WHY (SOME) LARGE COMPUTER PROJECTS FAIL

by Robert N. Britcher

The FAA's Advanced Automation System (AAS) began in 1981, at about the time President Reagan dismissed 11,400 striking air traffic controllers. The timing is not incidental.

President Reagan, long awed by technology (Buck Rogers had been one of his adolescent heroes), believed that, inevitably, computers would relieve mankind of the brute labor that has kept it from pursuing its dreams. Automation, he often said, would ease our lives, even solve our social problems. It would certainly protect us. He dreamed of a system of heavenly lasers that would defend us forever from nuclear attack. Within months after taking office, he authorized it: the Strategic Defense Initiative, the project everyone soon called "Star Wars." While that project never left the ground (although millions have been spent on the drawing board), the Advanced Automation System took off.

In 1981, at the dawn of airline deregulation, about 300 million passengers were taking commercial flights in the United States. It was predicted—accurately—that the number would grow to over 500 million by 1994. Today, there are over 20,000 scheduled airline flights a day; the number of passengers expected in 1997 is 546 million. By the year 2007, over 2 million passengers a day will be in flight. The number of jets in the U.S. fleet is expected to grow from 4,775 in 1996 to over 7,000 in 2007. Add in general aviation (over 180,000 planes) and military flights and you have busy skies.

On the ground, the air traffic control facilities are (and have been for some time) stressed out. The New York Tracon alone supports 7,000 flights a day. Tracons control air traffic around the terminal areas within a 60 to 90 mile radius of airports. There are over 180 such facilities, hundreds more towers, where planes are metered between the runways and the sky, and 21 en route centers. These are staffed by 17,000 controllers, about the same number today as in 1981.

On a national scale, the computer was introduced to air traffic control in 1963. The Tracons were the first to be automated. The Univac

This article was written especially for this [illegible]

computers and computer programs converted analog radar signals to digital form, and presented to the controller an annotated picture of planes in airspace. The en route system followed, but it took longer to complete, largely because the en route system must process radar data and, for each flight, predict the time of arrival at each fix along its route. The present position of an aircraft is of interest, but so is its future. The second requirement takes advantage of the computer's facility as a calculator, but it also makes the system more complex. The en route system has to juggle real-time sensor data with the management of a persistent (and, as designed, labyrinthine) data base. Tower operations are largely manual. Control is still a matter of vision and voice. The large towers receive data from the computers in the Tracons and en route centers. In the 1970s, the FAA used the computerized data from the regional facilities to seed another function: traffic management, or flow control, so that the density of our domestic airspace can be summarized and trends evaluated.

The role of automation in air traffic control is modest. (With the AAS, that was to change.) This fits well with the often immediate and habitual nature of ground supervision of aircraft. In no job is immense responsibility and habit so entwined—more than in police work or neurosurgery. Cognitive scientists would have a field day with air traffic controllers. They must perceive the (computer-modeled) relationship among aircraft, racing along at over 300 miles per hour, while they simultaneously interact with a computer and talk to the pilots. The interaction of perception, judgment, and response appears to be almost autonomic. For good reason, their job is always listed at the top of the most stressful professions. And professionals they are. Controllers work 6-day weeks. In spite of the mandatory two breaks a day, some work what is called the "Quick Turn": 3 in the afternoon to 11, followed by a 7 A.M. to 3 P.M. shift. These are the young tough guys who hug those planes and passengers to their bosoms. Others, often burned out, will take only light loads. But no one works the boards for long.

The urge to replace the controllers with technology and the alarming projections of the growth of air travel were not the only motives behind the AAS. There was a more practical issue. The equipment was (is) getting old: radars, navigational aids, computers, cables,

telephones, the whole gamut. The favorite word is "antiquated." Even the software was old, some said, who do not realize that old software is more reliable than new software.

But, in the end, the drivers were financial. The FAA has been under pressure from the airlines to upgrade the computer algorithms to permit "preferred" routes of flight, which could save each airline millions of dollars in fuel. According to the major carriers, today's rigid procedures, based on fixed airways, derived from a decades-old topology of ground navigational equipment, cause over 20,000 delays every day, and waste $5 billion a year. The advent of satellite navigation makes off-airway flying potentially reliable—and profitable.

The other economic issue is related to technology, specifically the unique property of software to be adapted to many environments but it is more directly aimed at downsizing. (The two, technology and the displacement of jobs, have been cohorts for centuries.) The entire system of terminal and en route facilities, save the towers, would be consolidated. The existing facilities would be combined into some 30 or 40 area control facilities, most, if not all, in new locations. Shutting down over a hundred Tracons and en route facilities—which meant uprooting and laying off people, renovating jobs, phone lines, security operations, you name it—would save an estimated $4.5 billion in operating costs.

In summary, it was believed that the successful implementation of the Advanced Automation System would mean fewer people, less fuel, lower costs, cheaper flights, fewer delays—and (satisfying our deep American drive for technology) more automation.

Change is inevitable. But the world of air traffic control is as vulnerable to change as it gets. Change, if and when it comes, must be as imperceptible as possible. A slight shift in the position of a knob can distract a controller and create what the FAA calls an incident—when planes get too close to one another or to objects. The biggest problem in creating a new version of an old system is the transition. The air traffic control system operates all day, every day, and any change must be made while the system is running. The FAA had no clear concept of how this would be done. They set up the mission of the AAS to make transition, not just awkward, but impossible. Contrary to the sensitive character of the air traffic control system, and

especially the people who support it, the new system would be revolutionary, as radical a departure from well-worn mores and customs as the overthrow of the czars. Someone must have suspected as much, for there were advertisements to the contrary. AAS was to be a model for the evolution of systems. In fact, there was one incremental phase, in which the en route controllers would get new displays. This was evolution feigned. In the main, the AAS was designed to sweep away the old and replace it with the new. If there was a debate, the technologists prevailed. A subtle and fragile culture would be transformed. By 1998—the planned end date for the project—the modus operandi of air traffic controllers, maintenance engineers, computer operators, guards, secretaries, and supervisors would become a rumor, the stuff of anecdotes. Where they had been and what they had done would be obliterated.

The Mission

The Advanced Automation System began, in concept, in 1981 and ended in 1994, "terminated for convenience" by the government. Billions of dollars were spent on it. It is hard to describe. You can't learn anything from the name. You know it's about air traffic control because I told you, or because you read about it in the papers. Maybe part of the problem was the name. It sounds like the system to end all systems.

One engineer I know described the AAS this way. You're living in a modest house and you see the refrigerator going. The ice sometimes melts, and the door isn't flush, and the repairman comes out, it seems, once a month. And now you notice it's bulky and doesn't save energy, and you've seen those new ones at Sears. So it's time. The first thing you do is look into some land a couple of states over and think about a new house. Then you get I. M. Pei and some of the great architects and hold a design run-off. This takes awhile, so you have to put up with the fridge, which is now making a buzzing noise that keeps you awake at night. You look at several plans and even build a prototype or two. Time goes on and you finally choose a design. There is a big bash before building starts. Then you build. And build. The celebrating continues; each brick thrills. Then you change your mind. You really wanted a Japanese house with redwood floors and a formal garden. So you start to re-engineer what you have. Move a few bricks and some sod. Finally, you have something that looks pretty good. Then, one night, you go to bed and notice the buzzing in the refrigerator is gone. Something's wrong. The silence keeps you awake. You've spent too much money! You don't really want to move! And now you find out the kids don't like the new house. In fact, your daughter says "I hate it." So you cut your losses. Fifteen years and a few billion dollars later, the old refrigerator is still running. Somehow.

On the Advanced Automation System, everything was to change. Each air traffic controller would have his own workstation. He or she could operate an ensemble of computers and communications channels, software and peripheral devices, all hidden away inside the cabinetry. The workstation was the show piece of the new system. A 20 x 20-inch Sony display brought high-resolution graphics to air traffic control. Each controller could tune the splendid graphics to suit his or her situation and preference.

The original Tracon and en route computers would be replaced. There would be the new algorithms. A new digital voice system was coming along. There would be the latest in fault-tolerant computing, algorithms to protect the air traffic control algorithms, so the system would (almost) never fail. (The system could fail only three seconds a year, according to Mitre's analysis.) Although this was the most discussed requirement (later both Mitre and the FAA admitted it was not achievable), there were many extreme requirements, any one of which could undermine the successful implementation of a digital system. For example: recording video by sending computer files over a local area network because VCRs could not handle the huge Sony displays; replacing paper terrain maps with electronic renderings whose geographical markings could not be accurately certified; cutting over from one version of software to another while the system is running; distributing the processing of a single flight over dozens of computers whose synchronization could be undone by a single fault; replacing paper clearances ("flight UAL22 is cleared to altitude . . .") with electronic notes, coupled with the complete removal of printers (the FAA was zealous about having a paperless system, this system designed upon glaciers of paper. Deprived of a

means to old habits, the air traffic controllers would have no choice but to adopt new ones.); a training system that was to simulate the entire system within one workstation; the capability for an entire air traffic control center to backup any other or all centers (this would have required super-computers not yet imagined, embedded in each workstation). Ada—the FAA fell in with the Department of Defense in mandating this "final and definitive language" for programming—was a requirement, especially the spirit of Ada, bearing upon its back ferocious government supervision.

The Project

The project was conducted in two phases: a design competition, which ran from 1983 to 1988, pitted IBM against Hughes; and the big prize, the acquisition phase. The design run-off included prototyping and the development of detailed plans and specifications, very few of which made it through the acquisition phase. In the design phase, years before programs would be written, years before a contractor was chosen, specifications for computer programs were written to the bit level. Systems engineers worked hundred-hour weeks to develop thousand-page specifications for programs whose loops would begin to unfold in about seven years. (By then, the original specifications had been rewritten many times.) The software design was also to be completed in the design phase. It was believed that writing the design in Ada would make coding easier. But, the programmers, unschooled in the use of abstractions, delivered what amounted to code in the design phase. This was even touted by IBM, which proudly proclaimed that it had written a million lines of Ada design. Virtually none of it survived.

The FAA later admitted that the design competition, which cost the taxpayers $1 billion, was of little value. Nevertheless, it was an opportunity for the private sector to advance. Hughes and IBM brought on their experts, people who, in the prime of their careers, had helped put a man on the moon, had built weapons systems, had been heroes of another age in air traffic control automation. This was to be their greatest accomplishment, an accomplishment they would share with the dozens of subcontractors the FAA hired to watch

them. Throughout the ranks, from top to bottom, in both the private and public sector, this would be their finest hour.

Things began to go wrong almost immediately. By 1986, it had become clear that the consolidation of facilities was never going to happen. It is hard to believe that anyone ever thought it would. Yet, the FAA pressed on. Just before the start of the acquisition phase, they decoupled the Tracon and the en route systems. But by then, both contractors had conceived a design that embraced both. The mission altered, the technical approach went forward as if it had not.

The outcome was decided in 1988. Hughes—having taken on many of the characteristics of its founder, Howard Hughes, who brought a Hollywood approach to the aerospace business—predictably, outscored IBM's proposal. But IBM won the job, because it was cheaper and because of IBM's long-term collaboration with the FAA. (IBM had built the en route automation system.) Hughes' protest was denied.

At $3.7 billion, the Advanced Automation System was one of the largest civilian computer contracts ever; maybe the largest. It was the largest single contract in IBM's history. From the moment it was awarded, until near the project's end, IBM patted itself on the back. The herculean efforts of thousands of people in the design competition phase fueled a years-long celebration. There was something for everybody, beginning with a great ball in Union Station, featuring Chubby Checker and "The Twist."

At its peak the project employed over 2000 people. About a million dollars a day. If you thought like the IBM project manager, this was a good deal. Many people were working and money was being made. It was going to last forever. No one considered that it wouldn't. And everyone was getting ahead. (One of the ironies of conceptual work is that it is easy to believe you are farther along than you are; only symbols are being produced.) How could it end?

The AAS project lasted longer than the siege of Troy—by four years—and the story, as at Troy, lay in the effort itself. If you want to get a feel for how it was, you can read the *Iliad*. There, for example, you can learn in Book II:

From the camp,
the troops were turning out now, thick as bees
that issue from some crevice in a rock face,
endlessly pouring forth, to make some cluster
and swarm on blooms of summer here and there,
glinting and droning, busy in bright air.
Like bees innumerable from ships and huts
down the deep foreshore streamed those regiments
toward the assembly ground—and Rumor blazed
among them like a crier sent from Zeus.

rom "Bugs in the Program," 8/3/89

A Report to the 101st Congress on problems in federal government computer software development and regulation, submitted by the Subcommittee on Investigations and Oversight.)

Dear Mr. Chairman:

The Comptroller General reported in 1986 that the FAA Advanced Automation System—the foundation of the nation's ability to manage future air traffic—was being developed with an approach that does not adequately mitigate technical risks . . . The program has already encountered significant cost growth (at least 50 percent in the design competition phase).

"What may finally force a redefinition of this procurement system is continued failure to balance the budget. The strange policy where the Government pays twice for a system—once to buy it and again to make it work . . . cannot be sustained in an era of multibillion dollar shortfalls in the Treasury.

"Behind every successful software system, be it air traffic control or medical diagnosis, is a software manager. Like software itself, these managers must be developed.

"Adaptive intelligence in safety-critical systems may be beyond the capability of regulatory agencies to certify.

"Nothing recommended in this report will have any effect on the software problem."

The AAS must have been the most supervised project in history: this atop its enormous size and complexity, and the extreme and constantly changing requirements. One programmer described it this way: "Working on the project was like working on a car inside the garage with the motor running. Eventually, even the crickets hopping around the tires suffocate." I wonder now if we would have finished the original en route system today, with another thirty years of government rules heaped on? Surely, one thing that helped us then was our ignorance.

What I saw on the FAA's Advanced Automation System would make Sisyphus weep. Bullfinch does a nice job of describing his life in the "infernal regions": "his task was to roll a huge stone up to a hill-top, but when the steep was well-nigh gained, the rock, repulsed by some sudden force, rushed headlong down to the plain. Again, he toiled at it, while the sweat bathed all his weary limbs, but all to no effect."

Whatever commitment and discipline there was—and I believe all involved, both in the FAA and in the private sector, were deeply committed—was worn down by a battery of watchfulness that I can only ascribe to a fear of failure. In spite of the tens of millions of dollars spent on new computers for AAS, the most important piece of equipment on the project was the overhead projector. There were endless meetings attended by dozens of people—as if we were never quite sure about the whole thing. The people in charge simply lacked the confidence and the finesse of the space team: NASA, contractors, and astronauts.

The AAS was the intellectual equivalent of building a bridge over the Atlantic Ocean—all at once. The programmers knew better: they were building five bridges over the Atlantic with tools never before used in bridge-building.

If you could sit down for an hour and imagine everything you might want to know about a system and its programs, in advance of their production, and write it all down in a contract, then add it to the lists of a hundred of your friends who were asked to do the same thing, you would fall short of the requirements for workmanship and documentation levied on the AAS. For every line of software, a hundred pages were written about it. On such projects, hardly anyone programs. One test plan I saw ran to 800 pages to test five re-

quirements for gathering online performance data. And, when it came to something as important as system availability, the documentation scaled up. None of this was done once. It was done many times. Scores of people were paid to review and critique the work, which they did with glee. Every decision resulted in more work. Nothing was replaced.

from Government Computer News, 8/21/89

FAA MANAGER BLAMES IBM'S ADA TOOLS

"The FAA program manager, Michael E. Perie, said last week that immature technology in the Ada Programming Support Environment supplied by IBM has delayed the AAS.

"IBM officials took issue with Perie's remarks: 'Our APSE does not impact the software development schedule.'

"The FAA also has had some 'requirements questions' regarding the contract, Perie said. 'And we're still wrestling with how big that problem is in terms of schedule and cost.'

"IBM officials disagreed: 'We're confident that if we encounter any early delays, that those would be overcome by progress made later.'

"Some of the software development problems 'are just classic,' Perie said."

from Datamation, 12/11/89

FAA TO ASK REVISION OF IBM

"Last summer, IBM stumbled on the third of seven builds. IBM has indicated plans to extend the fourth build from three months to six while adding one month to builds five through seven, Mullikin said."

"Build four, Mullikin said, also will be lengthened to allow IBM to rehost software from the (sic) its next-generation RISC processor.

"The original schedule did not allow for rehosting."

from Electronic News, 9/3/90

FAA PROJECT FACES MORE DELAYS

"The FAA's $3.5 billion air traffic control program, already 13 months behind schedule, will likely be further delayed another nine months, according to a recent report by the Government Accounting Office.

"'The 13-month extension does not consider the time required to resolve remaining requirements," the GAO report concludes.

"'Further, little time has been allotted for resolving problems that may arise from system testing.'"

Subject: In your mail . . .

"In your mail you will be receiving a copy of a letter from Riebau to Dennis Trippel. The letter expresses a concern that IBM is modifying PU10 without FAA approval, namely by changing STNs that are fulfilling PU10 DID requirements. The letter requests Trippel to notify IBM that the applicable STNs are frozen and that any changes have to be formally submitted to the FAA for review and approval—and that exceptions must be formally submitted via the deviation and waiver process. I will be working with my representatives to determine what ramifications this may have on the mechanism that we already have in place for getting FAA approval for STN changes. I hope that whatever we work out will have little to no impact on our internal process of STN change."

"Have a nice day, Jenny"

In 1991, Harmon, a lead programmer, convinced his team to read their own source code aloud at inspections, not have it read by a disinterested party, as the procedures required. He wanted to reduce the overhead of inspections, which had become grossly over-attended and over-documented. Bloated, he called them. "They were interfering with the natural pace of programming."

Inspections have not changed much over the years. They can apply to any document: requirements, design, code, test plans, proce-

dures. They are administered by project and conducted by peers. Management does not attend meetings. This is a take-off on Heisenberg's Uncertainty Principle, wherein observing the experiment changes it. Besides, it has been suggested, the public discovery and tallying of errors is enough to discourage making them. A separate organization, quality assurance, often placed strategically within the company or the project, approves the procedures for conducting inspections, monitors their conduct, then collects, collates, and reports statistics to management.

Inspections are often referred to as formal inspections. They do resemble a liturgy. Invitations are issued. About three days in advance, prospective participants are told the time and place of the inspection meeting and given the material to be inspected. The meeting is presided over by a moderator, who attends indoctrination sessions to prepare for the role. A presenter, a sort of ad hoc deacon who cannot be the author, reads aloud from the exhibit, usually not line-by-line. The errors and questions are scrupulously recorded by the moderator, or another. The meeting lasts about two hours. Participants must itemize the amount of time spent preparing for the meeting, reading and inspecting the exhibit, and documenting their findings. This is considered important data, because the cost of inspections is high: fifty programming statements, written in three hours, may cost twenty to forty labor-hours to inspect.

But then Harmon. The streamlining paid off—at first. Programs in his area were getting done on schedule, and they worked. There were a number of reasons for this, but he attributed his team's success to leaner inspections. He was also an honest fellow, so he reported the new method to his quality assurance department, hoping to institutionalize it. Shortly thereafter they reported it to the FAA quality assurance officer. This was in keeping with the rules of quality assurance. The contractor may define his own procedures, but, having written them down, the contractor and the sponsor quality assurance departments collaborate to penalize any violations. There are no exceptions.

In about a month, the FAA wrote a letter to IBM insisting that IBM had violated its commitment to the very methods it had created. This could result in a reduction in fee. There were several meetings

involving the FAA's quality assurance officer (Riebau), and contracts officer (Trippel), IBM's software development manager, IBM's Director of Development, and scores of others. No one could agree. The matter was escalated to Mike Perie, the FAA's project manager for the Advanced Automation System, and Bill Carson, IBM's Vice-President for Air Traffic Control Systems. Here, things were clarified. The system development plan, delivered with the proposal, was considered part of the contract. And the hundreds of standards and procedures developed along the way, riders to the plan, were likewise contractually binding. Unhappily, IBM was in breach of the contract because a programmer had given voice to his own code.

After many more meetings, IBM apologized. Harmon was reprimanded, but legal actions were avoided.

A Technical Problem: The All-American State Machine

The smallness of computers and our penchant for giving everyone his own, created on the Advanced Automation System a technical problem that befit the grandness of the project.

Soon after—it seems now like moments after—the first VLSI chip rolled off the line and a few promising protocols for distributed processing had been published, the sponsors of large systems here and abroad were insisting their systems be distributed. The programmers who had just come off a decade of grinding it out on mainframes—making programs work on a single computer, albeit something fancy like a multiprocessor or an array processor—had some idea of what they were in for. But, experts were saying that distributed systems were safer, and more reliable. "The loss of one computer won't mean much; it's the family that's important," a friend said. No fool, he also said: "distributed processing appears in theory to make problems smaller, but it will make them bigger. The solution will wash over the problem with complexity."

One of the nice things about theory is you can stay close to home. In fact, it's hard to think deeply about one subject while you're musing about four or five others. So we invented polymers and didn't think too much about the effect polymerization would have on the health of chemists. The theory of distributed computing is one thing. But the theory of distributed work is another. That's where the

programming comes in. Moving the work to a machine, the machine, some machines, any machine, or all machines at the right time, is no small beer, especially in the presence of failures and heavy traffic. And then clocks have a way of drifting just like programmers' thoughts. Is it easier to send work from one machine to all machines than to some, or to just one? It depends. How important is the work? Must it arrive within a small window in time? Is it transient—Can the next message be sent again with negligible effect? Must the work be synchonized across machines?

Mathematicians speak of states. (A state is a value at a moment in time.) There can be a next state and a previous state, in fact, an entire history of states. There can be incorrect states, incoherent states, contingent states, even states of no consequence. These wonderfully adaptive states can apply to the values programs produce as well as to the programs themselves. A program can be in an incorrect state temporarily, but the values it has produced are OK. Or—and this is where work involves not just the machine but the human—the values may be temporarily incorrect, but, to the human, it does not matter.

There was no standardized software product to provide control services for distributed processing at the time of the AAS design. There was nothing available like IBM's OS/360—and its descendents—which, for the mainframe, insulate the application programmer from the particulars of storage regions, channels, multiprogramming and concurrency, disk drives, tapes, communications lines, and such. To implement large-scale, real-time applications on a local area network of cooperating processors, you had to write your own control program. On the AAS, such a program was written and debugged over a dozen years. (It is in service today, adapted to other air traffic control projects; but it is not standard, and it has never been marketed as a product.) This may have been the programmers' greatest achievement; but no one anticipated the time it would take to pull it off.

Part of the AAS control program was intended to solve a most absorbing problem: replacing programs with persistent states (the half-life of the flight data of the en route system can be measured in minutes and hours) while the programs are running. There are always

new versions coming along containing fixes or new features. Their insertion into the operational suite can be an intractable problem when no part of the human's work is done on paper, or when there is no completely independent digital backup system. It turns out, if the work can continue elsewhere, you can stop the machine for awhile and carefully reload, resynchronize, and reconcile new programs with old values. Otherwise, you must reconcile them while the programs are running. The latter was the case on the AAS.

No one could figure out how to do it. There were a thousand conversations about it, about how it was no big deal if you did this or that. But we simply couldn't do it. There was a backup system; but it shared programs with the primary. And the FAA insisted: no paper. "We want an All-American State Machine." The financial estimate to solve this problem was escalated to Congress. It got their attention; the ticket was in the hundreds of millions of dollars. But the fact is it could not have been done for a trillion dollars.

A Psychological Problem Dressed Up as a Technical Problem

It has been noted by everyone from the New York Times to the Vice-President of the United States that the main problem on the Advanced Automation System was "changing requirements." For those involved in large-scale computer systems, that is nothing new. No one can perfectly surmise the shape and feel of a system years in advance. Even replacing some aspect of a system you know by heart is not immune from thinking twice about it. It's in the conceptual nature of things, one. And two, we all—if we cannot explain it, at least—sense that software is a form of rapid specialization. That is, with lightning speed, we can change one "machine" into another, turn a lamp into a gate. We are no longer amazed by—in Herbert Simon's words—the "automated navigator."

The requirements churn (it was called) on the Advanced Automation System was not normal. It was the result of our enchantment with the computer-human interface, the CHI. The new controller workstation, fronted by a 20" by 20" color display, because it was capable of a seemingly endless variety of presentations, mesmerized the population of AAS like the O.J. Simpson trial mesmerized the nation. In simple arithmetic, each controller would operate a computer

filled with 64 megabytes of software just to display and exchange commands with the original en route computer, which runs all of the air traffic control algorithms in about 3 megabytes. In simple psychological terms, we all went nuts.

The project was handed over to human factors pundits, who then drove the design. Requirements became synonymous with preferences. Thousands of labor-months were spent designing, discussing, and demonstrating the possibilities: colors, fonts, overlays, reversals, serpentine lists, toggling, zooming, opaque windows, the list is huge. But, it was something to see. (Virtually all of the marketing brochures—produced prematurely and in large numbers—sparkled with some rendition or other of the new controller console.) It just wasn't usable.

The display management software was designed as a single program, thousands of statements long, its task to interpret and apply hundreds of rules (in combination) aimed at mere presentation. It took years to write, and it was rewritten a dozen times. The requirements themselves were highly structured and complex, spanning the idiosyncracies of individual facilities, sectors of airspace, and controllers. Never before would the human retina play such a profound role in the design of a system—and, as it turned out, the dexterity of the human hand. (In retrospect, it may be that we were able to field the original en route and Tracon systems within a reasonable time because, on the front end, in development, we could use only punched cards—there were no CASE tools—and our programs produced data for mechanical equipment and displays that operated much as they had in the 1950s. No human factors analysis. No indecision. We could concentrate on the arithmetic—which was tough enough.)

The cost of what turned out to be a 14-year human factors study did not pay off. Shortly before the project was terminated a controller on the CBS evening news said: "It takes me 12 commands to do what I used to do with one." I believe he spoke for everyone with common sense.

The Testament

Rummaging through one of the closets at the far end of the hall on the fifth floor one day, looking for some standards document, I found an

envelope left by someone who left the company—as many did after so many years advancing against stone, while the wheels of commerce were accelerating on what everyone referred to as "the outside." It contained "A Brief History of the Advanced Automation System." It was printed by hand and left, perhaps inadvertently, or perhaps with the hope that some anthropologist might some day discover it and make a pronouncement. In every important way, it is the truth.

"A young man, recently hired, devotes years to a specification written to the bit level for programs that will never be coded. Another, to a specification that will be replaced. Programmers marry one another, then divorce and marry someone in another subsystem. Program designs are written to severe formats, then forgotton. The formats endure. A man decides to become a woman and succeeds before system testing starts. As testing approaches, she begins a second career on local television, hosting a show on witchcraft. An architect chases a new technology, then another, then changes his mind and goes into management. A veteran programmer writes the same program a dozen times, then transfers. The price of money increases eight times. Programmers sleep in the halls. Committees convene for years to discuss keystroking. An ambitious training manager builds an encyclopedia of manuals no one will use. Decisions are scheduled weeks in advance. Workers sit in hallways. Notions about computing begin in the epoch of A, edge toward B, then come down hard on A + B. Human factors experts achieve Olympian status. The Berlin Wall collapses. The map of Europe is redrawn. Everything is counted. Quality becomes mixed with quantity. Morale is reduced to a quotient, then counted. Dozens of men and women argue for thousands of hours: What is a requirement? A generation of workers retires. The very mission changes and only a few notice. Programming theories come and go. Managers cling to expectations, like a child to a blanket. Presentations are polished to create an impression, then curbed to cut costs. Then they are studied. The work spikes and spikes again. Offices are changed a dozen times. Management retires and returns. The contractor is sold. Software is blamed. Executives are promoted. The years rip by with no end in sight. A company president gets an idea: make large small. Turn methods over to each programmer. Dress down. Count on the inscrutability

of programming. Promote good news. Turn a leaf away from the sun. Maybe start over."

IBM Response to FAA's Cure Letter: 12/10/92

"IBM recommends . . . a new schedule, faster processors, and a new software development manager. The Chairman of the Board of IBM's Federal Systems Company (who was quoted later, 'if people could have just left it alone, we could have delivered it') will take over as IBM's AAS Program Manager. IBM will renew its efforts to test the software."

The Curtain

When the project ended it was reported in The Washington Post that there were over 3000 software defects. Other papers confirmed what everyone already suspected: the software could not be written. It was riddled with bugs. Everyone was relieved. It was the software. It was the programmers, the way they programmed, and how they were managed, how loops were divined, and the assignment of numbers to variables, how modules were laid out, and how queues were allocated, and the time spent in the lab (was it enough?). The original software development manager had been the first to go. She was defamed, first from within and then outside, according to the rules of blame, written down nowhere, but embedded deep within our unconscious. Then the test manager, the IBM project manager and his financial officer, then hundreds more left.

It was the software of course. But not in the sense we all want to believe. It was in the *nature* of software, the nature of programming; its roots tangled in the mathematics of contradiction and logic and symbol-production, plied by young tradesmen schooled in the tools of the latest process, its promise intensified by greed and ease, its complexity beyond imagining, man mingling with machine to a degree that cannot be described, felt to be immune to the laws of size—these computer programs: easy to miss within the great facilities that house them, amidst the spandrels and joists and tree-like arches and great enclosures, decorated for art's sake.

I don't know how the programmers felt about the demise of AAS. I didn't ask them. Most of them worked for years and knew about as much about the forces at work as the Fijians knew about the causes of the Second World War. They worked too hard to know. Twenty-five hour weekends. Sleep-overs. In-plant babysitting. (Ask anybody who's been a programmer on any project.) Afterwards they left, ran to other jobs, any jobs. Many are on their second or third job now. They didn't leave for advancement. They left to get away. I don't know how they endured. They did good work. In the shank of the project, the suffocating bureaucracy and the beckoning of video games their friends in other places were creating in the new age of software did not affect them. They were a cadre of disciplined programmers, who would not cut corners on writing well-documented and thoroughly-read code, who tested their bottoms off, and believed in metrics, who wrote reliable programs, when they had every reason to give in to hurriedness. Or just give in.

Coda

from the hearing before the Committee on Public Works and Transportation, Wednesday, April 13, 1994, 9:30 a.m.

The Honorable David R. Hinson, Administrator, Federal Aviation Administration:

"This Subcommittee is well aware of the troubled history of the AAS program. The review . . . reflects a range of costs from $6.5 billion to $7.3 billion for completion of the program, and slippage of implementation dates by 9 to 31 months.

"I tasked the Center for Naval Analysis with conducting an independent 90-day review. I wanted that unvarnished look from an outside group."

The project ended quietly enough. There were a few articles in the big city papers, some personnel were shifted, but, in general, civilization flowed on as usual. The CNA performed its task and made their recommendations on cue, among them the "clear evidence that electronic flight strips were unnecessary; air traffic control companies that provide this capability found it necessary to provide paper strips." The all-digital approach would not do. Too

much automation, they reasoned. The FAA had insisted upon the utmost in automation. With 80% of ground control already automated, they beat the drums for the other 80%, squeezing out the remaining 20%, and redoing much of what already worked. (That is true, but so is this: put enough time and money into any project, up-front, and it will fail.)

In 1994, when the curtain fell, the FAA pressed Congress for help. Within months, Congress passed legislation permitting the FAA to waive the burdensome federal procurement regulations—over the protests of Senators John Glenn and Strom Thurmond, who argued that other government departments do just fine under the already streamlined rules. Senator William S. Cohen spoke out as well: "Regrettably, Congress bought the argument that federal procurement and personnel policies (personnel rules were reduced from 1069 pages to 41 and "integrated product teams" were put in place) have prevented the FAA from modernizing the air traffic control systems; but federal policies are not the problem. It was poor management." (Senator Cohen is mistaken, of course. Poor management? For decades, the FAA has "managed" to keep the most complex system on earth running 24 hours a day, seven days a week. I know of no more competent and committed managers. Is management the problem down the block as well, at the IRS and FBI? The IRS management is being replaced because of software project overruns. The FBI is losing Congressional support, not because of law enforcement problems, but because of two software projects. Where will it end, I wonder? How many managers in how many departments in how many nations will be replaced because they were undone by the seduction of loops?)

So, cumbersome regulations replaced software as the problem, its slate wiped clean. From now on, the FAA, unlike the Department of Defense—who must push on conservatively—can acquire systems without much oversight, so that, according to David Hinson, the past FAA administrator, the FAA can keep apace with industry. (If he meant the software industry, which is more and more taking on the character of alchemy, the problems of air traffic control may go from bad to worse.) Thus, the Advanced Automation System, failing to overhaul air traffic control automation, succeeded in restructuring

the FAA. And, officially, we have learned nothing about the pitfalls of software and its use in large systems. As for IBM, it sold the Federal Systems Company to raise cash.

Epilogue

When I wrote this piece for Bob Glass, a few years back, he asked me to summarize the lessons we might learn from the AAS. He meant the lessons for practicing programmers—putting aside the limitations of programmable computers and the madness of the human condition embodied so vividly on the Advanced Automation System.

I mentioned some originally, and I would not go back on them. Surely, the problem of size and complexity is central. Although, these can easily be gotten around by misinterpreting software as a manufacturing process, composed of source statements and such, instead of the complex logico-mathematical phenomenon that it is. (On AAS, virtually all measures of effectiveness, including costs, turned on the notion of software as a collection of "lines of code.")

The two technical issues highlighted in the article also brought to the surface "limits": distributed processing, more accurately described as distributed semantics, in which meaning must cohere in the face of latency and a panoply of failure modes (as opposed to the relatively "stateless" nodes of most networks); and the "infernal CHI," as one manager called it, which became a preoccupation non-pareil.

I would add, also, the problem of programming design, which, in the late 1960s and 1970s was beginning to make inroads. It seems that there is no good way to prevent programmers from rushing ahead in the quest for intellectual control. What was once called "stepwise refinement," the gradual decomposition of structure and semantics, is not enforceable, and, if it were, it would be too easily breached. Mostly, programmers just code—as is their and their managers' preference.

There is one last thing. It is equal parts psychological and technical. Software is both a source of amusement and engineering achievement. It is easy to change, to the point of whimsy, and it allows us to do things heretofore unthinkable, like putting man on the moon. It is contradictory and miraculous. (Perhaps all miracles are contradictory.) As the genetic engineers and nano-technologists well

know, get far enough down into the world of symbols and one can transform the very nature of structure and meaning.

But the miracles produced by computers and computer programming are not enough. We've gotten accustomed to them. Now we would like them to be easy: we invent tools, themselves programs, that make vendors wealthy, but seem only to complicate the already complicated, and we invent new paradigms, such as reusability and non-developed items, and commercial-off-the-shelf, to convince ourselves that programming is not so hard after all.

Not satisfied with easy miracles, we want them to occur immediately: the Department of Defense has spent millions on productivity, and private companies sling their products out the door without hesitation, as if heaping more technological side effects on a planet already choked by them were not relevant. And, to boot, the trickier the nomenclature the better: we delight in giving old concepts new names, so that software remains, perforce, a field for the young. I believe the issue has been decided: once near-science has surrendered to cant.