

## **Appendix Q**

# **On Board Software for the Boeing 777**

---

## Content

<b>Q.1 Abstract</b> .....	Q-3
<b>Q.2 Introduction</b> .....	Q-3
<b>Q.3 Perspective for Success</b> .....	Q-4
Q.3.1 Size .....	Q-4
Q.3.2 Communication .....	Q-5
Q.3.3 Requirements .....	Q-5
Q.3.4 Functional Analysis .....	Q-6
<b>Q.4 The ICD Database</b> .....	Q-6
Q.4.1 Protocol Test .....	Q-7
<b>Q.5 Simulation and the Laboratories</b> .....	Q-7
Q.5.1 Software Metrics .....	Q-8
Q.5.2 COTS Software .....	Q-8
<b>Q.5 Summary</b> .....	Q-9
<b>Q.6 References</b> .....	Q-10

**Robert Lytz**

Boeing Commercial Airplane Group

---

## Q.1 Abstract

The 777 is the most software-intensive commercial airplane the Boeing Company has produced, and it was the most mature at entry into service. This paper describes critical factors in the success of the embedded software effort for the 777.

---

## Q.2 Introduction

In May of 1995, United Airlines accepted delivery, on schedule, of the first Boeing 777-200 airplane to enter passenger service. At that time, five 777s had completed 1,751 test flights totaling 3,379 hours, and 90 flights had been flown by United to simulate day-to-day flight maintenance and operations procedures. The delivery celebration occurred four and one-half years after Boeing had committed to 777 program go-ahead. The program goal of delivering a service-ready airplane on schedule was achieved, and it was done with a large amount of software developed during a relatively short time. About 2.5 million source lines of on-board code were newly written for the 777. This is six times the amount produced for the 747-400, the most recent Boeing Commercial Airplane program. On-board software for the 777 totals over four million lines of code when commercial off-the-shelf software (COTS) and software provided as options to customer airlines are included. The software was partitioned into more than one hundred major components corresponding to physical boxes (line-replaceable units, or LRUs) in the airplane's control system, and it was produced by various suppliers external to Boeing Commercial Airplane Group (BCAG), the Boeing operating group which designs, produces and markets commercial aircraft.

The Airplane Information Management System (AIMS), supplied by Honeywell, is a good example of use of software in the 777 for advanced functionality and integration of functionality. AIMS provides the following functions:

- primary pilot display
- flight management
- thrust management
- central maintenance computing
- airplane condition monitoring
- data communication management
- data conversion gateway
- flight data acquisition

Along with several other large systems, AIMS represents the “high end” of software in the 777. It is important to recognize, however, that virtually every system involved in controlling the airplane had a large software component.

---

## Q.3 Perspective for Success

In discussing 777 software-development, it is useful to examine some traditional problems in the embedded software business. A recent example, the 1994 Scientific American article “Software’s Chronic Crisis” [1], noted that “some three quarters of all large systems are operating failures that either do not function as intended or are not used at all.” This situation has historic roots. A quarter-century ago a NATO conference of software experts [2] included the following in a list of severe software-development impediments:

- Poor communication among groups working on the same project
- Lack of understanding in system requirements on the part of customers and designers
- Difficulty in monitoring progress in a software project, since program construction is not always a simple progression in which each act of assembly represents a distinct forward step
- Lack of inventories of reusable software components
- Rapid growth in size of software systems

In the 777 program, solutions to these software problems were worked by Boeing, its customers, and its suppliers. These solutions — in communication, understanding of requirements, progress tracking, reuse, and size — are a useful framework for understanding software development for the 777. In 1987, Fred Brooks [3] advised the software community not to look for “silver bullets” with which to slay the “monsters” that afflict software development efforts. In the 777 program, several tools, techniques, and facilities were crucial to delivering a mature product on time. These should be seen as viable means of accomplishing established software objectives, rather than as magic bullets.

---

### Q.3.1 Size

In his Silver Bullet article, Brooks also emphasized that many classic problems of software development stem from nonlinear increases of complexity with software size. This means that if one system has twice as much software as another, it is likely to be more than twice as complex, require more than twice as much labor to develop, and take more than twice as long to build.

One standard engineering solution to inefficiencies of scale is to partition a system into well-defined pieces with well-defined, simple interfaces among them. This was done for the 777, allowing the pieces to be built independently and simultaneously. However, reduction in complexity and inefficiency with this approach, and ultimate success when the pieces are put together, are not guaranteed. They depend on the quality (stability, coherence, and completeness) of developers’ understanding of the functionality of the pieces and their interfaces. The need to partition a large system is so obvious that it usually goes unremarked. Yet the difficulties associated with partitioning — incoherent functionality on integration, unstable or poorly communicated interfaces — are so common that partitioning often seems to be more trouble than it is worth. Most of the measures taken in the 777 program to produce successful software were intended to take advantage of the benefits of partitioning a large software system and to avoid the pitfalls associated with doing so.

---

## Q.3.2 Communication

“Working Together” was the slogan of the 777 Program. At the 777 All Operators Flight Operations symposium in October 1995, United Airlines fleet captain Lew Kosich stated, “I’ve never seen a program go like this. Working together was absolutely awesome. When the airplane was finally inaugurated into service June 7, it was almost a non-event for us. We were inaugurating a mature airplane.” The reality of this working-together approach in 777 development is evident in listening to anyone who helped specify, design, or build the airplane. Pilots, engineers, and maintenance personnel from customer airlines worked with their BCAG counterparts from the start. Suppliers, including software suppliers, were involved earlier and more completely in the overall process than for previous BCAG airplanes. For example, Boeing made protocol-test and simulation tools (described below) available to suppliers so they could begin verifying message interfaces early in the program.

It is hard to quantify day-to-day causes and effects in the working-together policy. However, its utility needs to be taken seriously. Most participants in the 777 program believe the working-together approach was an absolute prerequisite for success. The rest of the approaches, techniques, and tools described here would have been less valuable if they had been applied in a less collaborative environment.

---

## Q.3.3 Requirements

At the outset, program management set challenging “service-ready” objectives which drove the 777 program. These objectives, met at delivery, contributed to early refinement and stability of requirements. They were:

- The airplane and its systems are mature at first delivery.
- Airplane support programs are complete and available at first delivery.

Achieving these objectives meant that many development and test activities occurred earlier in the 777 program than they had in previous ones. For individual LRUs, there was emphasis on having all stand-alone testing performed at supplier sites prior to beginning any testing at Boeing. Similarly, detailed system-integration testing was begun as early as feasible in the program. For every activity that was moved forward (compared with previous development efforts), there was increased opportunity to catch defects, particularly defects in requirements. It has been well established for twenty years that the earlier in development software defects can be caught, the less expensive it will be to correct them [4]. Emphasis on maturity at delivery contributed to increased maturity at each point in development, and *vice versa*.

Several top-level documents contributed to the cohesion of the software effort. The 777 Design Requirements and Objectives (DR&O) document was the single point from which all other airplane requirements, including software requirements, were ultimately derived. For each system, a System Requirements and Objectives (SR&O) document, setting forth high-level requirements, was prepared, based on the DR&O. Each SR&O, in turn, was the basis for the Specification Control Drawing (SCD), the definitive requirements document for each system. Requirements flow-down was managed systematically with appropriate technical detail at each level, resulting in a secure basis from which to build software.

The General Technical Requirements (GTR) document was new for the 777. This single source provided generic material that would otherwise have been included redundantly in each SCD. Pulling the GTR together and referencing it from each SCD not only saved the work of restating general design considerations; more importantly, the GTR promoted uniform understanding of those considerations.

---

### Q.3.4 Functional Analysis

A concerted requirements-validation activity contributed to timeliness, stability, and coherence of requirements, making software re-work less likely. Validation analyses for the 777 documented the equipment-operability conditions under which the airplane would, or would not, be allowed to be dispatched, and they provided assurance that safety and redundancy features would accomplish their intended function if called upon to do so. In addition to the prime intent of satisfying safety requirements, 777 validation activity ensured that documentation needed for customers would be available immediately on first delivery. The significance for software was that functionality and interface issues were worked thoroughly.

The system functional analysis for validation proceeded in three phases. In the first phase, the functionality expected for each individual system was examined for intra-system effects of single and multiple-point failures. This phase was concluded before program go-ahead, and its results were folded into the SCD for each system. In the second phase, the effects of single and multiple-point failures were examined across system boundaries. In the third phase, similar analyses were done top-down from the airplane point of view. In each phase, discrepancies were formally registered and worked, and SCDs were updated as a result. In addition, the analysis scenarios used were incorporated into laboratory integration tests and flight tests where feasible. The resulting maturity in definition of functionality and interfaces materially affected software development and quality.

---

## Q.4 The ICD Database

For the 777, the overall bus architecture and its interfaces were designed to provide appropriate connectivity, isolation, redundancy, and bandwidth across the airplane and its systems. The buses involved were four ARINC 629 systems data buses, three ARINC 629 flight control data buses, and numerous ARINC 429 buses. The ARINC 629 systems data buses redundantly provide high-bandwidth broadcast messages and directed messages among LRUs over the entire airplane. The ARINC 629 flight control buses provide similar communication facilities for the redundant flight computers, actuator electronics, and associated power supplies and sensors. ARINC 629 buses were first used in the 777. ARINC 429 buses, an older technology, provide point-to-point connections — typically between LRUs that are derivatives from previous airplanes.

Consistency of expectations between the transmitter and ultimate users of each message is a pressing issue for any program where computers must exchange data. For the 777, the issue was amplified by the number and variety of messages, sources, and destinations. The 777 development program managed this issue systematically and with a whole-airplane perspective. This was done using procedures and tools associated with the Interface Control Drawing (ICD) Database.

From the beginning of the program, the 777 ICD Database documented bus messages within the 777 computer architecture, as well as most discrete-level signals. In all, the 777 ICD Database describes 3000 analog signals and 40,000 digital signals. There are several factors that account for success of the ICD database in dealing with the variety of 777 message and signal traffic:

1. Prior to program go-ahead, the ICD Database was the single, authorized repository of 777 bus-message information.
2. Use of the ICD Database was integrated into the 777 development process.

3. Tools associated with the ICD Database detected and reported discrepancies between sources and users of communication data.
4. The ICD Database automatically generated required interface documents.
5. The ICD Database directly fed protocol-test and simulation software.

Interface descriptions were put into the ICD Database as they became available. By November 1992, a tool to check for discrepancies between transmitters and receivers of information was placed on-line. Initial use of this tool confirmed the need to have both the transmitter and the receiver(s) of any given message document their expectations independently. Within two months of initial deployment of the tool, 50,000 instances of discrepancies between transmitters and receivers had been detected. The discrepancies were worked intensively. As a result, their number decreased to 3000 in two more months. In addition to the discrepancy checker, other tools were built to take advantage of the ICD database. One was a documentation generator that pulled printed message tables from the ICD database for inclusion in required documentation. As described below, other tools pulled message information from the database to support emulation and simulation directly. Overall, use of a single, on-line source of interface descriptions promoted stability of on-board software during its development and test.

---

## Q.4.1 Protocol Test

Message extracts from the ICD database directly fed the Protocol Interface Test Tool (PITT), which was written by BCAG to test two critical interfaces to each LRU with connections to the newly-developed ARINC 629 bus. These interfaces are the Central Maintenance Computing Function, which monitors the health of each LRU, and the Data Load Gateway Function, which loads on-board software and software-configuration data into each LRU. PITT provided facilities to test nominal and off-nominal message transmission associated with these functions. The tool was based on the new message protocol for ARINC 629 buses. Suppliers used PITT at their sites for about three-quarters of the software LRUs in the 777, and BCAG used it in verification of all software delivered. In addition to verifying protocol for individual LRUs, the PITT identified instances where protocol definition needed to be upgraded. This activity, like the others described here, contributed to the maturity and stability of the product and development environment.

---

## Q.5 Simulation and the Laboratories

Laboratory simulation, used on a scale unprecedented at BCAG, was crucial to delivering the 777 on time. Ground-based integration and integration-testing of systems have traditionally been accomplished on actual airplanes at BCAG. At the beginning of the 777 program, a corporate decision was made to commit resources to BCAG laboratories that would allow integration testing of the entire control system of the 777 and future airplanes. Laboratory work was expected to be, and proved to be, essential to the fundamental goal of producing a truly service-ready airplane. Compared with ground-testing on prototype airplanes, laboratory testing yielded improved access to components and enhanced ability to instrument the software and hardware. There was a great deal of flexibility in configuring and scheduling various arrangements of airplane equipment, test equipment, on-board software, and simulation software. In the laboratory environment, planned stages of testing were accomplished sooner and in a greater variety of configurations than they could have been otherwise.

Strong emphasis on providing a good simulation environment augmented the attention BCAG and its suppliers gave to requirements and interfaces. Developers of simulation software were an additional group of users of requirements for 777 functionality and interfaces. Thus, in addition to meeting its primary goal of testing on-board software and equipment thoroughly, the laboratory simulation effort provided one more source of early requirements analysis, making the software development process more stable.

The simulation environment was large, and the simulation work could not have been accomplished without automatic code generation facilities. In all, about 3.4 million lines of code were developed at BCAG to support laboratory simulation — more than the total for new on-board software for the airplane. About 60% of the simulation software was autocoded, including all 1.4 million lines of interface software, which were generated directly from the ICD database.

---

### Q.5.1 Software Metrics

Uniform use of software metrics was instituted about half-way through the 777 development program. From the beginning of the program, supplier software engineers and their BCAG counterparts used various software measures to communicate status of the work at hand. Due to the diversity of these measures, it was difficult to get a comprehensive view of the overall state of software development. Late in 1992, suppliers were asked to report simple, standard software metrics, and they began doing so for most systems within a few months. The metrics used [5], included plans and actuals for software design, coding, test definition, test execution, and resource utilization. In addition, actuals were collected for software problem report totals.

Simple as the uniform metrics were, they provided an excellent medium for communication and an impetus for more detailed understanding of plans and status at all engineering and management levels. From the beginning of their application, uniform metrics encouraged application of reasonableness checks on plans and discussion of those checks. Over time, there was variation among systems in their degree of adherence of actuals to plans. Where actuals and plans diverged, there was opportunity for BCAG and its suppliers to examine causes and solutions or work-arounds for problems underlying the metric data. The conversations between Boeing and suppliers that were prompted by metric data were as important as the metric data itself.

Analysis of software-metric data at the conclusion of the development program yielded useful lessons for future programs. In seeking to identify the most potent early indicators of final schedule behavior, there was a serendipitous result. Both the time it took suppliers to begin reporting metrics coherently and the degree of adherence to early software-testing schedules were found to be strongly correlated to final software-test schedule slides and with the date when systems were ready to pass from engineering-development control to manufacturing control.

---

### Q.5.2 COTS Software

Commercial off-the-shelf software (COTS) was used to a greater extent in the 777 program than in previous Boeing commercial aircraft. Two examples of COTS are the Ada run-time-systems whose code became embedded in the software for many LRUs and the DOS / Windows environment. In both cases, substantial economic benefits were realized.

For some systems, such as Cabin Entertainment, products like DOS, Windows, and X-windows were used. These had the enormous benefit of not having to re-invent functionality which was already available commercially. For some other systems where desired functionality was available in COTS (for example, X-Windows), the off-the-shelf products were not used. This is because, in order to meet certification for the level of safety required, the cost of developing appropriate tests to perform upon COTS would have been greater than the cost to develop the functionality directly. One consideration here is that testing COTS functionality, which often exceeds the functionality that a particular application needs, can be more expensive than testing only the functionality that is required. Also, the Ada run-time system represented a substantial amount of “off-the-shelf,” reused code for the 777, even though it is often not thought of in this way. Ada was used, with restrictions depending on system criticality, for over half of the LRUs in the 777.

COTS will probably become even more accessible to future airplane development efforts. This is because interfaces and version interoperability are becoming better defined and standardized, because COTS software is being tested more rigorously as time goes on, and because generic tests one might perform on COTS products are themselves becoming available as COTS.

---

## Q.5 Summary

In software development for the 777, new techniques, tools, and facilities were crucial to program success. Rather than seeing any of these as “silver bullets,” it is probably most useful to regard them as effective ways to achieve well-established, software-development objectives. In the 777 program, there was effective communication. Requirements for functionality and interfaces were developed early, they were systematically refined through the program, and they were carried through to final integration, laboratory test, and flight test. Simple, consistent measures of software progress were used. Commercially-available software was incorporated where feasible. Overall, the system and the development effort were partitioned and managed to reduce complexity, to allow work on components to proceed independently and “in forward steps,” and to ensure successful delivery of a mature product.

---

## Q.6 References

- [1] Gibbs, W. W., “Software’s Chronic Crisis,” *Scientific American*, 271, 86-95, (September, 1994)
- [2] Naur, P. and B. Randell (eds.), *Software Engineering: Report on a Conference Sponsored by the NATO Science Committee*, Brussels, Scientific Affairs Division, NATO (January 1969), as quoted in: Cusumano, M. A., *Japan’s Software Factories*, Oxford University Press (1991)
- [3] Brooks, F. J., Jr., “No Silver Bullet,” *IEEE Computer*, 20, 10-19 (April, 1987)
- [4] Boehm, B. W., *Software Engineering Economics*, 39 (1981)
- [5] Lytz, R., Software Metrics for the Boeing 777: A Case Study, *Software Quality Journal*, 4, 1-13 (1995)