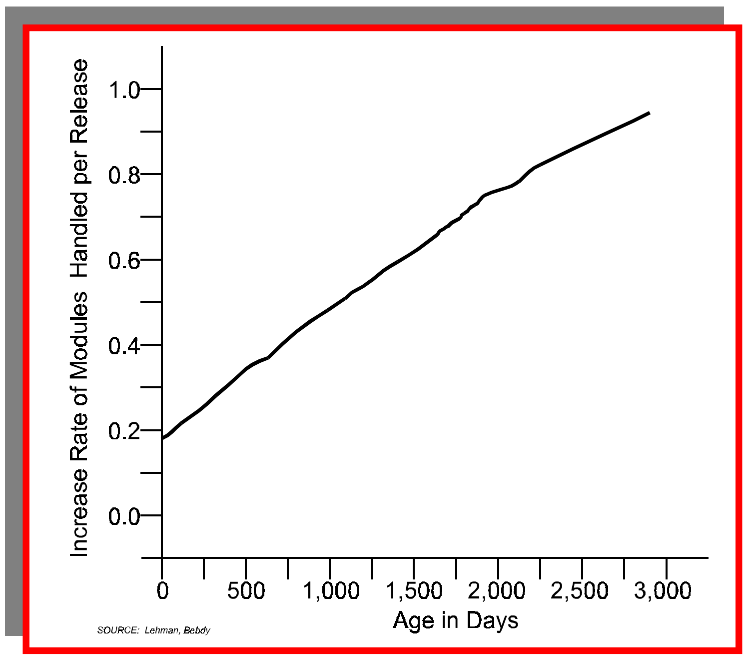

Figure F-7B. Increasing Complexity with Age

F1.3.3 Architecture: The Invisible Component

The DoD does not usually buy architectures — it buys systems that meet explicit functional and performance requirements specified by the user or the acquisition agent. In most cases, the DoD does not ask for an architecture to be delivered; it should therefore come as no surprise that very few architectures are delivered. This is not to say that the DoD does not receive a system architecture. Every system has some form of architecture, but the architecture the DoD receives may be quite convoluted and inflexible by the time the system moves from concept to fieldable implementation. The DoD does not specifically buy an architecture because there are no explicit specifications for its characteristics, no formal tests of its capabilities, and no formal control of its structure to prevent arbitrary changes once it has been defined. This is one reason why architecture is fundamentally invisible — operational users are not often aware that an architecture is even present if it does not directly affect the functional capabilities they are using.

Yet architecture is the main determinant of a system's characteristics. The efficiency of the system, and thus its performance, depend on how the architecture handles resource utilization; architecture determines how the system sustains operations when parts of the system fail. The architecture also determines how maintainable the system is; that is, (1) how much effort is required to find and fix errors; (2) how easy it is to add new capabilities through software; and (3) how much is required to move the software to different computer hardware. Although they may be invisible to the user, these characteristics, which are all determined by architecture, are very visible to developers and maintainers who must modify and add to the operational capabilities of the system. If the DoD wants to buy architectures, it will first have to know how to ask for them, specify them, test them, demonstrate them, and prevent them from degenerating; in short it will have to perform all the operations that it performs now when buying other products.

In addition, DoD must perform a new task that is currently not part of its acquisition strategy — maintain explicit control of the architecture for the life of the system. One way of accomplishing this is to add architecture to the other aspects of the system that are controlled by the configuration management system. Since the maintenance phase contains a large fraction of the system's software costs, the ultimate maintainer of the system — and thus, the government — must eventually assume control of the architecture. This will require a significant change in the way the government currently views architecture and its importance.

F1.3.4 Architecture: A Definition

There is no single, commonly accepted definition of a digital system architecture. In the broadest sense, architecture is a system or style of building having certain characteristics of structure. When applied to digital computer systems, architecture includes the hardware and software components, their interfaces, and the execution concept that underlies system processing.

The simplest level of system architecture defines how the hardware and software that make up the system are partitioned into components, and how software components are assigned to hardware components. Figure F-8 is an oversimplified example (only primary functions are shown) of a fighter aircraft's federated hardware and software structure, which consists of separate computers networked on a standard bus with individual software functions assigned to the individual computers. At this level, the defense industry generally does a fairly thorough job of understanding architecture, mainly because developers need to understand how much hardware of which types they need to buy.

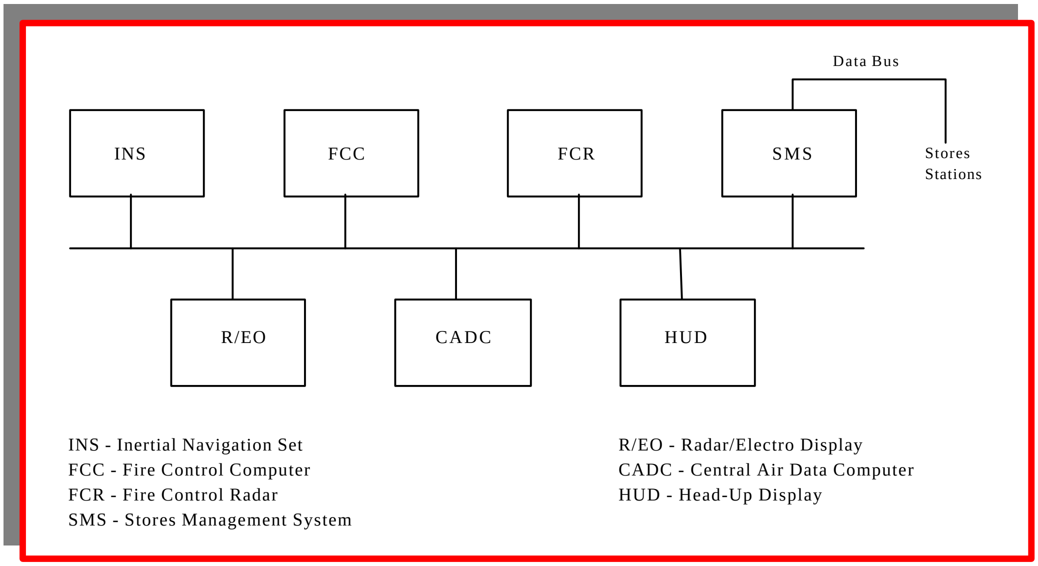

Figure F-8. Hardware and Software Structure

Figure F-9 is another view of the digital system architecture for the same aircraft, showing both the application software in the previous figure and the software that performs system-wide functions. The functions can be described as grouped into layers; in this view, software in any layer may utilize software only in its own layer or the layer below it. The computers in the lowest layer represent the segregation of hardware from software to increase their independence and to enhance software portability. This is an example of the first part of the definition of architecture — the arrangement of hardware and software components (namely, the structure).

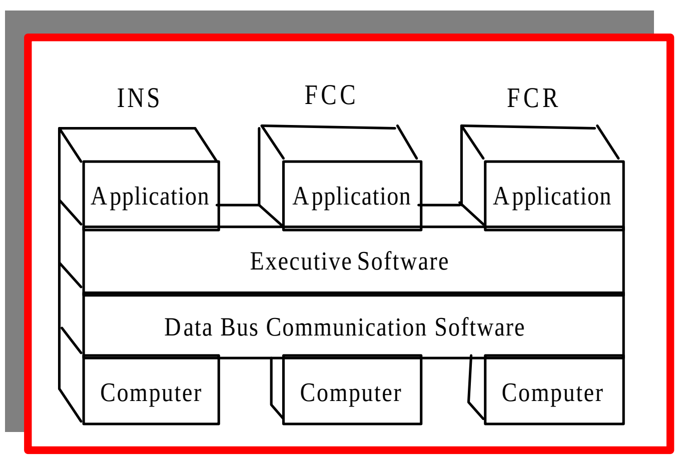

Figure F-9. Digital System Architecture

The second element in the definition deals with the interfaces among key elements — for example, data communications according to a standard protocol **(MIL-STD-1553)**. All computer-to-computer messages in the aircraft's avionics architecture must use this protocol; hence, adding new computers and new functions to the system is relatively simple (from a communications perspective) as long as the data bus has the needed capacity.

The third element in the definition of architecture is the execution concept. In the sample avionics system previously shown, this concept is based on the cyclic execution of each function, precisely timed to repeat the computation on a planned schedule. Taking structure, interfaces, and execution concept together produces one definition of architecture. Of course, different vendors interpret the software part of the architecture in different ways.

Figure F-10 illustrates an IBM view of software architecture. In many cases, commercial companies can provide off-the-shelf components for the general system capabilities of DoD systems; in addition, groups of commercial hardware and software vendors are defining standard interfaces among layers and components. These open system architectures may provide the flexibility necessary to integrate, with a minimum of effort and system disruption, new hardware and software components with improved capability or maintainability. For example, the International Standards Organization (ISO) Open Systems Interconnection reference model defines the functions of each layer and the protocols for peer-level layers of a communications interface. Standards of this type permit the upgrading of elements of the system at particular layers without requiring the alteration of elements at other layers.

Figure F-10 also illustrates the concept of service layers in the part of the architecture that is developed uniquely for one class of application (such as command centers or communications systems). These or other services must include error detection and recovery, interprocess communications, scheduling, and synchronization of processes. At this level of architecture, we must rely on the applications designers for standards within their design, as well as the quality control procedures to assure adherence to their standards.

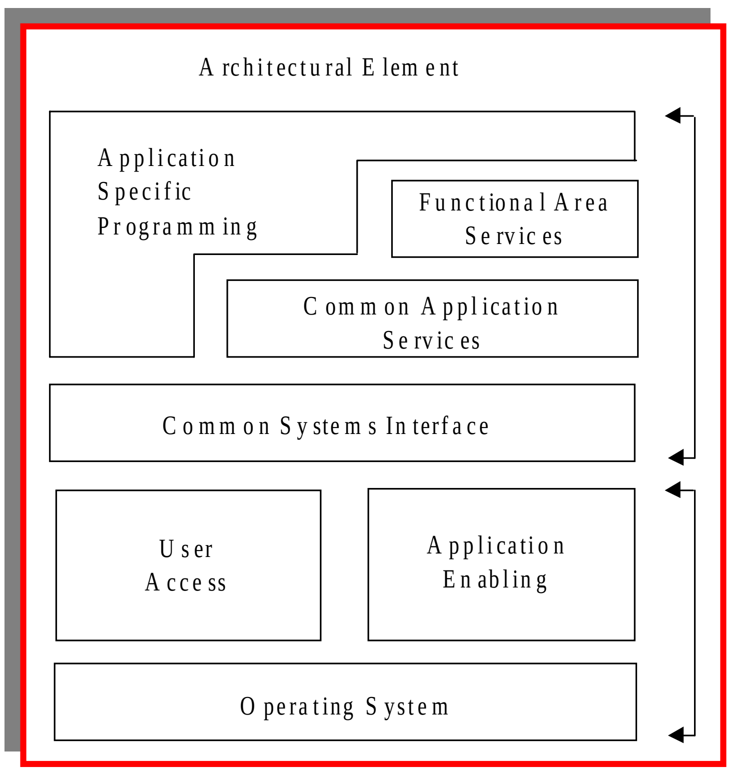

Figure F-10. IBM View of Software Architecture

F1.3.5 Architecture: Ramifications

The lack of a good architecture has a serious bearing on the cost, effectiveness, and availability of DoD systems. For many applications where high reliability and availability are necessary, the architectural concepts must incorporate failure management as well as other mission requirements. Trouble follows when this is not part of the initial architectural design.

Error handling is a critical component of any system, since errors will inevitably occur. Most systems have software to detect errors and to recover from an error when it is detected (for example, when a numerical value goes beyond expected bounds or when an operator pushes the wrong button). Given the critical nature of most DoD systems, it is crucial to keep the system in operation when errors occur. When we leave it to each programmer who has developed a part of a system to determine how to handle errors, the result is an unintegrated set of sometimes widely varying procedures that are often completely incompatible and even dangerous.

Recently, MITRE scanned the software for a large, safety-critical command and control system, and identified over 200 instances in the code that handled errors incorrectly. In many cases, the system detected the errors and then ignored them, or passed them to another part of the system that could not handle them. What was missing was a consistent, coherent, system-wide error-handling strategy, a critical attribute of architecture. Furthermore, there was no method of ensuring that individual programmers adhered to the failure management standards that should have been established with the architecture.

Data flow diagrams can show the execution concept of the architecture of a system (see Figure F-11). In this view, the sequence of processing, and which hardware and software components are involved as specific data moves through a system, are apparent. This end-to-end view of the system's treatment of an external input is called a string; in actuality, there are many levels of detail that can be represented by a hierarchy of data flows for the same string. A string is useful for assuring users that the system will perform the right functions on their data; it is also useful for estimating and controlling the time the system will take to respond to an input. What complicates the design of an architecture to meet response times is the large number of such strings that may be awaiting execution at the same time (as when many sensor reports are received or must be transmitted) and the contention over which string will use shared resources such as computers and communications lines.

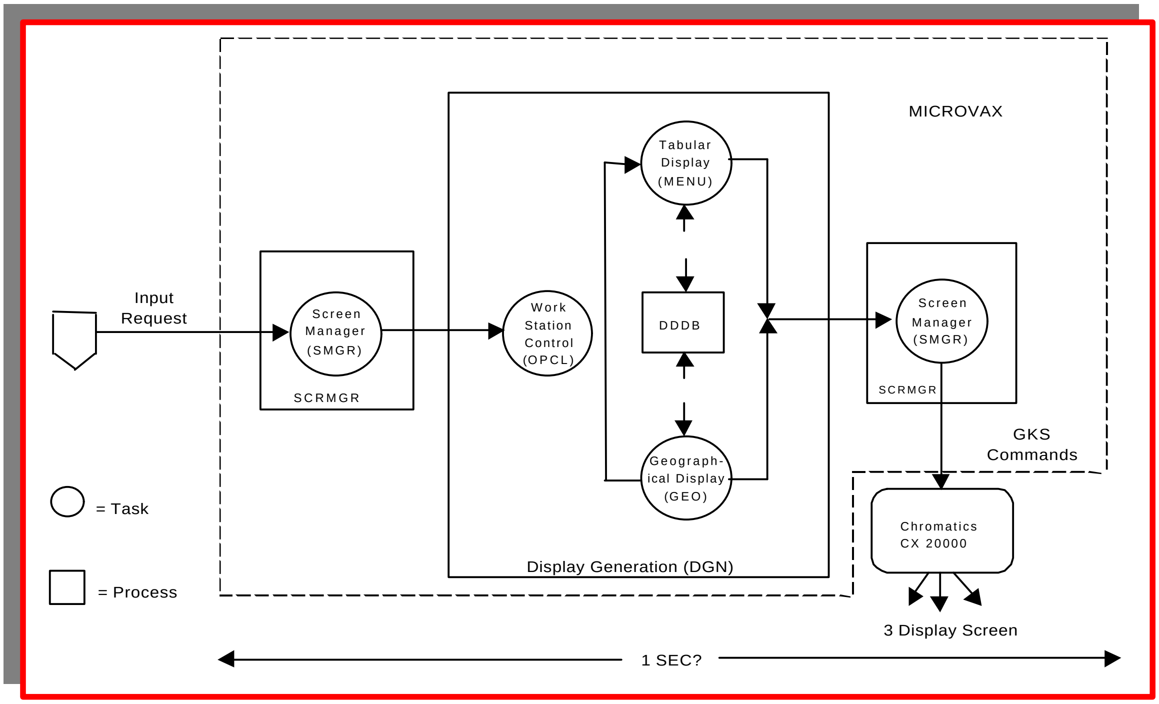

Figure F-11. Data Flow Reference Model

To understand the timing aspects of a system, it is often necessary to develop a simulation that models the system architecture and the load on hardware and software components or to execute benchmark software on the actual hardware. The validity of the results depends on how accurately the load, the data flows, the hardware speed and capacity, and the timing of individual processes are represented in the model — even the most elaborate model yields useless results if the parameters are not accurate. The designer of the architecture must be given accurate information to design the architecture and to evaluate its performance; in other words, it is essential that there be good communications between modelers and architects or designers.

Since the demands on the hardware resources will change over time, the architecture must provide the flexibility necessary to upgrade hardware to faster or larger processors to accommodate requirements for increased processing loads or faster response time. Similar increases in bandwidth may be necessary in communications hardware to provide for increased loads. Models that correspond to an architecture can be useful in planning for and evaluating the effect of changes in the hardware configuration of a system architecture to meet new demands.

F1.3.6 The Complexity of Architecture

Perhaps the main reason that we fail to address these different aspects of system architecture lies in the increasingly complex nature of the systems we build. Figure F-12 illustrates the top level system architecture of the Joint Surveillance Target Attack Radar System, or Joint STARS. The actual architecture includes many more computers, many different data busses, and a large number of other components (not shown in the figure) to perform its demanding mission. The result is a large, complicated system that makes it difficult for developers to consider the many different aspects of architecture.

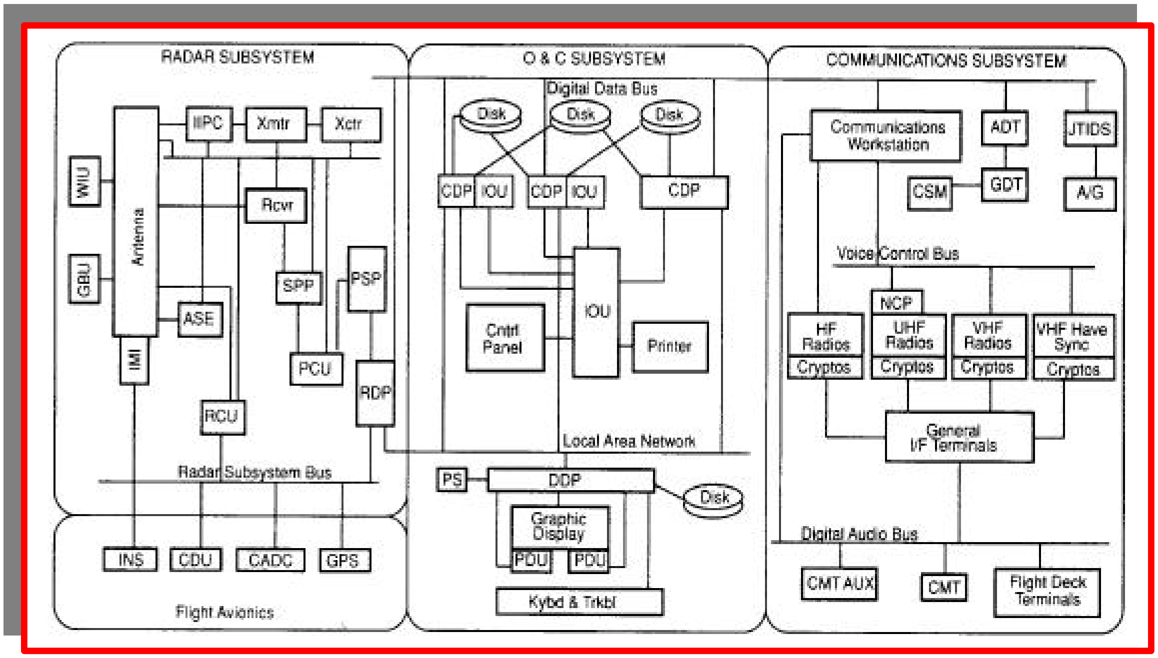

Figure F-12. Joint STARS System Architecture

At the same time, the larger and more complicated the system, the more important good structure becomes. Developing and maintaining structure may be very difficult in a system of such complexity, but the rewards for doing so are even greater. These rewards include higher quality during the initial development, lower life cycle software costs, and the increased likelihood that the system will remain in operation far longer (due to its greater flexibility and ease of upgrading). Furthermore, the reuse of known and expandable architectures will reduce the amount of new software that has to be developed and increase the quality of the systems that use them.

F1.4 Architecture: The Waiting Solution

F1.4.1 Technical Focus

At the start of a development program, when considering architecture pays the greatest dividends, the technical focus in the typical DoD program is often not on architecture. Rather, functional and performance requirements are generally focused on by both DoD and the contractor (refer to Figure F-13). This lack of attention to architecture occurs because the government expresses its requirements in terms of specific, measurable system functions and performance requirements that matter to the immediate user, and not in terms of flexibility, which matters to the maintainer and next-generation user. Government standards, such as DoD-STD-2167A, require proof that a design satisfies all functional requirements, not that it is adaptable to change. Design documentation and reviews track individual system and software components, with less attention on the overall architecture until the components are integrated.

Note: DoD-STD-2167A has been replaced by J-STD-016-1995 (EIA/IEEE)

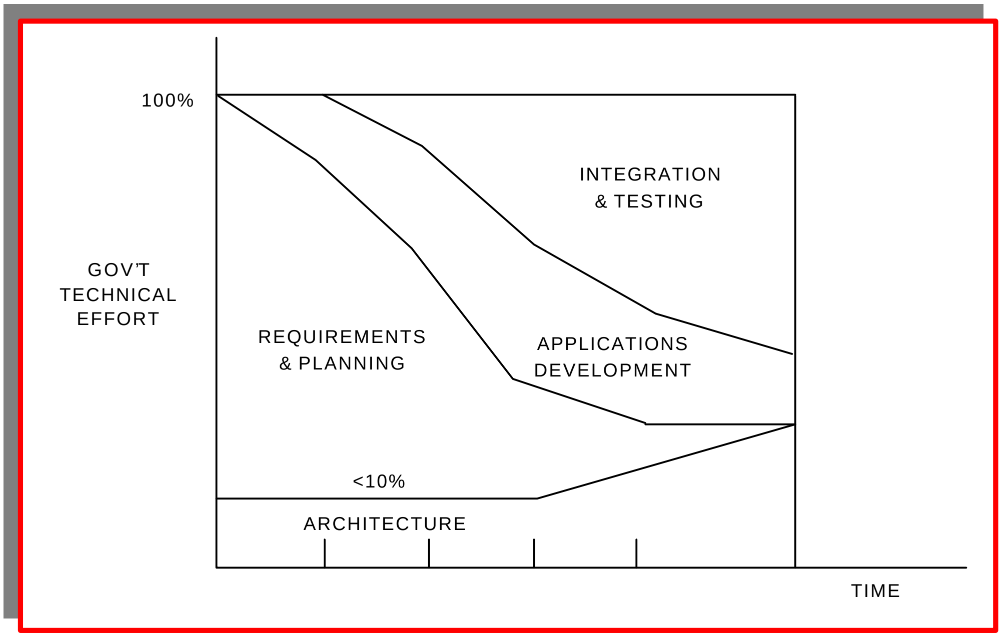

Figure F-13. Technical Focus (Estimated)

As the figure shows, the failure to consider architecture throughout the program's development has serious ramifications as time goes on. The performance and control problems described earlier begin to mount, and the contractor is often forced to call on Red Teams and other desperate measures to modify the original architecture. Since it is done in haste and then only to allow the product to meet the specifications, this last-minute change in architecture does little to ensure the necessary efficiency and flexibility, and usually results in further degeneration of the basic design.

F1.4.2 Faulty Emphasis

Both government and industry typically put almost all their efforts into the initial performance and functionality of a program in spite of the fact that these will change substantially over the life of the system. At the same time, there is a near-total lack of attention to an architectural baseline that would form a stable foundation for incorporating the system's changing requirements. What we do ask for does not address the important architectural issues. For example, we state that the system shall be modular but don't state a good way to partition it into modules that will allow future expansion and change.

We also specify requirements for system growth in an ineffective way that does not relate to operational capabilities such as adding new message types or increasing message traffic. Timing and sizing margins — for example, half the time and twice the memory — cannot ensure that the resources provided are allocated in such a way that they can be used to meet future requirements. With the advent of distributed systems, timing and communications bandwidth margins become important in providing for future growth. The government needs to assure that growth is expressed in operational terms, and not just in physical terms.

Because of the government emphasis on meeting immediate requirements within schedule and cost, even industry perceives that the government is not seriously interested in controlling maintenance costs. In a 1990 Air Force Scientific Advisory Board study of software maintenance, 123 businesses were asked what the government thinks is important when awarding software contracts. Their view of the government's stress on cost and system performance, rather than long term maintenance, is readily apparent.

Overall project cost	6.2 out of 7
Proposed product performance	5.5
Contractor experience in area	5.5
Timeliness	5.3
Last contract an advantage	4.8
Project software development cost	4.6
Contractor software capability	4.4
Ease of software maintenance	**3.4**
Software maintenance cost	**3.3**
Software portability	2.9

F1.4.3 Commercial Architecture

It has recently become evident that commercial software users have become more concerned with architecture. As users become familiar with vendors' capabilities, their expectations increase. In turn, many software vendors are now providing software interface standards that enable interoperability across different computer hardware families and allow users to pick and choose among competing software vendors. These commercial architecture trends can do nothing but help DoD software efforts, because DoD is a large buyer of software and hardware that support these interoperability standards. Even embedded, special-purpose militarized systems rely heavily on commercial systems to assist in software support. The DoD cannot try to take the lead because these standards are driven by the commercial marketplace; however, the DoD can use to advantage the opportunities in the commercial market for open architecture standards. Unfortunately, these commercial standards cannot include the service standards that are heavily application-dependent; these must be left to the application designer to establish and implement.

F1.4.4 Availability of Tools

The commercial market is also in the lead in providing tools that support the designer in generating and documenting architectures. There are tools to enable developers to analyze the linkage between different software modules, the control flow, the flow of data, and the timing of the various operations, and to assess and improve architectures. Many tools can only perform their analyses after the software is already written. These tools can still be used to understand what has been developed and to evaluate how easily it can be modified, before it is fielded or later. The investment may be small, and the potential payoff large. The following table lists some representative examples of available tools.

The government must acquire these tools and use them if it is going to buy architectures and understand them. In addition to commercially available tools, project-specific tools can improve the productivity of software development for a specific architectural design. Referring to the CCPDS-R program again, the contractor developed a tool to automatically generate the communications software that linked applications. The applications programmer needed only to list the elements of data that were required from each application and were necessary to each application. The tool used this information to generate efficient and correct communications following a standard pattern.

Name	Vendor	Analyzes
Logiscope	Verilog	Module structure, path coverage
ACT/BAT	McCabe	Flow graphs, structure graphs
ADAS	CADRE	Dynamic behavior, timing
STATEMATE	i-Logix	Structure, dynamic behavior
CPN	Meta	Dynamic behavior, simulation
Adagen	MarkV	Ada static structure, dynamic behavior
CARDtools	Ready	Timing threads
TAGS	Teledyne, Brown	Dynamic behavior, simulation

F1.5 Recommenations: Finding and Applying Architecture

Good architecture potentially can provide significant cost savings as well as greatly increased system flexibility. To obtain these benefits, we must put architectural requirements in system specifications, emphasize the early satisfaction of these architectural requirements, give contractors incentives to use proven architectural concepts, and control the architectural configuration over the life cycle of the system. We believe this can be done.

F1.5.1 System Specification

Since contract requirements drive the entire development of a system, the surest way to ensure adherence to a sound architecture is to put architectural requirements in the system specification. This does not mean that the specification will define the exact architecture to be used, but rather that it will specify what the architecture is to do. In cases when the application domain is well understood and a sound architecture is already available, the government may find it in its best interest to be more restrictive than in other situations lacking such a clear precedent.

To specify accurately the criteria architectures must meet, we must also determine how to qualify them. There are few measures of system designs that accurately predict flexibility and expandability. We will have to depend on a combination of techniques, including demonstrations that the system can be modified as well as analyses of features of the architecture. We are beginning to establish relationships between measurable features and rate of errors as well as ease of change. As Figure F-14 shows, the more calls a module makes on other modules, the more errors occur.

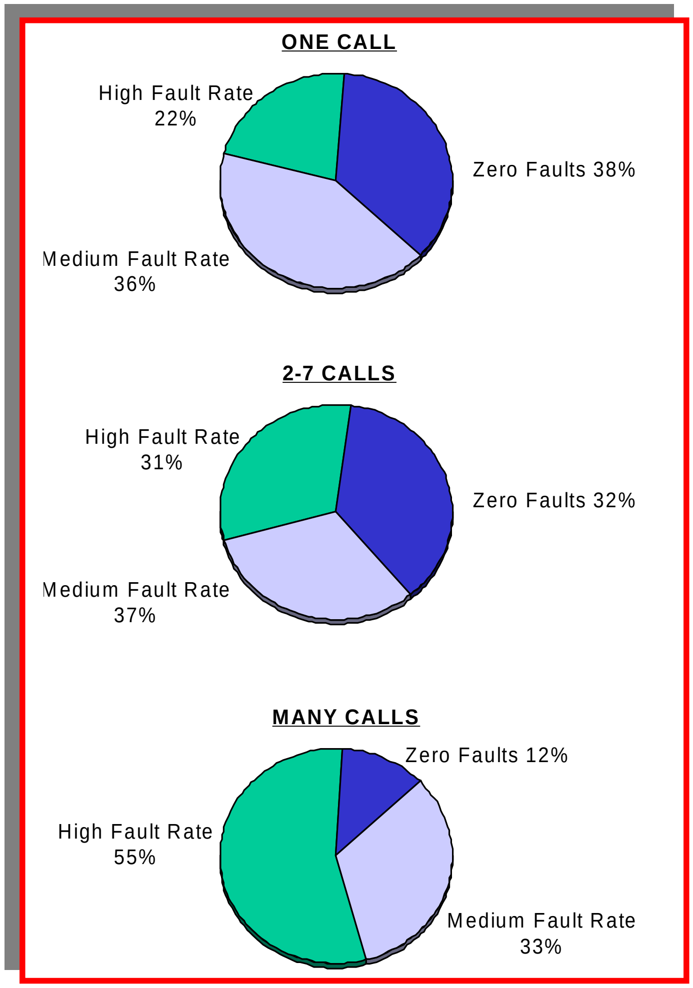

Figure F-14. Effect of Control Structure on Errors

F1.5.2 Early Satisfaction of Architectural Requirements

To reap the maximum possible benefit from architectural requirements, we should specify that contractors cannot write large amounts of applications software until they have developed an architecture that the DoD has evaluated and approved. The only applications software that would be written before this point would be that is necessary to help evaluate the architecture and reduce other serious risks, not to perform the actual task at hand. We can no longer afford the risk of developing architecture and applications concurrently; on the other hand, if contractors have successful architectures and control procedures that they have used before, they can use them again. In fact, the quality and effectiveness of a previous architecture as well as the tools available to support development of applications within the architecture should be an important factor in the selection of contractors on a program. We should also control these architectures after we evaluate and approve them. Changes would be weighed against the need for future flexibility throughout the life of the program.

F1.5.3 Contractor Incentives

Contractors will have to be given incentives to change from their current emphasis on meeting immediate requirements to a longer term view. They will have to set up their own controls to keep applications software writers from corrupting the architecture; in other words, during development, contractors will have the architecture under configuration control. Rules and standards have to be defined as part of the architecture. Tools should facilitate the integration and modification of components within the architecture so we know that the standards of the architecture are observed. Contractors who have good architectural awareness should be treated better than those who do not. The development community needs to start working on architecture with the software maintainers to ensure that we deliver to them whatever is necessary for them to sustain and use the architecture.

F1.5.4 Getting Started

It is recommended that the DoD put together a government and industry team to develop specification and contractual language for buying architectures. Approaches for evaluating and testing architecture need to be agreed upon as well. We are confident that this can be done and we have begun to develop an example. This team should also see that we use the experience that we have acquired on programs to determine what the contractors and the government have done to create good architectures, and to define the criteria for evaluating architectures.

We also need to consider buying single architectures for closely similar clusters of systems to reduce the cost of buying and maintaining unique architectures for each. For existing systems, we must work to introduce architectural improvements without disrupting operational use of the systems. It is crucial that industry participate as part of the team that would create the specification language and evaluation criteria. The insight of a joint government-industry working group on architecture will be of considerable benefit to the DoD during this time of changing missions and increased need for flexible systems.

F1.6 References

Arthur, L. J., Software Evolution, the Software Maintenance Challenge, John Wiley & Sons, New York, 1988

Card, D. N., Measuring Software Design Quality, Prentice-Hall, Englewood Cliffs, New Jersey, 1990

Day, R. A., "History of Software Maintenance for a Complex US Army Battlefield Automated System," Proceedings of the Conference on Software Maintenance, *IEEE*, New Jersey, 1985

DoD-STD-2167A, *Defense System Software Development*, 1988

Lehman, M. M., and Belady, L. A., Program Evolution — Processes of Software Change, Academic Press, New York, Academic Press, 1985

Rock-Evans, R., and Hales, K. Reverse Engineering: Markets, Methods, and Tools, Ovum, Ltd., England, 1990

United States Air Force Scientific Advisory Board, *Report of the Ad Hoc Committee on Post-Deployment Software Support*, US Government Printing Office, 1990

F2.0 Tab 2: A New Process for Acquiring Software Architecture

Barry M. Horowitz, Ph.D.
Thomas F. Saunders
Matt Mleziva
ESC-TR-94-207, September 1994

F2.1 Preface

This paper is the product of a substantial amount of thinking and work on the part of many contributors. Without their help, the topic might have been covered with an extensive, elaborate, but awkward discourse, or it might have been discussed at great length, without coming to any specific implementable recommendations. The quality of contributions from within the government and from the six industrial organizations who participated in developing the concepts and preparing the paper are greatly appreciated. The primary contributors were:

ESC Col T. Mackey, ESC/AL	**MITRE** A. Buchanan
Boeing J. H. Hanson R. L. Roe	J. A. Clapp S. W. Dardinski H. G. Goldman
GTE C. E. Ellingson J. Perry J. Roder	R. D. Haggarty E. J. Hammond C. G. Hanley M. R. Kurland
LORAL J. Stanfield	C. W. McClure R. Platcow H. W. Sorenson
PARAMAX R. Irwin D. G. Perry W. D. Sturm	**RAYTHEON** T. J. Haley
	TRW Dr. W. W. Royce

F2.2 Introduction: A Process to Control Software Architecture

Military systems must be able to change to accommodate new mission needs, threats, and technology. At the same time, military systems are being developed with budgets that are more and more limited. We need to find ways to keep costs down and to maximize adaptability — for new systems and for extensions to existing systems that must be preserved. A system's software architecture is a key determinant of the system's adaptability. A well-conceived and well-maintained architecture allows reusable components to be included in the original development, custom components to be smoothly integrated via standard interface protocols, and improved components to be incorporated as replacements or enhancements are needed. To be useful, the software architecture must first be articulated and include provisions for change; second, it must be controlled and maintained throughout the system's life cycle.

In this paper, we define a process that can be used to ensure that system acquisitions include attention to the software architecture. Attention to software architecture begins with the very first discussions of the system's scope and concept, and extends through system maintenance. The periods in a system's life critical to establishing and retaining a good architecture extend from formal notification to industry of the government's need for the system, through the evaluation of industry's response, the sequence of design reviews during the contractor's design and implementation, and long-term maintenance of the operational system. Before describing the process, we must first identify how to describe software architecture. In general, the attributes of software architecture include:

- Software partitioning,
- Flow of data,
- Flow of control,
- Timing and throughput relationships,
- Interface layering and protocol standards, and
- Hardware/software allocation.

Although these attributes may overlap with concepts considered to be *system* architecture, regardless, these attributes need to be established before software architecture can be evaluated or controlled. The specific content of what is controlled under software architecture must be determined for each program. Similarly, the level of detail contained in descriptions of the software architecture attributes, as well as distinctions between architecture and design, must be determined for each program. Attributes should be considered architecture when they express relationships that contribute to the system's long-term tolerance to changes; they should be considered design when they are implementation-specific.

We provide some background behind the initiative to have architecture defined and preserved in modern acquisitions,* and we define the term *software architecture*. We also provide information that can be used to structure the package the government uses to solicit proposals from industry for systems that strongly depend on computer software, and we offer guidance for monitoring the execution of a software development contract.

We believe that following the process described here for acquiring software will result in systems that better meet user needs by encouraging the development of more maintainable, flexible, extensible software architectures. The acquisition of software architectures will be most effective when there is cooperation between the government and industry, and attention applied by both towards the goal of building systems that can accommodate change.

F2.3 Back-ground: The Importance of Architecture to Software Flexibility

One of the prime benefits originally attributed to the use of software is its inherent flexibility. Unfortunately, for a variety of reasons, the potential benefits of such flexibility have been lost in many systems. In fact, users identify the lack of flexibility and adaptability as two of the major disadvantages of recent systems. Users have been forced to continue using outdated (and sometimes improper) procedures solely because of the expense required (time as well as money) to modify the computer software. The rigidity of some software has even dictated that obsolete hardware be retained because of the difficulty of moving the software to newer machines.

If a system is to have a long, useful life, the allocation of software capabilities to components may need to be altered. For example, technology advances may make available specialized hardware that can substitute for particular software components, new software algorithms may replace hardware devices, or future mission capabilities may be predicted. Buyers should not only anticipate such possibilities, they should provide bidders with an indication of future needs via a vision statement. Likewise, bidders are encouraged to propose structured architectures with appropriately selected components so that the system can be expected to have a long and stable life. Bidders must address how one or more proposed architectures provide for the prospect of a long system life.

With the anticipated duration of modern systems, it is vitally important that systems be able to evolve as application needs change and as users come to appreciate the potential capabilities of systems. Although at the outset of a systems development project the designers are able to describe desirable characteristics and a logical structure, by the time the systems are delivered their structures are too often constrained and awkward. Instead of receiving well-organized systems with clean interfaces, users receive systems whose distribution of functions appears contrived.

It has been observed that the original organized structure becomes distorted as the detailed designers choose expedient solutions to challenging implementation problems. As these expediencies compound, the software architecture of the delivered system becomes more and more convoluted. This convoluted structure makes it nearly impossible for the implementers or maintainers to find a logical point for introducing needed functions, and leads either to prohibitively rising costs or to ossification and eventual discard of the system. Such an end to well-conceived systems is consistently encountered because modifications, sometimes extensive, are inevitably needed during the course of a system's operational life.

To obtain the benefits of flexibility within a system, the software's structural attributes, which are present at the beginning of the design thought process, must be captured. These attributes are typically called "*architecture*," although the subtle distinction between architecture and design varies among practitioners. What is important is that a management process be established for preserving the architecture. This process must ensure that:

- A vision is established and documented that conveys a sense of direction for future capability growth;
- A structure is defined for the software, and is stable before decisions crucial to the implementation are fixed;
- The structure and its rationale are formally recorded for the implementers and approvers of the design; and
- The structure is revisited and reaffirmed, or modified in a controlled manner and preserved, throughout the life cycle.

For the process proposed here to work effectively, the government and contractor must designate people to oversee the architecture and ensure the system implementation preserves the architecture's desirable aspects.

F2.3.1 Cost Considerations of a Flexible Architecture

Cost is sometimes raised as a concern about trying to maintain the integrity of the architecture structure. Although a software acquisition policy may emphasize software architecture, there is a system architecture tradeoff to be made. Hardware capabilities and software functions may need to be included that were not in the initially specified requirements but are necessary to accommodate changes predicted for the future. Offerors and buyers must consider the long-term payoff for preserving software architecture. With the falling price of computer components, it is preferable in most situations to spend more money on hardware rather than sacrifice the system's extensibility because of a shortsighted compromise of the software architecture to reduce hardware costs. In fact, an architecture that allows reuse of software components or use of commercial software products may lower the system's total cost. Appropriate consideration of the full life cycle cost consequences must be included in any decisions regarding the generation, preservation, or abandonment of a software architecture.

F2.3.2 Software Architecture Defined

Software architecture refers to the fundamental structural attributes of a software system. Software architecture is a top-level definition of a software design that is defined early in a system's life cycle. It is the result of system design activity to synthesize a software system that will support the system's functions; be in concert with a synthesized hardware system; be responsive to imposed developmental, environmental, and operational conditions; and be demonstrably supportive of a vision for growth and change. The software architecture defines the software components that will provide the required algorithmic functions, their interfaces, and an underlying execution concept for orderly and efficient accomplishment of the algorithmic functions. As design progresses, the architecture captures more specific information, such as a software system execution model that includes specific operating system selections, specific interface and communication protocols, and specific multiprocessing and multiprogramming paradigms.

The distinction between software architecture and software design is not absolute. However, design is considered more detailed; it is a realization of the architecture into a product that fulfills the system requirements. The essential properties that must be described to express a software architecture include:

- **Partitioning software into components**. These should be the major software entities that will be recognized and manipulated by the software developers as the natural partitions within the software. In this context, "component" is used in a general sense and does not specifically refer to Computer Software Components (CSCs). Components may be determined by grouping algorithmic functions, objects, and reusable components such as operating systems and database systems, or by any other scheme appropriate to the proposer's development methodology.
- **Flow of data**. This should reveal the mechanisms for managing the flow of data through the components. It should show the major data paths between and among transformation processes as well as to/from data storage. Where the flow of data is controlled by table-driven design, the table data structures and their rules for modifying data flow should be described. It should define data structures and how file management and database management structures will be used.

- **Flow of control**. This should reveal the mechanisms for managing the flow of control among the components. It should show the major execution sequences, where execution sequences may be asynchronous or parallel, and how synchronization is managed. Where execution control uses data-driven design, the mechanisms for accessing the table and managing execution should be described. It should also show how anomalous conditions such as error handling and exception conditions that may dynamically alter the flow of execution are managed.
- **Critical timing and throughput attributes**. The architectural devices used to manage critical timing events, interrupts, throughput demands, and data buffering should be described.
- **Interconnection layers, standards, and protocols**. The layered view should show the grouping of software components into layers containing collections of services whose interfaces to the rest of the system are defined. It should specify the rules for allowable interconnections among the layers. The use of standardized interface protocols and bindings within and between layers, such as SQL, GOSIP, Motif, OSI, IEEE 806.X, etc., should be indicated.
- **Allocation of software to hardware**. The mechanism for assigning software to specific hardware devices should be defined. Where critical design performance is determined by such allocation (e.g., signal processing software might need to be executed on custom systolic array hardware), the allocation should be defined.

The definition of what constitutes a "*good*" software architecture is not provided here. Goodness must be judged on a case-by-case basis during source selection and each time modifications or refinements to the software architecture are proposed during a system's life cycle. In general, architectures are judged to be good when they satisfy the objectives outlined in the government's specification and vision statement.

F2.3.3 Preparing a Request for Proposal

The government's expectations about a new system are conveyed to industry as a Request for Proposal (RFP), which comprises a number of documents. For software architecture information to be adequately evoked, the RFP package must include specific information that traditionally has not been included in the RFP. The package must make clear to potential bidders that the government's assessment of the proposal will include assessing the software architecture.

F2.3.3.1 System/Segment Specification

The System/Segment Specification (SSS) provided with the RFP package provides functional and performance requirements that must be satisfied by the contractor. The SSS also contains requirements, such as interface standards, language constraints, fault tolerance, and security, that influence architecture and must be considered by the contractor when the architecture is being defined. The SSS should include the requirements necessary for the system to meet the user's known needs. These may include architectural requirements when specific (testable) flexibility and extensibility requirements are known. However, the SSS should not be developed so as to dictate a particular architectural solution. Rather, it should emphasize the need for long-lived viability of the proposed architecture. If an acquisition has a requirement for exact interfacing with (or replacement for) an existing system, then the architecture may be constrained to the extent that architectural properties constitute a legitimate requirement, such as conforming to standards or using standard components. Even in this case, though, the SSS should not impose unnecessary restrictions on the architecture.

F2.3.3.2 Vision Statement

In this new process for acquiring software architecture, acquisitions or developments should include a vision statement in the RFP package. The capabilities described in the vision are not prerequisites for the successful proposal; however, architectural flexibility sufficient to accommodate those capabilities is sought by the government. The SSS contains known requirements the delivered system must fulfill; the vision statement describes new capabilities the system might need to provide in the future.

In the vision statement, the user should provide, to the extent possible, potential future changes in threats and in the role and mission of the system. The acquisition organization should integrate the user's vision of future changes with other potential changes, including technology advances, and prepare the vision statement. Examples of what might appear in a vision statement include new functions (e.g., provide routing directives in addition to tracking targets), anticipated technological improvements (e.g., provide larger or better displays), and radically innovative technology improvements (e.g., direct input through voice recognition). The architecture-relevant topics in the SSS and the vision statement together create the basis for evaluating proposals.

If there are known requirements for flexibility, maintainability, and future growth that must be met in the delivered system and are firm when the RFP is issued, then they must be a part of the SSS (e.g., if the system must accommodate growth and changes in the message catalog), and the software architecture must meet them. The vision statement adds another dimension to the SSS by helping the offeror make tradeoffs in selecting an architecture that can easily adapt to fulfill potential future requirements at minimum risk to the government, the user, and the maintainer.

A vision statement is important to the architecture process because major system benefits may emerge if the chosen structure is capable of handling different demands. With a robust and flexible underlying structure, a system can absorb tasks different from those identified to meet the initial deployment. The vision statement requires thinking beyond the immediate objectives of the users in acquiring a new system. It should be linked to the user's long-range planning and speculation (10 to 20 years), or to potential alternative users, and should not be restricted by predefined allocations of functions to organizations, systems, or persons.

F2.3.3.3 Statement of Work

The Statement of Work (SOW) must reinforce the importance of the architecture in the acquisition by identifying tasks related to documenting the architecture, keeping the documentation relevant as the design progresses, and validating that the architecture is used in the actual software design and implementation and that it allows satisfaction of system requirements.

The System/Segment Design Document (S/SDD) is the vehicle for expressing and controlling the architecture. Consequently, a specific task must be included in the SOW to ensure that the S/SDD is produced, updated, and maintained under configuration control by the contractor.

The Software Development Plan (SDP) is the vehicle for defining the software development process the contractor expects to use. The SDP must include a description of the processes and the tools or software engineering environment the contractor will use to ensure the architecture is preserved by the software implementation. Consequently, a specific task to develop and maintain the SDP must be included in the SOW.

Although this process is devoted to preserving architecture, it is not devoted to preventing its refinement or modification. Consequently, engineering analysis tasks to evaluate, refine, or demonstrate the validity of the software architecture must be included in the SOW. Results of these analyses should be discussed within design reviews (e.g., at the software walkthrough) and at presentations to the government. These engineering analysis tasks should reveal how the architecture will behave in the proposed application, and how it can be generated, modeled, or prototyped with the contractor's proposed software development process and environment. These tasks should commence prior to the System Requirements Review (SRR). Because of the need to perform these studies, the schedule for the contract must allow time before the SRR. The results of the analyses must be presented to the government at the SRR, and again prior to the SDR, as drafts of the architecture portion of the S/SDD. Both the architecture and the analyses that provide the rationale should be documented in the S/SDD and used to reach an agreement between the developer and the government on the architecture. At SDR, the architecture portion of the S/SDD should be placed under configuration control. A task must be defined to conduct at least one Technical Interchange Meeting (TIM) for the contractor and the government to review and agree on the architecture between SRR and SDR.

Additional engineering tasks should be included in the SOW for validation of the architecture itself, and preservation of the architecture throughout the implementation. At a minimum, updates to the S/SDD should be provided at each formal design review (after the SDR, PDR, and CDR), at any time an architectural change is proposed, and at Test Readiness Reviews (TRRs).

Most software architectures involve a complex structure that is difficult to understand and therefore difficult to predict. Early in the formulation of the software architecture, a model or prototype is needed; this should be an executable model that could be simulated to yield insights into the software component interface relationships, data flow, execution flow, and timing and sizing performance of the ultimate system. Where complexity of the ultimate system warrants (to be decided case-by-case), a specific task to develop, document, and use such a prototype/model should be included in the SOW. The prototype/model will validate that the architecture and the implementation coincide and that the architecture can meet system and software requirements. The prototype/model is not required to be an extract of the final version of the system, although it could be; rather, it is prepared to allow meaningful examination of the architecture. Results of analysis of the prototype/model should be delivered to the government at each design review. In some cases, the prototype/model may also be a deliverable for use during the support phase of the system.

Within the context of any implementation, the risk of failing to meet some of the requirements must be carefully managed. To ensure that architectural features are not needlessly abandoned when attempting to resolve other problems, the contractor should be required to analyze the proposed system for features likely to pose implementation difficulties or increased costs. Any requirement, capability, feature, or option that, during the course of the contract, is discovered to fall into the realm of a potential risk to preserving the architecture and its attributes must be identified to the government and a risk management action proposed.

A task must be defined to conduct tradeoff studies whenever a change to the architecture is contemplated. The tradeoff studies should involve simulation modeling or demonstration-based results that provide a quantifiable impact to the architecture attributes. The results of such tradeoff studies must be presented to the government, which must concur before any efforts are initiated that implement the changes. Government concurrence should be managed according to the procedures for managing the S/SDD configuration. Attachment I contains sample wording that should be in the SOW. However, the exact definition of the tasks to be performed must agree with the goals of the system under development.

F2.3.4 Contract Data Requirements List (CDRL) System/Segment Design Document

The S/SDD is the vehicle for defining the design of a system/segment and its operational and support environments. Under this new process for acquiring software architecture, the architectural portions of the S/SDD will be expanded. Further, architecture information will be requested from the contractor at the time of the proposal, at SRR, and before SDR. The proposal submittal may be in the form of a draft of the architecture material from the S/SDD or it may be separately formatted. In either case the information necessary to understand and evaluate the software architecture must be presented.

In accordance with DI-CMAN-80534 (S/SDD DID) and Attachment II of this paper, a description of the architecture should be provided with the proposal, a preliminary description should be provided at SRR, and an approved architecture description should be provided before SDR. Obviously, the early submissions will have less detail defined than the later ones. For example, specific choices for software decomposition are not expected in the proposal. Only the components that represent significant software capabilities should be identified as software architecture components (e.g., operating system, database management system, signal processing system, or other breakdown of components consistent with the developer's proposed methodology, such as functional decomposition or object-oriented design). The contractor must also be required to submit revisions to the S/SDD whenever an architecture modification has been approved and the architectural information has become outdated. In general, the software architecture information should be contained in Section 3.4 of the S/SDD.

Many diagrammatic or textual representations are possible to emphasize software attributes such as data flow, execution flow, layering, or interface protocols. The specifics of the notation and format of the information are not as important as recording all the architectural information; for that reason, one or more contractor-preferred representations may be selected to complement one another when assembling an architecture description. Regardless of the forms used, to be acceptable the architecture representation must cover the essential properties listed in our definition of software architecture. In addition to the architecture description, the S/SDD must be supplemented with explanatory material for the architecture. The contractor must include the following rationales for:

- **Structure of software**. Explain why the partition of software components was selected, emphasizing why it offers tolerance for changes in later stages of the system's life cycle. Similarly, explain why architectural features such as table-driven design were selected and what attributes of tolerance for change, e.g., flexibility and extensibility, are provided by the chosen software structure. Mechanisms for managing the flow of data and the flow of execution should also be described, with the rationale behind how they were selected.
- **Rules for interaction of system components**. Explain why particular standards or de facto standards (e.g., Motif, Windows, POSIX GOSIP, Hewlett Packard New Wave Object Management Facility) are being adopted. Provide the rationale for how an architecture model or profile was selected and how interconnection rules or layered protocols will be enforced in the architecture.
- **Structure of the hardware and software**. Provide the rationale for the choices made to aggregate software components on processors or to distribute them on separate processors.

In addition to discussing how the architecture relates to the needs of the current system, the S/SDD should address how the architecture will handle new capabilities and features as noted in the vision statement. The architecture description should include provisions for exploiting migration paths associated with improved versions of nondevelopmental item (NDI) products, and it should describe the limitations or capabilities to migrate to improved custom or NDI software. It should also explain what aspects of the software are substantially enhanced by the proposed architecture, for example:

- Software testability
- Software maintainability (e.g., isolating and correcting software malfunctions, and
- Software extensibility.

Attachment II contains a candidate rewording of Data Item Description (DID) DI-CMAN-80534 to direct the contractor's preparation of the S/SDD. Caution: when preparing the description of the architecture, the distinction between design and architecture needs to be considered. Since the architecture will be placed under configuration control relatively early in the contract life cycle, implementation detail information is best left out. The mechanisms, rules, standards, and generic structural properties should be included. Specific, low level, component-unique information that is not part of the main structure of the system should not be captured. The distinction between what is design detail and what is structural attribute needs careful consideration by both the government and the contractor. Consequently, the words used to tailor a DID for a specific contract should be carefully considered.

Software Development Plan. The software architecture implementation is carried out by the creation of many software components, modules, and routines. The creation steps must be guided by a process, defined in the SDP, that is fully cognizant of the importance of the software architecture. Therefore, not only is an SDP required from the contractor at the beginning of the contract, it must be resubmitted whenever changes are made to the development process. The SDP must describe the development environment and any tools used to ensure and/or verify the preservation of architectural features during development. The degree to which tools can be used to enforce architectural decisions should be highlighted. Whether or not automated tools are used, the SDP must explain the procedures by which architectural constraints are enforced throughout the design, coding, deployment, and maintenance phases of the system. Information to be included in the SDP includes:

- The extent to which software components are NDIs or reused, and guidelines for determining the suitability of reused or NDI products.
- The process (sequence of activities) and tools that will be used to manage the software architecture, including:
  - Generating software in accordance with the rules of the architecture;
  - Documenting and controlling the architecture and changes to it;
  - Verifying that the architecture will meet its functional requirements and its requirements for adaptability, extensibility, etc.;
  - Predicting that the system will meet its performance (timing) requirements;
  - Controlling conformance of the software design and software implementation with the architecture;
  - Determining how and when the architectural model or prototype will be executed; and
  - Creating an organizational structure to support architectural development, preservation, and verification.

Attachment III contains a candidate rewording of DID DI-MCCR-80030A to direct the contractor's preparation of the SDP.

Other documents. There may be other documents (e.g., model/prototype description, timing and sizing reports) that are influenced by architecture. For each one, the DID should be amended to express the need for preserving software architecture.

F2.3.4.1 Instructions for Proposal Preparation

The Instructions for Proposal Preparation (IFPP) must remind bidders about the presence of the software architecture policy and the government's emphasis on software architecture definition, evaluation, and preservation. The IFPP should reiterate the relationship between the specification and the vision statement, the parameters and attributes to be included in the software architecture definition, and the differences intended between architecture and design. It must provide some form of completeness criteria that will enable contractors to know how much of the vision statement characteristics need to be discussed in a proposal. It must also provide for enumerated vision attributes so proposals that offer solutions covering more of the vision can be distinguished from those covering less.

Bidders must be informed of the need to provide a description of the software architecture. This will be a tailored subset of the information contained in the S/SDD. The S/SDD material should be of sufficient depth for the relationship between proposed architectures and the specification and vision attributes to be understood by the buyers. This architecture-relevant information should have a page limitation (e.g., 50 pages, including cover, index, and supporting text). The bidder must adequately describe the architectural approach to be used and must show how a proposed architecture will support the system requirements by including a rationale for the major architectural properties the offeror proposes. Further elaboration on the proposed architecture and its rationale may include descriptive examples of previous systems that used a proposed architecture, results of demonstrations from previous contracts, or results of simulations and models available for a proposed architecture.

The IFPP must also inform bidders of the need to submit a description of the software development processes and the contractor's plan for developing software. This material should define the processes and tools used to support creating and preserving the software architecture. The technical proposal should provide the rationale behind why and how those software development processes and tools ensure preservation of the architecture. Where appropriate, examples from past experience should be cited.

A presentation of the techniques used in a previous project should be encouraged to lend credibility to a bidder's representations about the approach to emphasizing architecture that is proposed. An ideal way of combining past experience and the current effort would be to have the bidder demonstrate the planned system architecture and tools through execution of a high-level model of a proposed architecture during source selection. On a case-by-case basis, the acquisition organization should determine whether or not it is appropriate to expect or require proposers to be prepared to include a demonstration as part of source selection. The bidders should be encouraged to show how their model reveals whether an architecture supports the specification and vision. For example, the model could show the ability to port software components to multiple hardware platforms, to modify components via the use of computer-aided software engineering (CASE) tools, or to replace one commercial-off-the-shelf (COTS) product with another.

Because the ultimate success of the architectural approach to the system acquisition will be determined by the designers and implementers, the role of the technical leaders of the design team is of much interest to the government. The bidders are encouraged to describe whether a single architect or architect team will be appointed, the scope of this person's or team's authority, and how the architect or team is expected to interact with the rest of the project management team. Examples of previous projects that used the proposed management structure would validate the likely success of the approach. Attachment IV contains sample wording that can be used in the IFPP.

F2.3.4.2 Evaluation Criteria

So that bidders will understand the importance attached to the architectural structure of a proposed system, Section M of the solicitation package must indicate that the proposed architecture, its development process, and its maintenance provisions will be evaluated. Further, it should state that failure to provide an adequate response to the architectural aspects of the proposed system will be deemed to indicate noncompliance with the basic solicitation. The actual factors and standards that the government will use in evaluating the proposals will be based on the statements in Section M, but will not be provided to the bidders. The evaluation criteria included in Section M should be based on a need to assess:

- How the proposed approach and architecture meet system requirements;
- To what extent the architecture can meet the potential long-term needs of the system described in the visio statement; and
- To what extent the proposer's software development approach assures the preservation of the software architecture.

Although these criteria are generic here, they must be made more explicit for the particular system being acquired. The evaluation criteria should be derived from the types of tasks or capabilities the new system may need to accommodate, and they should allow buyers to evaluate whether or not proposed architectures have a flexible underlying structure that will accommodate tasks different from those identified by the initial specification.

F2.3.5 Activities After Contract Award

After the system implementation is under way, the government and the contractor must jointly monitor the effort to ensure that the architectural base for the system is maintained. The SOW defines the appropriate tasks and, for the most part, the contract defines the activities after contract award. However, cooperation is considered essential both in selecting the architecture to be used and in preserving it. Therefore, all parties to the contract should be encouraged to establish open communications and candid evaluation of the degree of success being achieved with respect to defining a sensible architecture and preserving it. TIMs can facilitate communication prior to formal reviews.

F2.3.5.1 Demonstration/Validation Effort

Engineering analyses should allow the contractor and government to identify and quantify the parameters that relate to imposing the chosen architecture on the system. They should be conducted from the outset of the contract to refine the architecture requirements and prepare the contractor and the government for architecture discussions at SRR and prior to SDR. They should also be conducted any time architectural modifications are proposed.

F2.3.5.2 Modeling Efforts

A realistic (but economical) modeling effort should be part of any complex project so that probable performance can be predicted. Without appropriate modeling, the true consequences of preserving the architecture in the face of possible design deviations may not be understood. These modeling efforts should include the architecture aspects of the system.

F2.3.5.3 Design Reviews

As a part of each design review during development, the integrity of the architecture must be explicitly reviewed. Any proposed deviations from the agreed architecture must be explained. Execution threads must be presented as a means of showing that the architecture supports the intended functions of the system.

F2.3.5.4 Documentation and Configuration Control

Between contract award and SRR, the contractor and the government will study the proposed architecture. Based on the results of engineering analyses done during this time, the architecture requirements will become stable. Between SRR and SDR, the contractor will prepare the formal S/SDD for review and comment by SDR. Upon submittal of the S/SDD, reflecting government comments, the architecture will be placed under configuration control. Thereafter, modifications may be made to the architecture but only after tradeoff studies have shown the necessity for change, and the government and the contractor agree to the change. The S/SDD should be updated to reflect the approved changes.

*Horowitz, B. M., *The Importance of Architecture in DoD Software*, The MITRE Corporation (M91-35), Bedford, Massachusetts, July 1991

F2.3.6 Attachment I — Statement of Work

The following material is provided for placement in the SOW to define tasks associated with preserving the software architecture. These paragraphs are general in nature and must be tailored to include specific references to the CDRL. The specific tasks chosen and the degree of tailoring must also match the needs of each program. In particular, the scope and scheduled completion of the tasks must be integrated into the overall plan for the program. The software architecture tasks are intended to supplement those tasks that would be part of an acquisition effort and thus already specified in the SOW; it may be convenient to merge this material into the descriptions of other tasks.

F2.3.6.1 x.x.x System Engineering — General

When performing all system engineering tasks, the contractor shall ensure that the software architectural attributes are considered. All trade studies, design decisions, and implementation actions that impact the software architecture will evaluate whether or not the system's tolerance for change is affected. In the event that it is affected, the contractor shall include the analysis that justifies modifications to the software architecture in terms of life cycle cost of the system.

F2.3.6.2 x.x.x System Engineering — Analysis

Before the SRR, the contractor shall analyze the SSS and vision statement and identify requirements that are judged to be architectural drivers. The contractor shall host a TIM, with government and support contractor participation, to review the software architecture. The purpose of the TIM will be to review the requirements from the SSS being satisfied by the architecture proposed, to assess the ability of the proposed architecture to satisfy the vision statement, and to coordinate revisions of the architecture requirements such that a clear understanding of the software architecture and the criteria or constraints involved in its definition is established. The architecture shall be described in an update of the architecture portion of the

S/SDD and shall be provided to the government at the SRR. A formal submission of the architecture description in the S/SDD shall be submitted and placed under configuration control at SDR. [DI-CMAN-80534]

An explanation of the architecture of the system shall be included as primary presentations at the SRR, the SDR, all software and hardware design reviews (SRR, PDR, and CDR), and the TRR. Associated with software architecture descriptions, the design process standards and tradeoff heuristics that were used as criteria or constraints by the contractor, or by the contractor and the government, for selecting the architecture shall be expressed in an Architecture Analysis Report. This report shall contain the essential information for later reviewers to understand why the architecture choices were made that led to the defined architecture. The intent is for later reviewers to be able to identify architectural features that are tied to design assumptions or constraints. In the event those assumptions are refined or the constraints are lifted, preservation of those architecture features may be reexamined. Also included in the report shall be a description of the procedures, tools, and training by which the government can maintain the architecture for the remainder of the planned system life after completion of the contractor efforts. This report shall be delivered at SRR, at SDR, and updated at any subsequent review whenever modifications to the software architecture are requested. [DI-MISC-xxxxx]

F2.3.6.3 x.x.x Software Engineering

F2.3.6.3.1 x.x.x Software Engineering — General

The contractor shall manage, design, develop, document, control, and qualify performance of the computer software to satisfy the performance, functional, and quality requirements of the SSS. In addition, to the extent technology allows, the contractor shall provide an architectural framework for the software that accommodates the flexibility and extensibility implied by the vision statement. "*The extent technology allows*" will be determined via TIMs where provisions for open system design, software reuse, incorporation of layered architecture reference models, and interface standards are compared against the goals of the program and the state of available commercial products, or the state of contractor-developed architectural products. The software architecture definition shall be governed by the long term cost-effectiveness goals to have the system under development be tolerant to change. Decisions vis a vis architecture selection or modification shall be made on the basis of requirements defined in the SSS, the vision statement, and this goal within the scope of the contract.

F2.3.6.3.2 x.x.x Software Engineering — Architecture Change Analysis

At each milestone after the SRR (i.e. SDR, PDR, CDR, and TRR), the contractor shall identify any new enhancements or changes to the architecture that affect the long term tolerance for change in requirements, and provide reasons for the differences. Before these changes are presented at the next review, the contractor shall identify the changes at a TIM convened 20 days prior to the review. The contractor shall update the S/SDD to reflect the changes as agreed within 30 days after the review. If any changes affect portions of the architecture already under government configuration control, the contractor shall be required to generate an administrative engineering change proposal (ECP) describing the changes and their rationales. [DI-MISC-xxxx]

F2.3.6.3.3 x.x.x Software Engineering — Architectural Model or Executable Prototype

The contractor shall prepare an executable prototype, model, or initial version of the software architecture framework. This model shall demonstrate the flexibility, extensibility, and growth provisions anticipated for the selected architecture. It shall allow testing of the timing and throughput relationships, the interfaces to products using standard protocols, and the fundamental data flow and control flow sequencing, and it shall demonstrate the anomalous condition or error handling mechanisms to be used in the system. The contractor shall present the results of architecture modeling or prototype developments and demonstrations at the SDR, and the results and the model shall be updated at the PDR and CDR. The contractor shall submit an agenda for each demonstration and shall deliver a report after each demonstration to record any action items and decisions. [DI-MISC-xxxx]

F2.3.6.3.4 x.x.x Software Engineering — Database Design

In the process of designing the logical and physical structure of the database, the contractor shall consider the effects upon the database of the selected architecture (and vice versa) and shall ensure that the database design is consistent with the architecture. The design of the database shall be presented as part of each design review at a level of detail determined by the maturity of the design effort at that time. The contractor shall deliver to the government a Database Design Document (DID-MISC-xxxx). A preliminary version of this document shall be delivered at the (first) PDR and a final version of this document shall be delivered at the (first) CDR.

At the SDR, the contractor shall describe the approach for implementing the database. The contractor shall identify and justify the need for any file management or database management software. The contractor shall discuss the partitioning of the database and shall identify benefits and drawbacks due to the planned database structure. Included shall be a description of the procedures, tools, and training by which the government can maintain the database consistent with the architecture for the remainder of the planned system life after completion of the contractor efforts.

F2.3.6.3.5 x.x.x Software Engineering — User Interface Design

The contractor shall develop the user system interface (USI) to be consistent with the planned architecture. The impact of the architecture on the USI (and vice versa) shall be considered. At the SDR, the contractor shall produce a description of the sequence and timing of user actions and software responses to illustrate the compliance of the USI with the overall architecture. In particular, the handling of anomalous and error situations and the resultant movement of information to and from the USI shall be discussed. The use of previously developed software in the design or operation of the USI shall be described. The contractor shall provide information sufficiently detailed so that the government can determine that the USI design and the planned hardware and software implementations are consistent with the planned architecture. This information shall be delivered in the form of briefings and portions of the S/SDD. [DI-CMAN-80534]

F2.3.6.4 x.x.x Security Engineering

The design of needed security features shall be accomplished by the contractor in such a manner as to preserve the architecture planned for the system. The impact of security features upon the overall architecture and design shall be presented as part of each design review at a level of detail determined by the maturity of the design effort at that time. No alterations to the security design shall be considered without a thorough determination of the impact upon the future of the system; alternatives that are in concert with the planned architecture shall be given greater weight than permitting changes to the architecture. The contractor shall deliver to the government a System Security Architecture Document [DI-MISC-xxxx] which identifies all components of the system that contain security-relevant functions; this document shall thoroughly describe the integration of security features into the design of the hardware and software.

F2.3.7 Attachment II — Proposed Amendment of Data Item Description for the System/Segment Design Document

1. Amendment to DI-CMAN-80534 (S/SDD DID):
   a. 10.1.1-10.1.5.3.4 No change.
   b. 10.1.5.4. **System Architecture**.

Replace the current wording with the following: This paragraph shall be numbered 3.4 and shall be divided into subparagraphs to describe the internal structure of the system and the software architecture. The system segments, HWCIs and CSCIs, shall be identified and their purpose summarized. The system-level relationships among the segments, HWCIs, and CSCIs shall be described. Paragraph 3.4 shall also identify and state the purpose of each external interface of the system. A system architecture diagram may be used to illustrate the system top-level architecture. There shall be subparagraphs that describe the top-level software architecture structure, timing, and allocation attributes. One or more software architecture diagrams may be used to illustrate the different software attributes that comprise software architecture.

10.1.5.4.1 **Software Structure**. This paragraph shall be numbered 3.4.1 and shall describe the major software entities that will be developed or integrated into the system by the software designers, the mechanisms for flow of data among the major software entities, and the mechanisms for flow of control among the major software entities. Software entities should be the major software entities that will be recognized and manipulated by the system developers as the natural partitions within the software. In this context, component does not specifically refer to Computer Software Components (CSCs). Components may be partitioned by algorithmic functions, objects, and reusable components such as operating systems and database systems, or by any other scheme appropriate to the proposer’s development methodology. The entities defining the software structure should reveal the underlying structure of the software rather than hardware allocation or administrative and management controls that are often used as the criteria for defining CSCIs.

This paragraph shall also describe the mechanisms for managing the flow of data through the components. It should show the major data paths between and among transformation processes as well as to/from data storage. Where the flow of data is controlled by table-driven design, the data structures and their rules for modifying data flows should be described. It should define data structure and how file management and database management structures will be used.

This paragraph shall also describe the mechanisms for managing the flow of control among the components. It should show the major execution sequences, where execution sequences may be asynchronous or parallel, and how synchronization is managed. Where execution control uses data-driven design, the mechanisms for accessing the table and managing execution shall be described. It should also show how anomalous conditions such as error handling and exception conditions that dynamically alter the flow of execution are managed.

Timing. This paragraph shall be numbered 3.4.2 and shall describe the presence of critical timing and throughput processes. The software architectural devices used to manage critical timing events, interrupts, throughput load balancing, and data buffering shall be described.

Interconnection layers, standards and protocols. This paragraph shall be numbered 3.4.3 and shall describe the software architecture rules for interconnecting layered software entities. It shall describe allowable interconnections among layers, and identify the use of standardized interface protocols or bindings such as SQL, GOSIP, Motif, etc.

10.1.5.5 **Operational Scenarios**. Replace the current wording with the following: This paragraph shall be numbered 3.5 and shall describe each operational scenario of the system. For each system state and mode, this paragraph shall identify the hardware and software entities that execute and the manual operations to be performed. A table may be provided to illustrate the states and modes in which each hardware and software entity executes and each manual operation is performed. In addition, this paragraph shall describe the general flow of both execution control and data between hardware and software entities while operating in the different states and modes. Flow diagrams may be used to illustrate execution control and data flow in each state and mode.

10.1.5.6 **Software Architecture Rationale**. This paragraph shall be numbered 3.6 and shall describe the selection criteria and constraints that drove the choice of software architecture. It shall include cross-references to requirements in the SSS, the vision statement, or derived requirements that govern the choice of software architecture.

F2.3.8 Attachment III — Proposed Amendment of Data Item Description for the Software Development Plan

1. Amendment of DI-MCCR-80030A (SDP DID)

    a. 10.1 — 10.2.5.11 No change.
    b. 10.2.5.12 **Software Architecture**.

This paragraph shall be numbered 3.12 and shall describe the development management provisions for ensuring the software architecture is preserved throughout the development life cycle. It shall describe the contractor's procedures and methods for establishing an architecture definition, ensuring software developers understand the architecture, and ensuring software developers follow design guidelines that enforce the preservation of the architecture.

    c. 10.2.6 — 10.2.6.2 No change.
    d. 10.2.6.2.1 **Software Development Techniques and Methodologies**.

Replace the current wording with the following: This subparagraph shall be numbered 4.2.1 and shall identify and describe the techniques and methodologies the contractor plans to use to perform:

- Software Requirements Analysis
- Software Architecture Analysis and Definition
- Preliminary Design
- Detailed Design

- Coding and Computer Software Unit Testing
- CSC Integration and Testing
- CSCI Testing

e. 10.2.6.2.2 — 10.2.12 No change.

F2.3.9 Attachment IV — Instructions for Proposal Preparation

The following material is provided for insertion into the IFPP to elicit information needed by the government in the area of software architecture. These words are general in nature and must be modified to match the explicit needs of each program and to be consistent with the remainder of the IFPP and the selection basis (see Section M). The information given below is intended to supplement the information that would conventionally be in the IFPP. It may be convenient to merge the following material into the other information contained in the IFPP. In particular, offerors should not be required to submit additional copies or mere reformulations of information requested by other portions of the IFPP. Offerors should also be required to provide an index that would indicate the location of architecture-related information throughout the proposal.

F2.3.9.1 x.x.x System Engineering

Describe the overall architecture of the system by indicating the partitioning of the system into major hardware and software components. Describe systems with which the contractor is familiar that have similar characteristics and/or are based on similar components. Explain the manner by which information is to be passed among components and the control of execution. Describe how the architecture selected satisfies the requirements in the SSS and accommodates the vision statement. Describe the system services that provide for communication (both control and data) among software processes and tasks, hardware processors, external systems and devices, and the USI. Any other information requested in the DID for the S/SDD, or from other sources, should be presented.

F2.3.9.2 x.x.x Software Engineering

Describe the standards to be followed in the software design and identify any conflicts with the proposed architecture caused by use of these standards. Identify the tools to be used in generating the software. Describe how inadvertent and deliberate departures from the proposed architecture are recognized, reported, and/or prevented. Identify any software components previously developed (COTS or other NDI) and how they may have to be adapted to conform to the proposed software architecture. Describe how and when the execution sequence is determined; include the handling of interrupts, errors, out-of-normal situations, and response to timing constraints. Discuss how the execution sequence can be redefined.

F2.3.9.3 x.x.x Database Engineering

Describe the proposed database architecture with particular emphasis on those attributes that contribute to or detract from the proposed software architecture. Identify any tools to be employed in developing the database design. Identify any available data management packages (COTS or other NDI) and whether the use of these packages affects the proposed software architecture.

F2.3.9.4 x.x.x User Interface Engineering

Describe the proposed USI and associated methods for data display, data entry, sequence control, and user assistance within the architectural framework proposed for the system. Identify any rapid prototyping tools that will be used to support the User Interface Demonstration and how these tools will be used in a manner attuned to the proposed architecture. Describe how any COTS/or other NDI software will provide the USI while preserving the proposed architecture. Describe how the results of USI demonstration activities will be incorporated into the design, development, and test of the system.

F2.3.9.5 x.x.x Security Engineering

Describe the proposed security design and the allocation of functions to hardware and software components.

F2.3.9.6 x.x.x Modeling and Prototyping

Describe how the architectural assumptions and conditions are to be imposed on the system models used. Indicate to what extent the performance models will be calibrated against other systems using the same (or similar) architectures. Describe how (and when during the design activities) the architectural model will be subjected to execution and to validation.

F2.4 Glossary

CDR	Critical Design Review
CDRL	Contract Data Requirements List
CSC	Computer Software Components
CSCI	Computer Software Configuration Item
COTS	Commercial-off-the-Shelf
DI	Data Item
DID	Data Item Description
ECP	Engineering Change Proposal
GOSIP	Government Open Systems Interconnection Profile
HWCI	Hardware Configuration Item
IFPP	Instructions for Proposal Preparation
NDI	Non-Developmental Item
PDR	Preliminary Design Review
POSIX	Portable Operating System Interface for Computer Environments
RFP	Request for Proposal
SDD	Software Design Document
SDP	Software Development Plan
SDR	System Design Review
SRR	System Requirements Review
S/SDD	System/Segment Design Document
SSS	System Segment Specification
TIM	Technical Interchange Meeting
TRR	Test Readiness Review
USI	User-System Interface