Digital Apollo:

Human and Machine in Spaceflight

David A. Mindell

The MIT Press
Cambridge, Massachusetts
London, England

7 Programs and People

Quest oculus non vide, cor non delet
What the eye does not see, the heart does not regret
"A lot happens that we are not telling you about."
—Opening lines of Apollo software source code

Programming the Moon Flights

With the exception of integrated circuits and extremely high reliability, the hardware for the Apollo guidance computer represented the state of the art when Apollo began. The same could not be said of the software and user interface. An aspect of the system barely envisioned when the program started, software turned out to be among the most difficult, and the most critical, components in the Apollo system. The software would carry all the burden of Richard Battin's complex guidance schemes. It would embody the goals and constraints of specific missions, and hence would require special finesse (sometimes completely new programs) for each one. It would control the flight at critical moments and stand between the astronauts, their machines, and the ground controllers and their tasks. Most important, it would interact with the astronauts, who might do unpredictable things at strange times in seemingly random order. As the Latin aphorism quoted above warned, the code contained secrets and hidden truths not apparent to the user.

Apollo began in a world when hardware and electronics were suspect and might fail anytime. It ended with the realization that as electronics became integrated, computers could become reliable, but that software held promise and peril. It could automate the flying, eliminate black boxes from the cramped cabin, and make the subtlest maneuvers seem simple. Yet it was also hideously complex and difficult to manage. If it went wrong at a bad time, it could abort a mission or kill its users.

When NASA first let the contract to the IL for Apollo guidance, programming was barely mentioned. In fact, the word *software* does not appear in the original statement of work. The word had only come into use in the late 1950s (*The Oxford English*

Dictionary lists the first usage of the term as 1960).[1] The original work statement referred to programming only twice, the most extensive reference being: "The onboard guidance computer must be programmed to control the other guidance subsystems and to implement the various guidance schemes."[2] Software was not included in the schedule, and it was not included in the budget. The absence of software in the initial plans reflected not only an underestimation of the effort, but also the belief that writing code was part of engineering the system, not a discrete activity. Dick Battin likes to recount that when he told his wife he was in charge of Apollo's "software," she found it unmanly and joked "please don't tell our friends."[3]

At the start, there were fewer specially trained programmers than engineers who could turn their ideas into executable code. As the IL's official history put it, "It was believed that competent engineers with a credible, solid mathematical background could learn computer programming much more easily than programmers could learn the engineering aspects of the effort."[4] And it was difficult to find people who were skilled at both. Programming was new enough that it was not broadly recognized as a technical, much less engineering skill. Jack Garman managed the software project for NASA for a few years, and he recalled an attitude that "the real problems [in spaceflight] are in propulsion or in . . . plumbing. I mean pressure vessels and stuff like that. That's where the real macho kind of space biz is."[5] Just writing instructions for the digital circuits still seemed a soft afterthought. The IL was fundamentally a guidance and control organization—mathematics, electronics, and the high-precision gyros were the dominant technologies. Obscure lines of machine code didn't have their own place of prestige in the organization.

Nevertheless, in the summer of 1961 a few engineers at the IL got started programming. Battin was in charge, with a laid-back management style typical of dedicated academics. "He didn't really run it," one of his engineers recalled, "he just nodded." Another thought of him as "a real hands off manager, just a guru in his corner."[6] In the early days NASA's attention was on Mercury and Gemini, and the IL engineers had a great deal of autonomy. Dan Lickly, who worked on reentry guidance, remembered a loose environment.[7]

Fred Martin, a calm, intense engineer, started at the IL in 1956, working on autopilots and fire-control systems before winding up in the Polaris program. He then went back to school at MIT, though he stayed at the IL and joined Battin's guidance and navigation group. Martin soon became the project manager for the CSM software, which, before the LOR decision, comprised all of the Apollo flight code. Martin remembered the "small group, seat of the pants effort that was unfettered and unhindered by NASA and devoid, pretty much, of real bureaucracy." He divided up the program, assigned tasks to different people, and kept track of their progress.

Some who worked for Martin remembered his subtle, hands-off style, "like a psychiatrist."[8] During this early period, the group focused on the fundamental engineering

issues associated with getting to the moon, particularly the subtle mathematical models and numerical filters required to robustly solve the guidance equations in real-time.[9] Some referred to it as the "systems engineering phase," when budgets and schedules were loose and the group could afford to look at the big picture.[10] Guidance to the moon seemed an interesting mathematical problem, and a calculating machine should be able to handle it without much trouble.

People at the lab worked long hours focused on a clear goal with compelling technical challenges. As in the rest of Apollo, sixty- or seventy-hour work weeks became the norm. "There was a feeling of national urgency, you might say, of national responsibility," Martin recalled. "You had the astronauts coming to the lab regularly. . . . You had the pressure of what was going on." The IL group felt that NASA and the astronauts held them on a pedestal: "Everybody was a guru and everybody was a genius." Lickly remembered "a very privileged position," as the programmers regularly received visits not only from astronauts, but also from von Braun and the top NASA management, "all those people we got to talk to as a small sort of elite organization."[11] Some IL engineers never found such creative excitement again in their careers.

Apollo software would be the product of many people's work, integrated into a coherent whole. Martin's group had physicists and mathematicians and engineers, "probably had a literature major here and there too," he said, but no computer scientists. As few people were coming out of school with computer science degrees (MIT did not even create its department as part of electrical engineering until 1968), the IL group included people like Alex Kosmala and Margaret Hamilton. Hamilton, one of the few women engineers in all of the Apollo program, had a mathematics degree and had already worked as a programmer on the SAGE air defense system at MIT's Lincoln Labs. Lickly worked on the reentry programs for the command module. Others focused on navigation, or the inertial platform.

The LOR decision was not made until a year after the IL began working, so software design and programming began in a vacuum of detailed mission goals and specifications. Engineer Ed Copps remembered an exciting time as they contemplated the possibilities of a small computer onboard a spacecraft flying to the moon: "We had automata doing things that we had not planned for them to do. . . . We were the new guys coming to grips with new capabilities, which were now placing conceptual issues within the reach of actual implementation. . . . We designed it in a very abstract way."[12] One day Kosmala, a programmer, went to the MIT library and looked up a series of star coordinates to align the inertial platform, which he then embedded into the code. That star catalog survived into the final version and became the basis for the astronauts' navigation to the moon.

The official IL report on the software effort credited its success to "an intricately-tuned interaction among men and machines."[13] Interestingly, this statement refers not to the interactions aboard the spacecraft, but rather to the coordinating and

scheduling of engineers and programmers on the ground, and their mainframes and simulation machines. Early on, of course, no actual Apollo computer existed to test the programs. The IL had a number of mainframes, including a Honeywell 1800 and the new IBM 360, which ran simulations of the Apollo computers. Indeed, throughout the program the software ran in simulation for validation on these machines, as well as on actual Apollo hardware. A digital simulator could analyze program operation in great detail, step by step, with a host of tracing and reporting tools, but at a comparatively slow speed. The simulator included routines like UNIVERSE, LUNAR, and TERRAIN to model various environments, and even one called ASTRONAUT that simulated the human operator. To complement these digital models, analog computers simulated the spacecraft's dynamics, from center of mass and rocket thrust to parameters for structural bending and fuel slosh, connected to models of the AGC that would run in real-time. Hybrid simulators mixed the two, and even included a sextant, an inertial unit, and a full user interface, allowing engineers and astronauts to exercise the system from the front panel.[14] Together, they amounted to building the Apollo spacecraft and traveling to the moon in a completely numerical, virtual environment, an electronic equivalent of the wind tunnels of an earlier era.

During the 1960s, just across the same campus, project MAC at MIT was incubating much of the technology and culture that underlay computing today, from graphical user interfaces to the nocturnal hacker culture. Students and young engineers were becoming enthralled with the creative potential and practical difficulty of interactive computing. Apollo programmers at the IL, while a separate group in distant buildings, shared some of this ethos. They tended to believe that any program could be written by a few smart people. As the project grew, they began to realize the need for more help, but were reluctant to jeopardize their small-group culture. "We've got to do this with either five people, maybe, or we've got to do it with 150," Jim Miller remembered thinking.[15] The group, he said, developed a closed, proprietary attitude toward their system: "We couldn't figure out why anyone else needed to know what was in the guidance computer besides us. What does NASA need to know for?"[16]

Of course, NASA needed to know because that was how its managers practiced systems engineering, where every subsystem had to be a white box open to scrutiny. And they were the customer. Martin later realized that MIT's hard-headed, creative exuberance appeared to NASA as exclusionary. "I think that that arrogance started to irritate people," Martin said, "who were tired of hearing that MIT knew best and knew better." NASA began to feel the need to "rein in" the IL, "to get them on this program not on their own program."[17]

Only in 1964 did NASA begin a series of meetings of a "G&N system Panel" to coordinate requirements among the contractors. Another series of meetings began to specify the interfaces—how the computers would read data and command actions of the spacecrafts' various sensors and actuators.[18] The computer, it turned out, especially

Block II, would tie together numerous different systems in the vehicle—from clocks and needles to hand controllers and rocket engines.

Software as Systems Integrator

Apollo's software derived from the basic design of the Mars mission. Designer Hugh-Blair Smith created a language called "Basic," a low-level assembly language of about forty instructions (distinct from the high-level BASIC programming language developed at Dartmouth at about the same time). On top of Basic was "Interpreter," brainchild of Hal Laning, a language that was really a collection of routines to do the higher-level mathematical functions associated with guidance and control, in the high-precision data format. Interpreter also had vector functions like matrix multiply and cross products that were useful for calculations of vector-based quantities in navigation and control.[19] A master program, "Executive," also a Laning creation, decided which programs ran when, including responding to high-priority interrupts.

Not by coincidence, Executive and Interpreter were both named after human roles. The software code reflected the social system of the lab. Certain programs were written by certain people or groups and they all had to work together and talk to each other, whether inside the machine or inside the laboratory. As Apollo flew, each particular operation depended on computer code written by a particular engineer, eagerly watching from Cambridge, Massachusetts, as a piece of his brain flew through space with the astronauts. Hamilton learned to think of an entire mission as one system, "of which part is realized as software, part is peopleware, part is hardware." Everything was linked, and an "error" in software could not be easily isolated to a particular line of code, as distinct from a specification, an interface, or a crew procedure.[20]

Executive and Interpreter comprised what we would today call the operating system, which also included code like program sequencers and interrupt-response routines. They reflected how the Apollo computer allocated its time, in a manner still novel in the 1960s but quite common today. The great innovation in computing in the 1960s was "timesharing," the idea that many users (or programs) could access a computer simultaneously, in real-time, as though each had the machine to him or herself. Timesharing worked by allocating a slice of time to each user or program, and then switching between them many times per second. When designing Apollo's software architecture, Laning eschewed this fixed-time-slice mode of operating. Instead, he made the Executive program asynchronous: it used a system of priorities, focusing its attention on whatever task was the highest priority at a given time. When that task was finished, Executive moved on to the next one, and so on down the priority list. This scheme had the virtue of always making sure the important things got done and of not taking resources at a critical time for operations that were less important. Updating the displays on the astronauts' keyboard, for example, was less critical than making

sure the LM's thrusters kept it stable and upright. Autopilots, counters, timers, sampling the inertial sensors, telemetry, and computer housekeeping events went on regularly in the background even as a major computation took place.

The asynchronous executive had one major drawback: the program flow was unpredictable. No one could say exactly how it would respond in a given circumstance, because not everything was scheduled in advance; subroutines ran in real-time in response to events or as they rose up in the priority list. One might make the analogy between the asynchronous executive in the Apollo computer and the Apollo "program" overall: numerous different tasks, working cooperatively but competing for attention and resources.

As one example, consider the critical program P40 that controls the guidance and navigation during countdown, ignition, and thrusting of the main engine on the CSM, used to change the orbit of the spacecraft. A previous program calculates the desired time of ignition and the parameters of the existing and desired orbits. P40 then calculates the direction of the thrust and velocity to be gained, and calls a host of other programs to calculate gimbal angles, display thrust values to the astronaut, maneuver to the appropriate attitude, update matrices, and count down. All the while the computer is running autopilot calculations, exercising the reaction control jets to keep the spacecraft pointed, performing system maintenance, and even responding to other programs of higher priority. The net effect is an apparent cacophony of programs within the computer, all vying for priority and sharing memory, displays, and processor time.[21] When it worked, it proved an intricate, well-coordinated symphony of code.

The worst thing that could happen to a computer on board a flight was for a bug to occur during a critical operation and cause the program to "hang" or freeze up (a common enough experience for computer users today). Rather late in the program, NASA made a decision that the software should allow the computer to be restarted, literally in the middle of a maneuver, without corrupting the process. This feature, known as "automatic restart protection" helped protect against transients on the power supply (such as from a lightning strike) or software problems that stuck the code into an infinite loop.[22] One programmer described it as "giving the computer an enema." A clever idea, to be sure, but it forced the programmers to rework every program, and every subroutine, to keep track of its current state in a permanent way, so if a restart occurred, it could pick up again without interruption. As Copps put it, "It was actually the right thing to do . . . but it really made things a bit more complicated. I would say a lot more complicated."[23]

The asynchronous executive also faced deep resistance. Synchronous scheduling was accepted practice, especially among those in the aircraft industry whose computer experience tended to be with timesharing mainframes on the ground. "It was very important to them," Martin recalled, "to know exactly what was happening every instant of time." He characterized the debate over synchronous versus asynchronous executives as "almost a religious war."[24] Again, the analogy of project management is instructive: should everything be directed from a central location on a precise schedule? Or coordinated in a looser, almost organic form? This seemingly arcane technical debate had implications that would surface dramatically in the final seconds of Apollo 11.

Building Programs

For about four years, the programming effort trundled along at a comparatively leisurely rate while the hardware design proceeded apace. No more than a hundred people worked on software until mid-1965, when the hardware effort peaked with more than six hundred people. Once the hardware design was completed, however, the manpower began to decline, and by the end of 1966 there were more people assigned to the software effort (about two hundred and fifty), which peaked at more than four hundred in mid-1968.[25] These numbers reflect the last-minute programming crises that arose before the flights (figure 7.1).

Apollo programmers faced an irony: the early missions were unique test flights requiring custom programs whereas the later, more challenging flights with landings tended to have more uniform requirements. For each mission, the IL developed a "guidance systems operations plan," or GSOP, that had all the specifications, parameters, and custom routines (the CSM and LM each had their own GSOP for a particular mission). Essentially a series of equations and flowcharts, the GSOP theoretically contained everything required in advance to produce the program code for each mission. In practice, the GSOPs tended to document code after it was written, rather than specifying it beforehand.[26]

For each mission, the infrastructure for running and managing the programs came together with the GSOP and generated code that implemented the equations for navigation, targeting, and trajectory calculations, based on Battin and Laning's work. Early missions, unmanned test flights, tended to be sequence-oriented, commanding the spacecraft to do a series of specified operations, or allowing control from the ground. Final versions were named after the sun (Apollo being the sun god): ECLIPSE, SUNRISE, and CORONA. Some were just collections of routines for testing, others, like SOLARIUM, actually flew and executed real maneuvers. SUNDANCE evolved into the program for the LM. These programs were divided into two categories, for the two spacecraft. A naming contest for programs beginning with "C" for the command module and "L" for the LM rejected "coughdrop" and "lemon." Instead, COLOSSUS flew on the command module, and LUMINARY flew on the LM, each in various

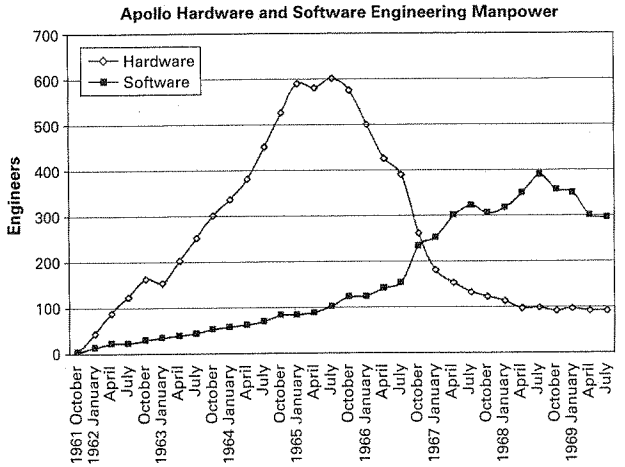


Figure 7.1
MIT Instrumentation Lab Apollo manpower plots, showing the time difference between the peak of the hardware and software efforts. Note the rapid rise in software manpower during 1966 when five core ropes (i.e., flight programs) were being developed simultaneously, and when Bill Tindall from NASA intervened. These figures do not include subcontractors. (Redrawn by the author from Johnson and Giller, "MIT's Role in Project Apollo, Vol. V," 19–21.)

versions (LUMINARY IA, for example, ran the Apollo 11 landing). These had full display and keyboard functions and allowed the astronauts to select and run individual programs (figure 7.2).[27]

Programmers labored under strict constraints. All of the programming was done with punch cards—there were no "online" terminals for rapid interaction with the computer. In typical hacker fashion, stacks of program cards were assembled at night. Managers used this rhythm to advantage and began to hold nightly "configuration control boards" that would incorporate any changes that had been made during the day into a single overnight computer run. Cumbersome as it might seem today, the nightly builds proved an effective means of keeping control of the software changes.[28]

Along with schedules, limited onboard memory imposed the tightest constraints. NASA and the IL had to come up with strict priorities about what was important enough to include in a flight. Jack Funk at NASA felt that from the beginning of the program, the IL "made a fantastic error in judgment" in their estimate of the memory

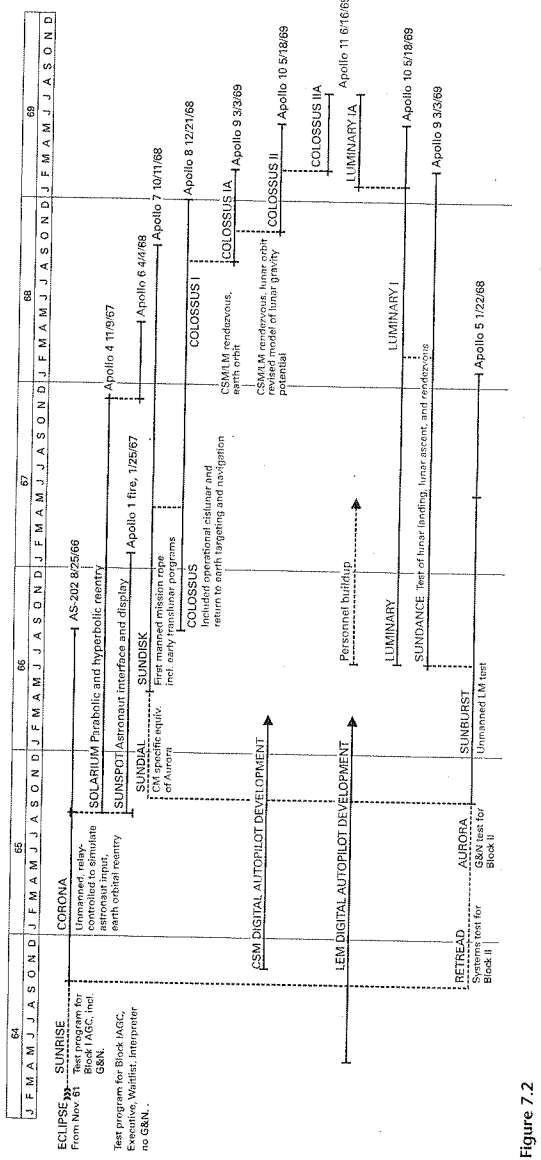


Figure 7.2
Timeline of software versions and Apollo flights. Note the busy years of 1966 and 1967, when five and six ropes were in development, respectively. (Redrawn by the author from Johnson and Giller, "MIT's Role in Project Apollo," 22.)

capacity required for the Apollo computer, in part by including only enough in their estimate for the calculations themselves and not enough capacity for the executive and other infrastructure that would make the life-critical system ultrareliable. "That was some battle, to convince people that we needed that big a computer to begin with," he said.[29] The original Mars computer had 4,000 words, which was what the IL proposed for Apollo. As built, the original Apollo computer had 8,000 words fixed memory, then it was doubled to 16,000, then to 36,864 fixed memory and 2,048 words erasable for the Block II.

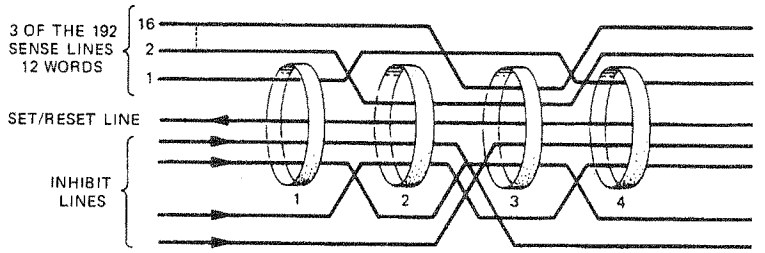
In today's world of prodigiously cheap memory, it is easy to forget how dear storage space was in those days. A great deal of programming effort went into shrinking the memory required. Different routines could actually share the same erasable memory for storing parameters, for example, but only if they never ran simultaneously. With the asynchronous executive, that was not always perfectly predictable, leading to subtle problems with overlapping data. Moreover the software tended to accumulate functions as time went on and hardware units were eliminated or consolidated, so memory and processor time became ever more stuffed.

Manufacturing Code

Today we are used to the idea that software is really "soft"—that is, it weighs nothing, is easily and perfectly copied, and is essentially cost-free in time and money to manufacture. Download a new piece of code into your PC or your cell phone and you're off and running. This was not the case in the 1960s, at least not for the embedded system of Apollo. Software became hardware.

The permanent memory, which stored the flight programs, consisted of a complex series of wires running in and out of magnetic cores that determined if a particular bit in a memory location was a one or a zero. At a given memory location, a wire going through a core represented a "1," a wire going around a core a "0" (actually, clever wiring meant that each core could actually store sixty-four bits). The small amount of erasable memory used a similar technique. Thousands of these cores, meticulously threaded with thin, hair-like wires, were packed together into "ropes" that held Apollo's programs. While cumbersome, this approach had one great advantage: the program was indestructible, literally hard-wired into the ropes. Astronauts became grateful for this feature when lightning struck Apollo 12 just after launch and the computer perfectly rebooted itself. Apollo spacecraft had no disk drives, no FLASH memory, not even any magnetic or paper tapes. The software for Apollo was an actual thing. You could hold it in your hand and it weighed a few pounds (figure 7.3).

This "firmware" meant the programs had to be manufactured in a factory, largely by hand. Raytheon, which built the flight computers for Apollo, was also responsible for manufacturing the ropes. This process entailed precise, painstaking sewing of very fine



TRUTH TABLE

Core No. \ Sense Line No.	1	2 . . .	16
1	1	0	0
2	0	1	0
3	0	1	1
4	1	0	0

Figure 7.3
Simplified schematic of principle behind core rope storage. The inhibit lines select a particular core "address," and the sense line returns a "1" or a "0" depending on whether it threads through or around the core. Each core thus stores 12 bits. (Hall, "MIT's Role in Project Apollo, Vol. III," 90.)

wires through thousands of tiny cores. The particular pattern of the wires, and which direction they went through each core, determined the pattern of 1s and 0s stored in the memory.

Raytheon did the manufacturing in its plant in Waltham, Massachusetts. The town had a history of precision machining (the Waltham Watch Company was nearby), and drew on an industrial community familiar with weaving and textile manufacturing: "we have to build, essentially, a weaving machine," Raytheon manager Ralph Ragan told the press.[30] Raytheon assigned the work to older, female workers. Engineers nicknamed them "little old ladies," and actually referred to them as "LOLs." Core rope weaving was a specialized skill, and Raytheon paid the women to sit around and do nothing if the software ran late, so they would not be called to other projects that would degrade their currency.[31] NASA was well aware that the success of the flights depended on the fine, accurate motions of these women's fingers, so they sent the astronauts through the plants to make the workers aware of the human impact of their work (as they did with many of Apollo's factories). One engineer fondly recalled that the women at the plant "adopted" the astronauts, and took great care and pride in the quality of their manufacture (figures 7.4 and 7.5).[32]

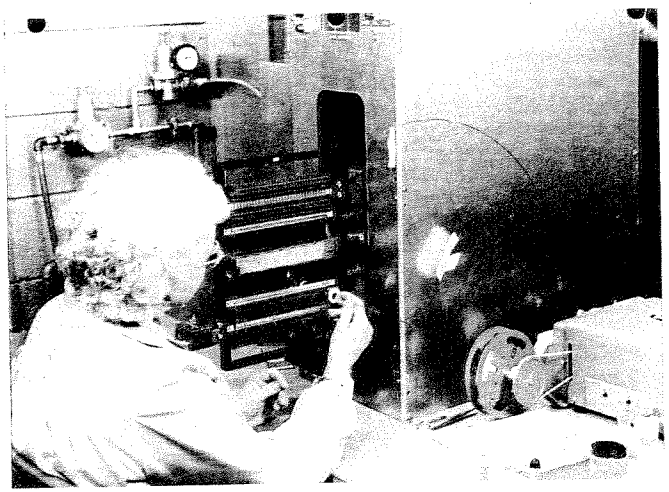


Figure 7.4
Sewing software: rope manufacture with a numerically controlled machine. The paper tape in the reel at right controls the machine to place an eyehole over the appropriate core, and the worker threads her needle through with a wire attached. (Raytheon photo CN-4-22.)

Rope manufacture did become partially automated. Another New England firm, the United Shoe Machinery Company, created a machine to speed up the process. The IL would convert their programs to a series of paper tape instructions to read into the machine. A rigid rack held an assembly of cores, and the machine used the numerical data from the punch cards to position the core assembly so the worker could quickly run her needle, with a wire attached, through each core. Then the machine would reposition the cores and the worker would run the wire through the other way. A series of folding, soldering, and potting operations then turned the core rope into a "module" that could be plugged into the Apollo computer.

Ideally, this process would happen once, producing single core rope that would program any possible mission. Each mission had its own unique features: vehicle mass, propellant volume, orbital parameters, and the like, that would go into the erasable memory. It never worked out that way—each mission was sufficiently different, and each program was sufficiently modified from the previous one that no two flights ever flew with exactly the same code (other than AS-501 and AS-502, which were unmanned).[33] By the later lunar landings, however, the programs did become relatively stable, and the changes increasingly minor.

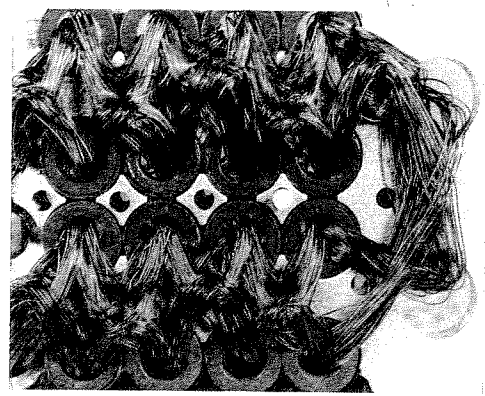


Figure 7.5
Apollo's software: close up of a core rope. (Raytheon photo CN-1156-C. Reprinted in Hall, *Journey to the Moon*, 15.)

Cumbersome and time-consuming, the rope manufacturing process took many weeks to go from functioning code to workable ropes. The pace had a critical implication for the programmers: the software had to be "frozen" at three to four months in advance of a flight.[34] IL engineer Ray Alonso remembered his initial shock when learning of the lead time: "What? I can't walk up to the launch pad and change whatever the program I like?" After all, what was the point of a programmable system? Sometimes the flights were sufficiently close together, in fact, that lessons learned from one flight could not be included in software fixes for the following one. A process for changing the ropes after manufacture did exist, but it required removing the potting that encapsulated the wires and surgically changing one bit at a time, in Alonso's words, "an incredibly small and arduous process to go through."[35]

NASA and IL management recognized that the schedule could help them impose a kind of discipline on the otherwise free-wheeling programmers. "Rather than a disadvantage," wrote David Hoag, the nature of the core ropes meant that "risky last minute changes of the program just before flight were physically prevented."[36] Manufacturing the software forced the engineers to make commitments, to stand behind their programs and get them into a form that would be suitable for testing months in advance.[37] It helped bring the young, unruly art of programming into the purview of program management and systems engineering.

But Apollo's programmers had to face more than the factory floor. They also had to face the users.

Astronauts and Automation

As eventual users of the Apollo guidance system, the astronauts had values, opinions, and goals that complicated the programmers' work. Most of the IL staff remember the early astronauts as resistant to electronics, computers, and automation. They were used to a world where electronics routinely failed. "The fact that we were going to do things in a digital system, an inertial system," Cline Frasier recalled, "they absolutely believed to their core that stuff wasn't going to work."[38] Sensitive about their level of activity in the spacecraft, astronauts found it difficult to accept the idea that the digital computer would be even partially in control.

As Alonso remembered it, the astronauts saw the computer like a calculator, a disposable appendage to the flight. "Of course we'll shut the computer off as soon as we're up there" he was told (indeed the early plans did involve shutting the system down between maneuvers).[39] Other times the astronauts wanted to approve, step by step, every move the computer made. Such a desire, of course, was wildly impractical with a computer running thousands of instructions per second and would lead to an impossibly high workload. Similarly, the users wanted direct control of the systems—right down to the valves themselves, asking, for example, for hand throttling of the LM's descent engine.[40] Nonetheless, the astronauts were not being reactionary, just hard-headed about which systems they were willing to stake their lives on. Lickly remembered his first lecture to the "new nine" on automatic control of reentry: "I was just overwhelmed by the smart questions. Neil Armstrong was all over it.... They knew the right questions to ask. They weren't Luddites."[41] Eventually, the word came down from NASA that the astronauts could suggest changes and provide input to the systems design, but that they could not dictate the basic controls philosophy.

Battin remembers a story he told to astronaut and Apollo 12 Commander Pete Conrad: "One day an airliner was on the field, and wasn't able to take off. An announcement came over the intercom, 'Sorry passengers, we can't take off, there is an indication of a problem with the airplane. We've got to replace a piece of equipment and it's going to be very difficult and take many hours.' Just half an hour later, another voice announced that the flight was ready to go. One of the passengers asked, 'I guess it was easier than you thought to replace that piece of equipment.' The airline representative replied, 'No, we replaced the pilot.'" Battin remembered Conrad being impressed by the story and repeating it to others.[42] The message: if the astronauts weren't happy with the automation, there were plenty of other people willing to go.

IL engineers gave regular courses about their systems to astronauts and other NASA personnel. In general, they found the astronauts disinterested in the computers, or even in the guidance per se. Research engineers at the IL had a habit of teaching the fundamental principles, but the crews wanted operational techniques and checklists. Indeed, much of the IL's training proved more academic and theoretical than the astronauts could absorb. Michael Collins found it sufficiently off-putting to record his response to one course in his diary: "This course was designed for someone who is going to either (a) build a better computer, or (b) repair and replace components of the existing computer. It was not a course for the pilot, who needs to know how to operate the computer and how to detect malfunctions."[43] Collins both rejected in-flight repair and made the point that the IL lacked experience in communicating with operators.

Still, astronauts made numerous suggestions to the IL engineers, most of them fairly concrete. For example, they wanted velocity units displayed in feet per second, even though the AGC used metric units internally. The software thus did a whole series of conversions between metric and English units for the user interface. Procedures and even trajectories were modified to allow the astronauts to monitor, and possibly interrupt, the automatic sequences.[44] A PRO button, for "proceed," allowed the astronauts to approve major actions (like firing the engines) before the computer commanded the hardware.

Difficult problems still arose, like the "attitude indicator" in the spacecraft, the primary display of which way the vehicle was pointing. Pilots were used to the artificial horizon in aircraft, a gyro-driven display that maintained orientation to the vertical, with a dark region on the bottom representing the earth and a light-blue region above representing the sky. In the spacecraft, this unit needed to be a ball, because of the wide range of possible attitudes. It became formally known as the "flight director attitude indicator" (FDAI), but colloquially as the "eight ball."

But how should the eight ball be oriented? In aircraft, a gyroscope drove the artificial horizon directly, which seemed elegant to the IL engineers, who proposed driving the Apollo eight ball off one of the gimbals in the inertial platform; like the platform itself, the ball would then be fixed in inertial space (i.e., in relation to the stars). This solution did not appeal to the astronauts, however, because it meant that as the spacecraft orbited the earth, the ball would seem to rotate as well and would not track the horizon. As an alternative, the ball could track the horizon, the "local vertical" (on earth or the moon), but that also created problems. A ball tracking local vertical would be essentially useless, for example, as the spacecraft pointed skyward on the launch pad and during launch before it pitched over toward orbit. A similar set of problems, in reverse, affected decisions on the eight ball for the LM. The astronauts got their way because they would have to use it, but it offended the IL engineers' sense of elegance and accuracy. "I do think that somehow they are being too much dominated by their experience in flying airplanes," Hoag said at the time, "to go over this [flat] plane which is the earth surface, rather than going out and away and trying to go between this earth and the moon."[45]

Another critical question: should the computer allow the astronauts to do something dangerous with the spacecraft? Initially, IL programmers put in a series of caveats and

constraints. Jim Miller remembered Alan Shepard objecting to this approach on his 1962 visit. "Take out all those inhibitions . . . if we want to kill ourselves, let us. It may involve saving ourselves," Shepard said. "Of course he was right," Miller remembered, "and it made the software a lot easier."[46] Yet Hamilton fought hard to get user-error-checking in the code. "We were very worried that what if the astronaut, during mid-course, would select the pre-launch [program] for example? Never would happen, they said." (But exactly that did happen on Apollo 8.)[47]

NASA insisted these were the most expert, highly trained users in the world. Of course the astronauts were very well trained, and experts in their systems. But they also got very tired, and building performance limits in the software could mitigate the effects of fatigue. Astronauts swore to Lickly they were going to fly the reentry manually, without using his automated program (this before Gemini's success with automated reentry). "As far as I know, none of them ever touched a manual stick," on reentry, Lickly remembered, for "they were so beat" after a two-week flight."[48]

At the 1963 press conference, Hoag put the issue more colorfully, describing the computer as "a young maiden who is asked something improper, and she'd also respond in that fashion. The computer, too, will not do things that it shouldn't." If an incorrect key or code were entered, the computer would signal "operator error" so the astronaut could try again.[49] In the end, the software came up against the limits of memory usage, and most of the self-checks had to be eliminated anyway. It did, however, check basic keystrokes and allow astronauts to recover from keyboard errors, analogous to a modern "escape" key or "undo" function.

The "Go to Moon" Interface

In 1965 a training session with the astronauts at the IL left one engineer confused. He was responsible for training the astronauts on the command module simulator for guidance and navigation, and noted how "proper operation of the G&N [guidance and navigation] essentially makes the astronaut a passenger." Then he asked for advice. "The astronauts seem to have an 'I want to fly it' attitude. This would occur only if G&N failed. Are we to train them to operate the G&N system, or train them to use everybody else's system when/if G&N fails, or both?"[50] How could you teach someone to use a machine that required nothing of him when working, and everything of him when broken? The appropriate user interface for the Apollo computer was far from obvious: chauffeurs or airmen?

The IL had built its expertise with inertial guidance systems, largely for missiles. Nobody flew an ICBM. With Apollo, the computer would have a user to input commands, request data, and ask the computer to do things that might be difficult to predict in advance. "We had a hard time getting used to the concept of somebody sitting there," Copps recalled, "banging on the keyboard, getting answers back, banging

something else in and asking questions."[51] The Apollo computer would have to interact.

Initially, Kosmala pictured the spacecraft with one button: "The astronaut goes in, turns the computer on and says 'Go to moon' and then sits back and watches while we did everything." Another version has the computer running two programs—"P00" to go to the moon, and "P01" to return home. These humorous descriptions capture one extreme of the engineers' view of the computer, one that didn't last long once the astronauts got involved in the design.[52]

John Miller remembered a constant battle. "The astronauts on one side wanted to fly the vehicle, I mean, they were test pilots. . . . On the Instrumentation [Lab] side, were, you know, automatic control . . . we're going to run this thing and the computer will run the thing. That battle went on kind of constantly."[53] An IL cartoon humorously captures the chauffeurs versus airmen extremes: one panel shows full automation, the astronauts smoking cigars and falling asleep, staring at the abort button. The next panel shows no automation, the astronauts overwhelmed by dials, indicators, charts, printouts, and inputs (figure 7.6a, b). The Apollo computer would have to operate somewhere between these extremes. But where?

Early in the 1960s, Charles Stark Draper himself and some colleagues had begun looking at the appropriate human role in aircraft and space systems, "as an off-line, parallel, complementary observer and actuator." To be an effective monitor, the human had to have some means to intervene in the system in order to take corrective action when problems arose. Draper and his colleagues used the word overseer; but the term supervisor became increasingly popular. Both evoke the relationship of a master to a slave, or a manager to a worker.[54]

"A transition in the art of piloting"

In 1962, the task of building the human interface for the Apollo guidance system fell to Jim Nevins, an IL engineer with a background in control systems. For Mercury, NASA had turned the job over to psychologist Robert Voas and to "human factors" experts at McDonnell, but Nevins began looking for control systems engineers with experience in ergonomics. He found one in Tom Sheridan, an MIT assistant professor with expertise in mechanical engineering and psychology who would come to define ideas in supervisory control and telerobotics. The group eventually grew to about thirty people. They began with traditional aircraft controls, giving the spacecraft hand controllers and throttles, then adding the gyroscopes and an artificial horizon (the eight ball), and inputs to control pitch, yaw, and roll. Each solution raised more questions: how would such machines be operated in a weightless environment? How by people wearing bulky pressure suits? Could an astronaut in a helmet put his eye to the sextant?

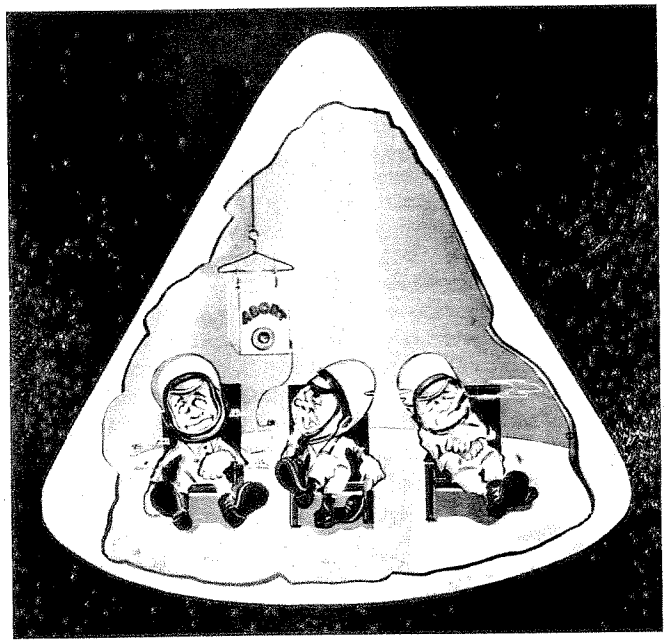


Figure 7.6a, b
MIT Instrumentation Laboratory cartoon showing the extremes of automation. Too much automation (above) leaves the astronauts bored, awaiting an abort, while too little (right) overwhelms them with work. (Draper Laboratories/MIT Museum.)

Nevins's group drew large pictures with columns labeled "astronaut" and "computer" and "ground," writing flow diagrams that detailed the transactions of data between these entities. "It basically described the decision that the astronaut had to make, the question that he would put to the computer, the answer that the computer was expected to give" and so on, Nevins said.[55] The designers were less interested in the mathematics than in the interactions between the astronauts and the machines. They detailed these tasks in verbal form, coming up with numerous "transaction schemes" for each type of operation (figure 7.7).[56]

They began looking at the information flows. What did the astronauts need to know at different points in the flight? What would display that data? Where would it go? How big should the displays be? How many numbers needed to be displayed at any

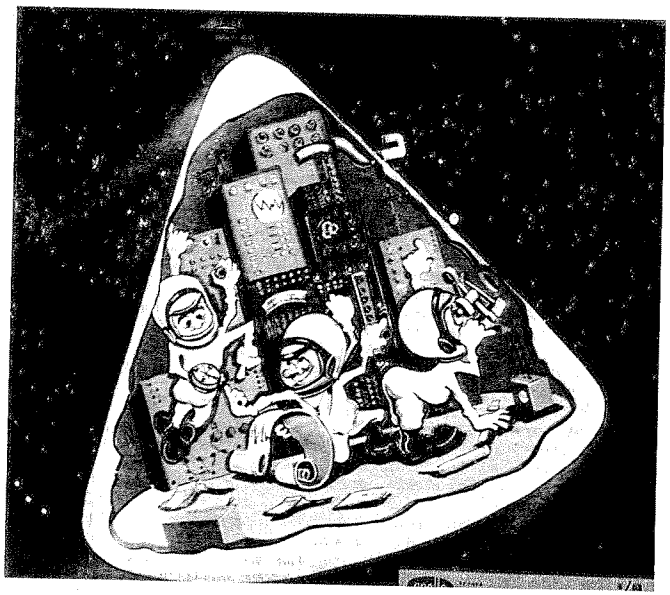


Figure 7.6a, b
(continued)

one time? Should there be a picture tube to display data? Eventually they settled on three lines of numeric display, because a picture tube would be too heavy and consume too much power, and because engineers were used to seeing vectors that had three components, three dimensions.

Nevins articulated his philosophy for astronaut-computer interaction based on their fit into the Apollo system overall. Aware of the broader implications of this approach, he called it "a transition in the art of piloting a vehicle." By "transition," he meant that astronauts would rely on interactions with the computer, and with controllers on the ground, to an unprecedented degree that would change flight forever. "The flight management system [in Apollo] instead of being an onboard operation is actually a highly integrated system of airborne and ground-based equipment." Overall, the astronauts and the computer formed but one part of "a finely structured multilevel monitoring and decision process." Ground controllers, Nevins predicted, will feel as though they are inside the vehicle, monitoring for slow degradations and trends while

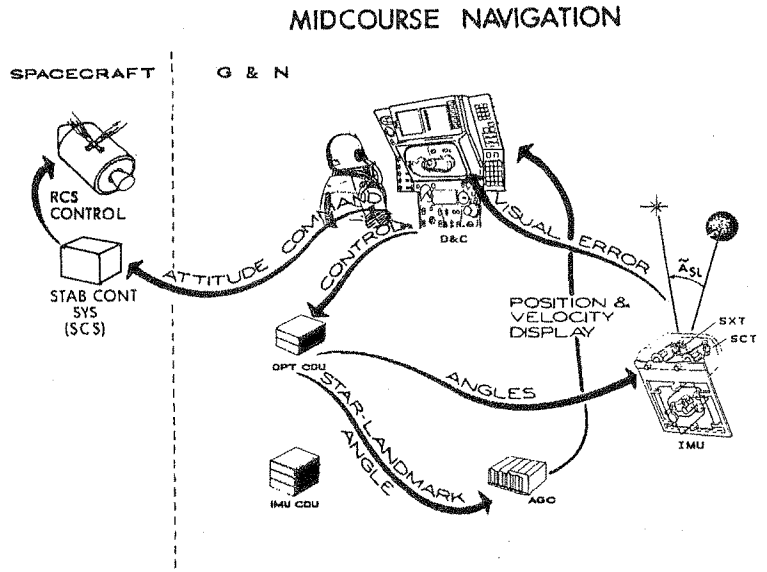


Figure 7.7
Interface flow diagram of a mid-course navigation illustrating information exchange between pilot and guidance and navigation system. (Draper Laboratories graphic 25381. Reprinted in Hall, *Journey to the Moon*, 62.)

the pilots remain alert for events that require quicker reaction. Flying to the moon would be a long way from piloting in the classical mode (although Apollo paralleled new developments in aviation where pilots shared control with controllers on the ground).[57]

For Nevins, the crew played a critical part by managing systems and occasionally aligning the gyros. Their tasks consisted of (1) monitoring and decision making in guidance and control; (2) sequencing and initializing automated systems for guidance; navigation, and propulsion; and (3) optical pattern-matching tasks associated with tracking and identifying the stars for aligning the inertial system.

Even so, "as technology improves . . . many of these tasks will be automated," Nevins believed. By contrast, the computer would (1) monitor sensor data; (2) determine thrust times and vectors, trajectory parameters, and lines of sight; (3) maintain attitude control; and (4) guide the vehicles during thrusting maneuvers. Punching a few buttons would bring the spacecraft to a particular attitude; punching a few more would fire the main engine to propel it in or out of orbit. The computer's job looked more like traditional flying.

Because of the complexity of the task, Nevins continued, it would rely heavily on human control to adjudicate among flexible, redundant systems. Most critically, the crew would monitor the primary and backup systems and watch for indications of failure on one or the other. Nevins illustrated his philosophy with a detailed exploration of crew tasks during the final phases of landing, which we will examine in the following chapters. His approach amplified Chilton's original vision of astronauts as redundant backups, and placed a heavy burden on the crews in training and workload, especially when something went wrong. Nevins noted that the Apollo spacecraft had 448 switches and indicators, as opposed to 150 on Gemini and 102 on Mercury. Much of the crew's efforts would resemble operating a telephone switchboard as they flipped switches and actuated valves to configure the spacecraft for various phases of flight.

Ironically, Nevins seemed almost apologetic that the astronauts had to be so busy, explaining that only "technological gaps" required their involvement at all. Ultimately improvements in computers would automate the monitoring tasks and allow the human role to become "more purely administrative, or supervisory." Future machines would "relieve man of the necessity to play such an extensive role in either piloting or supervising his vehicle" and would allow humans to spend more time doing scientific experiments or exploration.[58] Of course, if the crews were no longer involved in the flying, NASA might send different kinds of people, perhaps scientists instead of test pilots.

Display and Keyboard

"How do you take a pilot, put him in a space ship, and have him talk to a computer?" astronaut David Scott succinctly put the question facing the IL. Nevins and his team embodied their philosophy in the interface to the Apollo computer. They developed a "display and keyboard" unit, abbreviated DSKY (pronounced "dis-key"). Somewhat akin to an early calculator display, the DSKY had a numeric keyboard with plus and minus keys, and seven additional function keys like ENTER, CLEAR, and KEY RELEASE (figure 7.8). The display included three signed numbers for numerical data like navigation coordinates, and three shorter numbers to identify functions. Each line of display was merely five numerals, with plus or minus but no decimal point (similar to how a slide rule displayed its results). Astronauts needed to know independently, for example, that time would be displayed in three digits of seconds with two digits of hundredths, while gyro angles would be displayed in two digits of degrees plus three digits of thousandths. A series of warning and indicator lights (including the feared GIMBAL LOCK)

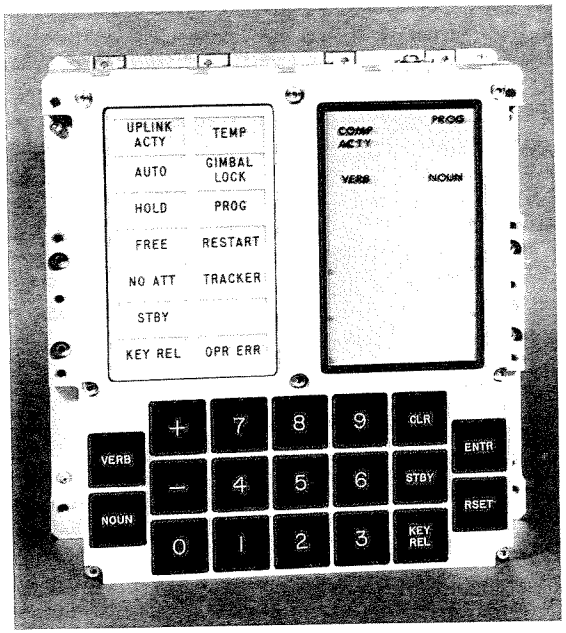


Figure 7.8
DSKY—display and keyboard unit—for the Apollo guidance computer. Note the space for digital displays on the right, with program, verb, and noun indicators, and three line numerical display below. Also note status lights on left, including warnings for gimbal lock, program alarm, and operator error. (Raytheon photo CN-4-268.)

signaled anomalies or faults. The displays used the "seven segment" format for alphanumeric data. While the style did not originate with Apollo, its use in the AGC foreshadowed the ubiquitous LED displays of the 1970s and helped make the boxy digits icons of the digital era. In the computer, a routine called PINBALL GAME BUTTONS AND LIGHTS ran the DSKY interface (see cover image).

Alonso began thinking about the command protocol: "It occurred to me that the sort of dialog between astronaut and AGC could fit into a rudimentary sentence structure, such as 'Display IMU Angles,' or 'Display Time,' or 'Fire Rocket' or 'Align IMU.'" Working with Albert Hopkins and Herb Thaler, Alonso developed a temporary design they could use for testing and demonstration while waiting for an official solution. Their stopgap became permanent.[59]

Astronauts entered data and commanded the DSKY with this "verb-noun" syntax. To enter a command, the astronaut would press the VERB button followed by a two-digit code for a command. Then he would push NOUN and the code for a particular type of data, and then ENTER. Noun 37, for example, referred to roll, pitch, and yaw data and Noun 89 referred to the coordinates of a particular landmark. The verb and noun codes appear on the display as they are entered. If the command requires further data to be entered, the verb-noun display will flash, as the program awaits further data entry on the keypad. It displays data in one of the five-digit numerical display lines. Each data number is then followed by depressing ENTER. Various illegal combinations of keys are rejected by the DSKY as invalid data. If an incorrect key is pressed, the CLEAR button allows the data to be reentered. The interface was a bit cumbersome and not simple. A task like aligning the inertial platform would require anywhere from thirty to one hundred-thirty keystrokes; an entire flight to the moon would take more than ten thousand keystrokes.

Certain verbs simply displayed data. Verb 01, for example, would display a selected value on the display, and Verb 11 would "monitor" that value, that is, update the display once per second. Other verbs ran specific guidance programs, such as Verb 41 for "Coarse align IMU," or Verb 46 for "Activate digital autopilot." To some degree, the astronauts could customize the displays for their own preferences, instructing the DSKY to display particular variables on particular lines. Any verb-noun combination, or any mix of DSKY keystrokes could also be entered from the ground, via the radio telemetry link (although the astronauts had a switch that could lock out remote control). The ground controllers would frequently update the "state vector," or the onboard computer's knowledge of its position and velocity, through the telemetry channel.[60] One of the ground controllers even had a DSKY display on his console and could push the buttons directly to command the onboard computer. Thus any DSKY command could be entered by the astronauts or from the ground.

While the astronauts could command the computer through the DSKY, the computer could also command them. Occasionally the display would flash a command, asking them to perform a certain item in a checklist.

As an example, Verb 37, "Change major mode," selected a variety of operating programs, depending on particular phases of flight. By pressing VERB and then "37" and then ENTER, the display would flash, after which the astronaut would enter a program number for a particular phase of flight. Program "00" (pronounced "poo") was the idling state for the computer. Additionally, programs starting with 0 related to preflight checkouts (e.g., P01 = "pre-launch initialization"), programs beginning with 1 performed boost monitoring (e.g., P11 = "earth orbit insertion monitor"), and the 2-series included "rendezvous navigation." In the command module, programs beginning with 6 specified reentry, whereas in the LM the 60-series programs controlled lunar landing. As we shall see in the next chapter, beginning with P63, "Braking phase,"

the LM computer cycled automatically through a series of programs until the astronauts would enter P68, for "landing confirmation."

IL staff remember having a difficult time getting their management and NASA to accept the verb-noun structure. "It's not serious enough," they recall hearing, along with the objections, "It's not military enough. It's not scientific enough."[61] Among the astronauts Nevins found two reactions: "One reaction is that we don't fly spacecraft through DSKYs, we don't fly planes through DSKYs." As for the other reaction, he said, "they appreciated immediately what they were looking at and that they had all this capability and power. That everything and anything that was dynamic in the spacecraft was under the control of the computer. And so those people appreciated it."

Nevins remembers having more difficulty with the earlier, Mercury astronauts. "And then we got our spies," people like David Scott who worked with Battin on guidance and received a master's degree in 1962; Charlie Duke, who received a master's degree in 1964 for studying human performance during Apollo navigation; or Ed Mitchell, Ph.D. 1964, who studied interplanetary guidance. These men learned how to think about guidance and control in the MIT way. "They all got brainwashed," as Nevins put it.[62]

Astronauts generally had good things to say about the DSKY on their missions, the result of long hours of training. Hoag recalled that "an early reticence by the crew members was in time replaced by enthusiasm and confidence in their ability to use the computer to manage many aspects of their mission."[63] Scott added sardonically, "It was so simple and straightforward that even pilots could learn how to use it."[64] Certain operations generated their own Apollo jargon, as when an astronaut aligned the inertial platform. The computer would display "residuals," or an indication of the quality of the alignment, in degrees. A display of "00000" indicated no difference, a perfect alignment, which the astronauts greeted with an exclamation of "five balls," a clear statement of success in the masculine culture.

Nevins's group put together a series of simulations, from simple jigs to evaluate sextant pointing to mockups to verify users' reach to the controls, to a full AGC interface in the Johnsville centrifuge. They also installed their systems' components in NASA's CSM and LM simulators.[65]

One day NASA headquarters called to inform Nevins that a formal complaint had been levied by the astronauts against the IL because the DSKY interface was too complex for a safe trip to the moon. Nevins responded by putting together a demonstration unit, called the space navigator, that would enable a user to "navigate" the earth as though it were a spaceship on its way to the moon, using "a physical marriage of the man, a complete G&N system, a real stellar environment, and a pseudo space craft motion generator." Adding some old radar-tracking hardware to swivel the simulator, the IL mounted it on the roof of their building as a demonstration and training device.

Nevins put together a three-hour training program for NASA managers, including Robert Mueller, Robert Seamans, and Chris Kraft, to teach the basics of the Apollo guidance, navigation, and control and train the astronauts. "They came to the conclusion," Nevins recalled, "that it's very complex, it's expensive to train—time and everything else—but it's doable."[66] A photo of Doc Draper in the machine with the Boston skyline and stars in the background became a popular image of the lab's participation in Apollo.

At MIT Scott had studied the statistical interpretation of celestial fixes for interplanetary navigation. When he became an astronaut he was assigned to monitor the MIT contract for the astronaut office. "I spent many nights up on a roof in Cambridge looking at the stars and working with a sextant, telescope and computer," Scott recalled. Some of his colleagues, he said, never got used to the keypunching, and requested programs that would combine the keystrokes into larger functions. Yet for Nevins, the turning point came when he observed Jim Lovell, Frank Borman, and Jim McDivitt operating the guidance system in a command module simulator at Kennedy Space Center shortly before their selection for the Apollo 8 mission. Lovell had mastered it, and was making it sing. "The system wasn't flying him," Nevins observed. "He was flying the system. And wow, he's flying."[67]

Change in Culture

For the first few years of the Apollo program, IL engineers defined tasks, built prototypes, conducted experiments, and did much of the creative, exciting part of flying to the moon. In the middle of 1966, however, everything changed. At the end of the Gemini program, Lickly recalled, "NASA descended on us."

"We awoke several years later realizing we had a big programming problem . . . and now we needed a new organization," Martin recalled. "Instead of having 30 people, we needed 200 people . . . we needed all this documentation. And we needed all these review meetings. And we needed all this NASA supervision." The whole team became "professionalized," Martin said. "We were going to have something called project managers." At NASA, Martin saw enormous management charts laid out on the walls, and "some contractor who would come up everyday with his blue tape or red tape or yellow tape and mark where the schedules were. . . . It was magnificent to see it." Martin seriously tried to adopt NASA's management methods but the increased oversight brought a new era to the lab, and "much less fun for a lot of people." NASA's managerial sophistication impressed a younger Ray Alonso differently: it was the first time he'd seen an electrically powered eraser.[68]

Part of the reason the programming proved so difficult was that writing serious, detailed requirements for the software amounted to defining the mission. Hoag felt

that not all of NASA's criticism was fair, as the programming was often hamstrung by lack of data, specifications, or procedures from elsewhere in the program. "It has turned out that MIT has had to do a large measure of the mission planning—how the test flights and lunar flights will be performed, in order to complete these tasks," Hoag said, and therefore the programming task "turned out to be far greater than originally estimated."[69]

NASA began considering turning creation of the actual mission programs over to one of the contractors, which would be a huge blow to the IL. Battin and Ralph Ragan successfully fought the move, and in mid-1966 NASA asked the IL to program not only the basic system, but the missions as well, a task that brought them intimately into the challenges of flight planning, including its severe time pressures. Feeling a late-Apollo budget squeeze, NASA also asked the IL to reduce its staffing and to transfer most of the less-technical work (training, ground support, etc.) to contractors.[70]

At this time the IL's software efforts received the attention of Bill Tindall, the most famous unknown player at NASA in the Apollo story. As part of the Mission Planning and Analysis Division (MPAD) in Houston, Tindall had been instrumental in the Mercury and Gemini programs, helping to turn the theory of orbital rendezvous into reality.[71] A true space enthusiast with a talent for clear, straightforward writing, Tindall began looking at MIT's software effort in early 1966 and he didn't like what he saw. His "Tindallgrams" became well known within NASA for their technical insight and blunt language. Some of the earliest concerned the Apollo software effort.

Tindall started making regular trips to the IL to see what was up. At first, nobody took him seriously. Martin remembered him as "an object of real derision.... We were rolling along doing all of this fantastic work building software, doing this, doing that, when NASA sort of somehow woke up and decided that these guys were totally out of control." NASA's concerns: the documentation was no good. There weren't any real schedules.[72] The IL group was having great fun, but was insufficiently attentive to testing. "Testing is the name of the game when you're playing with mission safety, critical kinds of systems," Jack Garman remembered, but the IL folks treated it like busy work and didn't understand why they had to write up numerous different test reports for NASA.[73]

Tindall began a devastating series of missives in May 1966. These Tindallgrams began with the alarm: "There are a number of us who feel that the computer programs will soon become the most pacing item for the Apollo flights." A strong statement. Previously, nobody had paid much attention to this novel thing called "software," the pet project of a bunch of academics up in Cambridge. Now software would make or break the schedule for the firm, end-of-the-decade deadline.

Tindall studied the organization, with Battin in charge and four subunits reporting to him. Yet he still complained: "I still do not have a clear understanding of how their

work is broken out between the four units." Tindall had brought some NASA engineers with him who were working with IBM on the Real-Time Computer Complex (RTCC), which did the heavy number crunching at Houston, and he hoped that MIT would learn from them and adopt their management techniques. Indeed the IL began assigning a person responsible for the program for a particular flight, for managing it through the entire process from coding through manufacture and test. In typical IL style, they called this person the "rope mother" (though it was usually a man).

Tindall was particularly worried about bloated program size, some of it brought on by an obsession with precision by the academically oriented IL engineers: "I am still very concerned about unnecessary sophistication in the program and the effects of this 'frosting on the cake' on schedule and [memory] storage. It is our intention to go through the entire program, eliminating as much of this sort of thing as possible. I am talking about complete routines, such as 'Computer Self-checks,' as well as little features, such as including the third and fourth harmonics of the earth's oblateness and drag in programs for the lunar mission."[74]

IL engineers all remember Friday the thirteenth of May 1966, "black Friday," when Tindall called everyone together to cut their favorite programs out of the software so it would fit into the limited memory. In the dry language of the official report, "these meetings became emotional because of disagreement about what was, in fact, nonessential."[75]

Two weeks later, Tindall was again writing, this time a memo titled "Apollo spacecraft computer programs—or, a bucket of worms." He began, "Well, I just got back from MIT with my weekly quota of new ulcers." Tindall was pleased that the rope mother for AS-204, the first manned mission, seemed to have adopted some of the IBM techniques (like a program development plan). Still, the mission was exceeding its allocated memory by as much as 500 bytes (at that time a large margin, or 2 percent). More distressing, the rope mother was trying to put together the total program without having tested all of the units individually. "Certainly a very unsatisfactory situation," Tindall remarked.

He concluded that the program for AS-204 "will be of less than desirable quality. It will not have undergone sufficient verification tests and will very likely still contain program bugs" when it flies into space. Tindall intended to put Houston people at MIT to watch over the IL group day by day, "with no alternative but to march along with our fingers crossed."

Programs for the later missions, especially the Block II and manned missions, were in even worse shape. "After recovering from our complete shock," about the schedule for AS-501 and AS-502, Tindall and NASA again concluded the programs would have to be ruthlessly culled and accelerated. Said Tindall: "The program paring must be done, I feel, solely for schedule reasons, which is really kind of weird when you think about

how long the programs have been under development. It will mean that we fly to the moon with a system which does not minimize fuel expenditure nor provides the close guidance tolerances which are ultimately within its capability."[76]

The eternal compromises of system engineering now compelled trading weight and fuel for schedule and memory capacity. Who would have thought that a few bytes of memory could compromise the accuracy of a moon landing? Who could have foreseen that imperfections in the programs would consume the scarce resource of fuel—programs nobody even knew they needed when Apollo began?

Over that summer, and for months afterward, the bad news continued to flow. There were not enough programmers, so the IL began hiring contractors. The number of people working on Apollo software began to rise steeply in the middle of 1966 and continued until shortly before the nearly Apollo 11 launch. Costs for the second half of 1966 were double those of the first half of the year, nearly a million dollars per month.[77] The IL did not have enough computers to support simulation on anything but the next mission, putting the efforts behind for subsequent flights. Rope mothers were assigned for each mission, and they each had to assemble a series of subroutines from the programming groups into working mission program. "In most cases," Lickly recalled, "the engineers did a so-so programming job and it had to be redone and put into a form that was all consistent for a particular flight."[78]

Despite the software turmoil, in August 1966 the unmanned, suborbital AS-202 flight tested a Saturn IB with a Block I command module aboard, the first flight of the Apollo guidance and navigation system. This was a high lob intended to give the CM heat shield a reentry test. It carried a Block I spacecraft and computer, with a "mission programmer," a set of relays to mimic the actions of an astronaut. The craft flew on all-inertial navigation for an hour and a half. The guidance system performed successfully, monitoring the launch, aligning the command module for firing of the service module's engine, and guiding the command module on its reentry trajectory.[79]

After this success, in September 1966 George Low wrote to Robert Gilruth of his concern that MIT was "very far behind in the preparation of software... the general conclusion is that we must put the MIT programming and scheduling on a more business like basis."[80] The program for the first manned mission, AS-204, caused the most serious problem. The crew had recommended a number of software changes to improve the flexibility of the data in their displays, but they would have to be left out because of schedule and testing requirements. Programs for at least the first four missions were late, too large, not reliably tested, and full of bugs. IBM and NASA programmers had developed ambitious plans for the ground computers to interact with the spacecraft, but these would have to go "straight in the trash can," in Low's words, because the flight vehicles would not be prepared to provide them adequate data. The IL was pushed to release a program that Kosmala remembered "was just so lousy, so full of bugs."

On a fateful January day in 1967, that program, and the three astronauts who were scheduled to fly it, burned up on the launch pad.

Recovering from the Fire

The tragedy of the Apollo 1 fire stemmed from a series of failures in the program, from quality control to configuration management, and from NASA's relationships with its contractors. It brought congressional hearings and a traumatic reevaluation across the Apollo program. The high-profile crisis it provoked concerning the larger Apollo spacecraft meant the software problems could be worked out below the glare of publicity. In the fallout, Joe Shea was forced out, and Chris Kraft rose to renewed prominence; his operations-oriented approach would dominate the remainder of the program. The balance of power shifted noticeably toward NASA Mission Control, toward Houston, toward the astronauts, and away from ballistic missile-style systems engineering.

None of this was lost on Bill Tindall. Barely two months after the fire, he was already acknowledging, "It is possible to take advantage of the stretch out of the Apollo flight schedule in the manner in which we develop the spacecraft computer programs at MIT." For Martin, "it was a pretty tense time," and he acknowledged that "we were not building that program in a way that was disciplined, and organized, and had traceability in it."[81] Battin, too, realized that he was too technical, too interested in the guidance itself to be the right kind of manager. "He [Tindall] really did apply good discipline to our shop, and we did learn how to get people. It was... a management technique which I, frankly, was not equipped to handle.... I wasn't really up to trying to direct the whole orchestra when I wanted to concentrate on a piece of it."[82]

Copps remembered the turning point. "One day Tindall just gave us hell.... He really beat us up. 'How can you possibly do this? Here you sit at the very center of the success or failure of this extremely important program. You're behind. Get it through your head you are fucking this up.'" By this time, however, the MIT group respected Tindall enough that they could hear his message.[83]

By March 1967, Tindall was calm for the first time, his remarks conveying a dramatic difference in tone. "It is my feeling that no major problem exists any longer in this area. MIT has an organization and facilities geared up to handle the workload in an orderly, professional, unharried manner."[84] It was just two months after the fire, but already things were looking up. The number of separate versions of the software was reduced. In principle, only two were needed—one for the CSM and one for the LM. But several others had been planned for earth-orbital flights, and several others when the intended programs were not ready. The flight schedule had slipped sufficiently that software was no longer the pacing item, nor was crew training, nor testing. Furthermore, the value of quality had pervaded the organization. Techniques like "code inspection," a fancy name for a human being closely reading a printout of code, had

proven effective in finding bugs. Still, Apollo software never flew completely bug-free; known problems remained in the code, which hopefully remained free of unknown errors.

In addition, there was a new realism about whether the programs could be prepared far in advance, or even whether they should be. "Instead of releasing the flight program for rope manufacture at the earliest possible date," Tindall directed, "we should release it at the latest possible date." Then any changes that might come along in testing would have the longest possible time to be included.[85] A "Software Verification Plan" clearly laid out the steps for verifying, simulating, testing, and qualifying any new programs and changes, its flowcharted procedures for organizations indistinguishable from computer programs.[86] While struggles continued into the fall of 1967, with delays and quality problems in software, and NASA continued to issue schedule emergencies, Tindall and the IL engineers found the software on increasingly solid footing.[87]

When Tindall retired, his colleagues wrote a poem for him. One of its five stanzas read:

Yes! Gemini was a hard act to follow!
But you did it again when we got to Apollo.
You became the world's foremost authority
On an agonizing process called data priority.
You said the onboard software was a GD bag of worms
And that was one of your milder terms.
You gave those programs a real thorough wash,
While MIT was out playing squash![88]

The Early Missions

To qualify Apollo for a lunar landing, the early missions had to test the systems. Basic hardware, performance profiles, reliability, abort modes, and a host of other parameters needed to be tried, measured, and analyzed. Guidance and navigation, of course, were critical to these tests. Could the astronaut control the spacecraft? Could the computer? Could the ground controllers? Could the crew track the stars and landmarks on the earth or moon? Early Apollo missions collected data on gyro drift, accelerometer performance, and ability of the inertal platform to keep stable track of position and velocity. They also began to refashion the pilots' relationship to their craft in ways that would mature during the landings themselves.

After the fire, the second unmanned flight, AS-501 (Apollo 4) was launched in November 1967, intending to fly into high orbit and reenter the earth to simulate a lunar return. An AGC controlled operations on board for five hours. It determined position and velocity, controlled numerous attitude maneuvers, and fired two major burns of

the big service propulsion system (SPS) engine on the service module. Ground tracking sent two state vector updates to the computer via telemetry, necessary because without people on board the vehicle could not take its own navigational fixes. The SPS engine burned for four and a half minutes, sending the vehicle into an orbit that would simulate a lunar return trajectory and reenter at 36,500 feet per second.

Unfortunately, a ground controller in Australia sent a turn-on command to the engine after the computer had already generated the command on board. Receiving the instruction, the AGC switched modes and began only taking commands from the ground. This necessitated ground control to issue the shutoff command to the engine, which it did, but 13.5 seconds late, resulting in a 200 feet per second overspeed on reentry, which actually created a more stringent test for the heat shield. Onboard computer control ended at 23,000 feet when the parachutes opened. Despite the extra reentry velocity, the computer controlled the landing to less than two miles from the aim point.

Apollo 5 flew in January 1968, an unmanned test of the LM with no command module. This was the first flight for the Grumman vehicle, the first time its engines would be powered in flight. It was also the first Block II computer flight, and hence the first use of the digital autopilot, using the SUNBURST program. About four hours after launch, the goal was to have the AGC aboard the LM fire its engine at 10 percent thrust for about thirty seconds, and then to go up to full power for twelve seconds.

When the LM began to fire its descent engine, the computer thought the ignition was late, and prematurely shut it down.[89] "Utter chaos took place in the Mission Control Center," recalled Jim Miller, who was rope mother for the flight. "Everybody was climbing all over everybody to find out what happened. Totally preventing anybody from finding out what happened." Houston sent a signal to turn off the computer altogether and assumed remote control, bypassing the computer. Miller thought that the mission controllers didn't understand the details of the software, or the subtleties and flexibility of running a software-controlled spacecraft. "I knew right away what was going on," Miller said. "Nobody had asked MIT anything. . . . They just knew better and took over." Miller suggested a way to correct the situation, but the flight directors decided otherwise, because it had never been tested in simulation.

Mission Control commanded a backup guidance system to take over, then issued the burn and the staging commands, and the mission succeeded. The LM turned around, fired its engine, and separated the descent stage. Then the LM's computer took over again, but it had not been informed that the descent stage was no longer there, so its stabilization system was calibrated for a much higher mass. The thrusters hissed and puffed, nearly going unstable.

The cause of the problem, though called a "software error," was actually an error of communications between organizations. "I had been told by the guy that wrote the descent-burn software that it had to have a very narrow window in the startup," Miller

recalled, and that there was a serious problem if the thrust didn't build up immediately.[90] But a series of pressurizations and valve closures had to occur before the engine even began to light up. The computer mistook this delay for a slow thrust buildup and shut the engine down just as it was firing up. Internally, NASA acknowledged that "the premature shutdown was the result of incomplete systems integration, and not the result of improper function of individual systems."[91] In this case, "incomplete integration" stood for a miscommunication between two individuals.

Publicly NASA used the incident to further a different agenda. Rather than acknowledging the lack of communication that led to the problem, they praised the advantages of having humans on board. "If men had been aboard to fly the vehicle," George Low told the press, "the flight test might have been a different story.... I don't like to fly unmanned missions... with hardware designed to carry men." Later Sam Philips blamed the trouble on "overly conservative computer programming."[92]

IL programmers now began to face the public and official misunderstanding of their work. Because software did so much to integrate the system and depended on so many technical and organizational interfaces, social and managerial problems could be obscured by blaming the code. "We who programmed the LM's computer hung our heads in disappointment," Don Eyles recalled, "and endured a public reaction that did not distinguish between a 'computer error' and a mistake in the data."[93] Miller did feel, however, that the incident "really shook up the Mission Control people who realized that their abilities to handle things was much lower than they thought." Computers were introducing new realities into spacecraft flight and mission control, generating tensions between the people who programmed and those who used the programs.

Apollo 6, the final unmanned mission, also aimed to simulate a lunar return. It had trouble on the way up: two engines in the Saturn booster's second stage cut off prematurely and could not be restarted, causing a more elliptical orbit than planned. Then the third stage could not be started to simulate a lunar trajectory insertion. The reentry came in too slow, at 32,800 not 36,000 feet per second. A known bug in the reentry software, which was not expected to be important at the intended higher speed, steered the capsule fifty miles short of its intended landing spot. Overall, the guidance and navigation worked well: the alignment remained under inertial control throughout the mission and the onboard state vector remained accurate to within two miles of ground tracking.[94]

Apollo 7 was the first manned Apollo flight and the first Block II spacecraft (no Block I flew with people in it). Most of the primary objectives related to the astronauts working with the computer to control the spacecraft. Of the flight's nine goals, the first one read, "Demonstrate GNCS [guidance-navigation-control system] performance." Others included demonstrating the inertial system's coarse and fine alignment, determining orbital parameters by earth landmark tracking, and exercising the attitude hold modes, both automatic and manual.[95]

Launched on October 11, 1968, Apollo 7 spent almost eleven days flying 164 orbits while running the SUNDISK program on its computer. The AGC fired the service propulsion system (SPS) rocket six times, using digital autopilot for steering, and the computer monitored the Saturn boost into orbit. The crew practiced a command module rescue of the lunar module, pretending the spent Saturn stage was the LM. They tracked the stage with the sextant, from which the computer estimated range. "This may have been the first in-flight use of Kalman estimation formulations," Hoag reported with pride. The astronauts turned the guidance and navigation system on and off several times and easily realigned the IMU with star sightings. During the flight, the computer experienced three "restarts," in response to "three abnormal procedures." These were likely due to erroneous keyboard entries, so the IL refined the logic that allowed the operators to cancel commands if they hit the wrong buttons. Some problems arose with visibility through the scanning telescope due to particles surrounding the spacecraft, and the more critical difficulty of finding good horizons for tracking so close to earth (where the atmosphere blurred the crisp horizon line). Reentry started manually, then switched over to automatic control, and the computer brought the capsule to within a mile of its aim point on the ocean (all subsequent entries were under automatic control).[96] Nevins remembered this flight as a major step in NASA's and the astronauts' faith in the computer and automation. "Performance exceeded our expectations and hopes," Hoag wrote to his team in a "how did we do" memo that would become a regular event.

While a technical success, Apollo 7 exposed tensions between the astronauts' ideal of mission control and those in Houston. Through much of the mission commander Wally Schirra proved uncooperative and at odds with flight controllers on the ground. "Wally was legitimately in command of the spacecraft," wrote his fellow crewmember Walter Cunningham, "but he attempted to expand that authority over the entire mission."[97] Whatever their control over the machine itself while in orbit, astronauts still had to contend within the larger organizational system, one with its own distribution of power. None of the Apollo 7 crew ever flew again.

Apollo 8: (Nearly) Autonomous Navigation

These early flights played dress rehearsals for Apollo 8, an inspiring triumph for the project overall and the IL team in particular. The mission originated in a last-minute plan. In the summer of 1968, George Low began to despair of reaching the moon during the decade. The lunar modules were chronically late. Waiting to test them would likely eliminate any chance of flying in 1968, leaving only one year to meet the goal. The first test flight kept slipping, right up to March 1969. Low came up with the bold idea of flying to the moon without a lunar module. Such a mission would collect valuable data and provide a public relations and morale boost. After conferring with his top managers and engineers, including Stark Draper, and a successful Apollo 7 flight, in

early November Low decided that Apollo 8 would fly to the moon; indeed it would orbit the moon, right around Christmas time.[98]

Mission objectives included tracking lunar landmarks to specify locations for future landing sites, calibrating the ground tracking from lunar distances, and evaluating the optical sighting systems on board. They gave pride of place to Battin's navigation scheme and the IL's computer. For much of the IL team, Apollo 8 would climax the project as the craft flew (nearly) autonomously to the moon, with great accuracy. In preparation for the mission, the *Boston Globe* assured its readers that "voyagers to the moon will not be mere passengers. Astronauts will not sit passively by while machines and controllers back on earth do all the work of celestial navigation."[99] Apollo 8 was to test the humans as well as the machines.

Apollo 8 launched from Kennedy Space Center on December 21, 1968, and orbited the moon ten times before splashing down after a six-day flight. When the Saturn rocket's third-stage engine burned to send Apollo 8 out of earth orbit toward the moon, Battin experienced the "longest and most thrilling moments of my life." On the way to the moon, guidance was so accurate that (according to the Battin decision scheme) four of the seven midcourse corrections were not even made, and the third required only a small, three-foot-per-second impulse from the reaction controls. Arriving at the moon, it was a tense and dramatic scene when the computer fired the engine, behind the moon, to enter lunar orbit. The plan called for an orbit of 60 by 170 miles. Actual lunar orbit turned out to be 60.5 by 169.1 miles, so accurate that at the IL, "the unavoidable emotional tension was broken with unrestrained cheers." IL engineers saw this move in particular as a validation of their entire approach to guidance, navigation, and automatic control.

The famous Christmas Eve moment of the crew reading from Genesis introduced the great drama of lunar flight to the public. More quietly, the inertial system and the computer functioned for the entire mission, and the crew exercised all of the guidance features using the COLOSSUS software for the first time in flight. Earlier programs for test flights contained earth-orbit code only, whereas now COLOSSUS handled two planetary bodies, two gravitational fields, and two coordinate systems. The digital autopilot controlled the slow rotation of the spacecraft for passive thermal control. Command module pilot Jim Lovell made numerous alignment checks and navigation sightings. Most of the optical alignments required updates of only a few hundredths of a degree. The computer maintained the state-vector autonomously for much of the flight. While it also uploaded state vectors from the ground, it kept the numbers in separate memory locations from the onboard solution, although the ground's numbers overwrote the onboard numbers before each maneuver.

Lovell realigned the inertial platform thirty times during the flight. Usually, he used the automatic pointing feature for realignment (P52); the computer pointed at the expected position of stars and landmarks and the user had only to mark the difference.

For position fixing, Lovell made over two hundred earth and lunar horizon sightings with the sextant while coasting to and from the moon. While tracking landmarks, the computer automatically kept the command module gently pitching over at 0.3 degrees per second. Thus as he orbited across the surface of the moon, the landmarks stayed in view while Lovell sighted them.[100] Far from "flying the spacecraft," on Apollo 8 the command module pilot spent most of his time tracking the stars, verifying his position in relation to vectors tracked from the ground.

Soon after emerging from behind the moon and heading back toward earth, Apollo 8 experienced one of the program's notable human errors. Lovell was sighting a star numbered 01 in the catalog. By mistake, he called up the program for pre-launch preparation. The missed keys did not cause the computer to do anything radical (it always asked for a PRO key press before firing an engine), but the confused program (trying to test for pre-launch while orbiting the moon) wrote to memory in ways that could have overwritten the reference matrix for navigation.[101] IL engineers spent a great deal of effort diagnosing the situation. Finally, they walked Houston and the crew through a check of the memory, which showed there was no deep problem. Still, Lovell's error had destroyed the inertial platform's alignment so he used the "coarse align" computer function to align the platform with the stars.

The computer fired the critical burn to leave lunar orbit and return, from the back side of the moon, using tracking data uploaded from the ground. It proved so accurate that only a single five-feet-per-second correction was later required to hit the earth's reentry corridor. Lovell made more than a hundred sightings on the return trip; ground tracking and onboard navigation proved "essentially identical." The automatically controlled touchdown was within one-third mile of the target.[102]

For engineer Aaron Cohen, the trans-earth injection of Apollo 8 "was the moment when I felt the program really had been its climax, because it really encompassed everything which I thought was complicated."[103] Astronaut Frank Borman called the navigation "absolutely miraculous." IL engineers considered Apollo 8 the great triumph of Apollo navigation because it worked so nearly autonomously (a number of engineers, in fact, left the IL right after this flight to form a company to write navigation software). The *Boston Herald* printed a picture of the IL engineers celebrating at an Italian restaurant in Boston's North End.[104] *Aviation Week and Space Technology*, the main industry publication, ran a headline bound to warm the hearts of the IL engineers: "Apollo 8 proves value of onboard control," and peppered its reporting with descriptions of the "accuracy" and "precision" of the lunar flight.[105]

The Rise of Software

Apollo 8's triumphant navigation raises a question: if the state vector on board was so accurate, essentially identical with the data from the ground, why did Houston insist

on overwriting it with the data tracked and computed from the ground? Hoag, and no doubt many of the IL engineers, felt that the entire mission could have been accomplished using the onboard navigation (indeed, simulations they later ran with only Lovell's onboard sightings showed that the mission would have easily hit its reentry corridor with onboard data alone). But just as the astronauts were enmeshed in a system that included power relations within and between large organizations, so were the computers and the IL. Once onboard navigation was demoted to the status of backup (as the astronauts had been years before), overwriting of its data symbolically enacted the new relationship: Houston had control. Battin was tremendously proud that, in his view, Lovell had navigated himself to the moon and back. "Sadly," he lamented, "they never did that again."[106]

Apollo software, from an element of the system little understood, indeed barely envisioned, when the program began, became central to the mission, mediating much of the astronauts' critical interactions with their machine. It incorporated information from a variety of sensors around the spacecraft, tying the complex system, with its diverse equipment made by varying contractors, into a coherent whole. It unified the work of a team of programmers, all working on subsets of the problem, into a unitary "mission." It tied the astronauts into the craft, recording and analyzing their navigation and control inputs into stable-state vectors. No wonder this critical, invisible component of the project proved so complex to create, unruly to manage, difficult to test, and hard to trust. In its unusual combination of virtual plan and handmade core ropes, Apollo's software embodied its missions, and embedded assumptions, fears, and social relationships. On no phase of the mission would software, and human action, prove as central, or as ambiguous, as on the lunar landing itself.

8 Designing a Landing

We are banking our whole program on a fellow not making a mistake on his first landing.
—Pete Conrad

This book began with a detailed description of the Apollo 11 landing, focusing on the human-machine interaction. We now revisit the landings as the culmination of the debates over pilots' roles, computer engineering, software, and human abilities. This focus necessarily excludes a great deal of the Apollo flights, but the landings represented critical moments of each mission. Neil Armstrong described them as "the hardest for the system and hardest for the crews." On a scale of one to ten, Armstrong rated walking around on the moon a one, whereas "the lunar descent on a ten scale was probably a thirteen" (he noted the speed, range, and altitude covered by the LM during its descent was similar to that of an X-15 flight).[1] Such trepidation, he would find, was well justified.

An Apollo flight encompassed hundreds of complex operations, but none were as demanding, time-critical, and plagued with uncertainties as the landing, executed in extreme conditions of darkness and cold, far from home. The landing trajectory had to accommodate a wide range of factors, from constraints on the systems' performance to the abilities of the crew and the idiosyncrasies of communications. Even the very calendar of the mission, the "launch window" of a few days per month, derived from the requirement for appropriate lighting conditions on the moon so the pilots could see while they were touching down.

The lunar landings played a microcosm of the entire Apollo program in dramatic ten-minute phases. Here the tensions between human and machine, between manual and automated, between pilots as controllers and pilots as systems managers manifested themselves in a string of life- and mission-critical operations, some smooth and some surprising. Engineers with computers and slide rules analyzed and scheduled every minute, and virtually every foot, of the final descent to the lunar surface. Constant testing and negotiation defined the human role and how it would trade off with the computer. Astronauts repeatedly practiced the intense moments prior to landing