2

Computers on Board the Apollo Spacecraft

THE NEED FOR AN ON-BOARD COMPUTER

The Apollo lunar landing program presented a tremendous managerial and technical challenge to NASA. Navigating from the earth to the moon and the need for a certain amount of spacecraft autonomy dictated the use of a computer to assist in solving the navigation, guidance, and flight control problems inherent in such missions. Before President John F. Kennedy publicly committed the United States to a "national goal" of landing a man on the moon, it was necessary to determine the feasibility of guiding a spacecraft to a landing from a quarter of a million miles away. The availability of a capable computer was a key factor in making that determination.

The Instrumentation Laboratory of the Massachusetts Institute of Technology (MIT) had been working on small computers for aerospace use since the late 1950s. Dr. Raymond Alonso designed such a device in 1958–1959[1]. Soon after, Eldon Hall designed a computer for an unmanned mission to photograph Mars and return[2]. That computer could be interfaced with both inertial and optical sensors. In addition, MIT was gaining practical experience as the prime contractor for the guidance system of the Polaris missile. In early 1961, Robert G. Chilton at NASA–Langley Space Center and Milton Trageser at MIT set the basic configuration for the Apollo guidance system[3]. An on-board digital computer was part of the design. The existence of these preliminary studies and the confidence of C. Stark Draper, then director of the Instrumentation Lab that now bears his name, contributed to NASA's belief that the lunar landing program was possible from the guidance standpoint.

The presence of a computer in the Apollo spacecraft was justified for several reasons. Three were given early in the program: *(a)* to avoid hostile jamming, *(b)* to prepare for later long-duration (planetary) manned missions, and *(c)* to prevent saturation of ground stations in the event of multiple missions in space simultaneously[4]. Yet none of these became a primary justification. Rather, it was the reality of physics expressed in the 1.5-second time delay in a signal path from the earth to the moon and back that provided the motivation for a computer in the lunar landing vehicle. With the dangerous landing conditions that were expected, which would require quick decision making and feedback, NASA wanted less reliance on ground-based computing[5]. The choice, later in the program, of the lunar orbit rendezvous method over direct flight to the moon, further justified an on-board computer since the lunar orbit insertion would take place on the far side of the moon, out of contact with the earth[6]. These considerations and the consensus among MIT people that autonomy was desirable ensured the place of a computer in the Apollo vehicle.

Despite the apparent desire for autonomy expressed early in the program, as the mission profile was refined and the realities of building

the actual spacecraft and planning for its use became more immediate, the role of the computer changed. The ground computers became the prime determiners of the vehicle's position in three-dimensional space "at all times" (except during maneuvers) in the missions[7]. Planners even decided to calculate the lunar orbit insertion burn on the ground and then transmit the solution to the spacecraft computer, which somewhat negated one of the reasons for having it. Ultimately, the actual Apollo spacecraft was only autonomous in the sense it could return safely to earth without help from the ground[8].

Even with its autonomous role reduced, the Apollo on-board computer system was integrated so fully into the spacecraft that designers called it "the fourth crew member"[9]. Not only did it have navigation functions, but also system management functions governing the guidance and navigation components. It served as the primary source of timing signals for 20 spacecraft systems[10]. The Apollo computer system did not have as long a list of responsibilities as later spacecraft computers, but it still handled a large number of tasks and was the object of constant attention from the crew.

MIT CHOSEN AS HARDWARE AND SOFTWARE CONTRACTOR

On August 9, 1961, NASA contracted with the MIT Instrumentation Lab for the design, development, and construction of the Apollo guidance and navigation system, including software. The project manager for this effort was Milton Trageser, and David Hoag was the technical director[11]. MIT personnel generally agree that they were chosen because their work on Polaris proved that they could handle time, weight, and performance restrictions and because of their previous work in space navigation[12]. In fact, the Polaris team was moved almost intact to Apollo[13]. Despite their experience with aerospace computers, the Apollo project turned out to be a genuine challenge for them. As there were no fixed specifications when the contract was signed, not until late 1962 did MIT have a good idea of Apollo's requirements[14]. One of the MIT people later recalled that

> If the designers had known then [1961] what they learned later, or had a complete set of specifications been available. . . they would probably have concluded that there was no solution with the technology of the early 1960s[15].

Fortunately, the technology improved, and the concepts of computer science applied to the problem also advanced as MIT developed the system.

NASA's relationship with MIT also proved to be educational. The Apollo computer system was one of NASA's first real-time, large scale software application contracts[16]. Managing such a project was completely outside the NASA experience. A short time after making the Apollo guidance contract, NASA became involved in developing the on-board software for Gemini (a much smaller and more controllable enterprise) and the software for the Integrated Mission Control Center. Different teams that started within the Space Task Group, later as part of the Manned Spacecraft Center in Houston, managed these projects with little interaction until the mid-1960s, when the two Gemini systems approached successful completion and serious problems remained with the Apollo software. Designers borrowed some concepts to assist the Apollo project. In general, NASA personnel involved with developing the Apollo software were in the same virgin territory as were MIT designers. They were to learn together the principles of software engineering as applied to real-time problems.

THE APOLLO COMPUTER SYSTEMS

The mission profile used in sending a man to the moon went through several iterations in the early 1960s. For a number of reasons, planners rejected the direct flight method of launching from the earth, flying straight to the moon, and landing directly on the surface. Besides the need for an extremely large booster, it would require flawless guidance to land in the selected spot on a moving target a quarter of a million miles away. A spacecraft with a separate lander would segment the guidance problem into manageable portions. First, the entire translunar spacecraft would be placed in earth orbit for a revolution or two to properly prepare to enter an intercept orbit with the moon. Upon arriving near the moon, the spacecraft would enter a lunar orbit. It was easier to target a lunar orbit window than a point on the surface. The lander would then detach and descend to the surface, needing only to guide itself for a relatively short time. Alter completion of the lunar exploration, a part of the lander would return to the spacecraft still in orbit and transfer crew and surface samples, after which the command module (CM) would leave for earth.

With a lunar orbit rendezvous mission, more than one computer would be required, since both the CM and the lunar excursion module (LEM) needed on-board computers for the guidance and navigation function. The CM's computer would handle the translunar and transearth navigation and the LEM's would provide for autonomous landing, ascent, and rendezvous guidance.

NASA referred to this system with its two computers, identical in design but with different software, as the Primary Guidance, Navigation,

and Control System (PGNCS, pronounced "pings"). The LEM had an additional computer as part of the Abort Guidance System (AGS), according to the NASA requirement that a first failure should not jeopardize the crew. Ground systems backed up the CM computer and its associated guidance system so that if the CM system failed, the spacecraft could be guided manually based on data transmitted from the ground. If contact with the ground were lost, the CM system had autonomous return capability. Since the lunar landing did not allow the ground to act as an effective backup, the LEM had the AGS to provide backup ascent and rendezvous guidance. If the PGNCS failed during descent, the AGS would abort to lunar orbit and assist in rendezvous with the CM. It would not be capable of providing landing assistance except to monitor the performance of the PGNCS. Therefore the computer systems on the Apollo spacecraft consisted of three processors, two as part of the PGNCS and one as part of the AGS.

EVOLUTION OF THE HARDWARE:

Old Technology versus New: Block I and Block I Designs

The computer envisioned by MIT's preliminary design team in 1961 was a shadow of what actually flew to the moon in 1969. There always seem to be enough deficiencies in a final product that the designers wish they had a second chance. In some ways the Apollo guidance computer was a second chance for the MIT team since most worked on the Polaris computer. That was MIT's most ambitious attempt at an "embedded computer system," a computer that is intrinsic to a larger component, such as a guidance system. Although the Apollo computer started out to be quite similar to Polaris, it evolved into something very different. The Apollo guidance computer had two flight versions: Block I and Block II. Block I was basically the same technology as the Polaris system. Block II incorporated new technology within the original architecture.

Several factors led from the Block I design to Block II. NASA's challenges to the MIT contract and the decision to use the rendezvous method instead of a direct ascent to the moon were decisive. A third factor related to reliability. Finally, the benefits of the new technology influenced the decision to make Block II.

Before NASA let the contract to MIT, but after it was known that the Instrumentation Laboratory would be accorded "sole source" status, several NASA individuals began studying ways to consolidate flight computer development. In June 1961, Harry J. Goett of Goddard Space Flight Center recommended that the computers needed for the Orbiting Astronomical Observatory (OAO), Apollo, and the Saturn launch vehicle be the same. He cited an IBM proposal for $5 million to do

just that[17]. On the same day Goett's recommendation, RCA proposed the use of a 420-cubic-inch computer with only an 80-watt power consumption and 24-bit word size as a general-purpose spaceborne computer[18]. This proposal got nowhere and NASA's Robert G. Chilton challenged Goett's idea, showing that the expected savings would not materialize. Even though the projected cost of the Apollo computer would decrease to $8 million from $10 million, the OAO development costs would rise from $1.5 million to $5 million[19]. Ironically, in the same month, Ramon Alonso from MIT met with Marshall Space Flight Center personnel about the use of the Apollo computer in the Saturn[20]. Although MIT got the Apollo contract and IBM got the contract for the Saturn computer, the idea of a duplicate system did not die. Two years later, when the deficiencies of the Polaris-based system were obvious and the solutions offered by the new technology of the Block II version still unproved, David W. Gilbert, NASA manager for Apollo guidance and control, proposed replacing the MIT machine with the one IBM was building for Saturn[21]. It did not occur because Gilbert wanted NASA to accept the reprogramming costs, and the existing configuration of the IBM computer would not fit in the space allotted for it in the CM. Nevertheless, MIT would still have to deal with NASA misgivings about the hardware design as late as May 1964, when Maj. Gen. Samuel C. Phillips, deputy director of the Apollo Program, reported on a meeting to discuss the use of the triple modular redundant Saturn launch vehicle computer in Apollo[22].

The decision to have a separate CM and the LEM influenced the transition to Block II by providing a convenient dividing point in the Apollo program. The early Apollo development flights were to use the CM only. Later flights would include the LEM. Since Block I design and production had already proceeded, planners decided to use the existing Block I in the unmanned and early manned development flights (all relatively simple earth-orbital missions) and to switch to Block II for the more complex combined CM–LEM missions[23].

Reliability was another force behind Block II. During early planning for the guidance system, redundancy was considered a solution to the basic reliability problem. Designers thought that two computers would be needed to provide the necessary backup; however, they dropped this scheme for two reasons. The ground had primary responsibility for determining the slate vector (the position of the craft in three-dimensional space) in translunar, lunar orbit, and transearth flight[24]. Moreover, none of the variations of the two-computer or other redundancy schemes could meet the power, weight, and size requirements[25]. One way to provide some measure of protection is to make the computer repairable in flight. The Block I design, due to its modularity, could be fixed during a mission that carried appropriate spares. At any rate, its predicted mean time between failures (MTBF) was 4,200 hours, about 20 times longer than the longest projected

mission[26]. But Block I's repair capability became a negative factor when sealing the computer began to be considered more important to reliability than the ability to repair it[27]. Aside from packaging, overall malfunction detection was improved in the Block II design, further increasing reliability[28].

The most important reason for going to Block II was the availability of new technology. The Block I design used core transistor logic. It had several disadvantages:

- It could not be complemented, a very important basic operation in computer arithmetic that changes a one to a zero or vice versa.
- It had the characteristic of "destructive readout," in which a datum read from a flip-flop using core transistor logic loses the datum; that forces the inclusion of a circuit to rewrite the datum if it is to be retained after the read cycle.
- Memory cycle time could not be fixed: in Block I it was an average of 19.5 milliseconds, which was quite slow for computers at the time, and the varying cycle caused timing problems within the machine[29].

These disadvantages led MIT to begin studying, as early as 1962, the possible use of integrated circuits (ICs) to replace core transistor circuits. ICs, so ubiquitous today, were only 3 years old then and thus had little reliability history. It was therefore difficult to consider their use in a manned spacecraft without convincing NASA that the advantages far outweighed the risks.

To accomplish this, the MIT team chose a direct-coupled transistor logic (DCTL) NOR gate with a three-input element,[30] consisting of three transistors and four resistors. NOR logic inverts the results of applying a Boolean OR operation to the three inputs. It took nearly 5,000 of these simple circuits to build an Apollo computer. Using a variety of circuits would have simplified the design since the component count would have been reduced, but by using the NOR alone, overall simplicity and reliability increased[31]. Also, the time it took the machine to cycle became fixed at 11.7 milliseconds, a double bonus in that speed increased and cycle time was consistent[32].

Aside from these advantages, MIT believed that the lead time to the first flight would permit reliability to be established and the cost of the ICs to come down[33]. At the time, the production of such circuits was low, and they were more expensive than building core transistor circuits. To place the production rate in perspective, MIT chose the NOR ICs in the fall of 1962 and by the summer of 1963, 60% of the total U.S. output of microcircuits was being used

in Apollo prototype construction[34]. This is one of the few cases in which NASA's requirements acted as a direct spur to the computer industry. When MIT switched to ICs, it kept the Apollo computer as "state of the art" at least during its design stage. It would be hopelessly outdated technologically by the time of the lunar landing 7 years later, but in 1962, using the new microcircuits seemed to be a risk. This view is contested by one member of the MIT team, who later said that the decision "wasn't bold; it was just the easy thing to do to get the size and power and other requirements"[35].

With the ICs fully incorporated in the Apollo computer and the transition from Block I to Block II complete, NASA possessed a machine that was more up to date technologically. It had double the memory of the largest Block I, more I/O capability, was smaller, and required less power[36]. Besides, it was also more reliable, which was, as always, the major consideration.

THE APOLLO GUIDANCE COMPUTER: HARDWARE

Overall Configuration and Architecture

The Apollo Guidance Computer was fairly compact for a computer of its time. The CM housed the computer in a lower equipment bay, near the navigator's station. Block II measured 24 by 12.5 by 6 inches, weighed 70.1 pounds, and required 70 watts at 28 volts DC. The machine in the lunar module was identical.

Crew members could communicate with either computer using display and keyboard units (DSKY, pronounced "disky"). Two DSKYs were in the CM, one on the main control panel and one near the optical instruments at the navigator's station. In addition, a "mark" button was at the navigator's station to signal the computer when a star fix was being taken. A single DSKY was in the lunar module. The DSKYs were 8 by 8 by 7 inches and weighed 17.5 pounds. As well as the DSKYs, the computer directly hooked to the inertial measurement unit and, in the CM, to the optical units.

The choice of a 16-bit word size was a careful one. Many scientific computers of the time used 24-bit or longer word lengths and, in general, the longer the word the better the precision of the calculations. MIT considered the following factors in deciding the word length: *(a)* precision desired for navigation variables, *(b)* range of input variables, and *(c)* the instruction word format[37]. Advantages of a shorter word are simpler circuits and higher speeds, and greater precision could be obtained by using multiple words[38]. A single precision word of data consisted of 14 bits, with the other 2 bits as a sign bit (with a one indicating negative) and a parity bit (odd parity). Two adjacent words yielded "double precision" and three adjacent,

"triple precision." To store a three-dimensional vector required three double precision words[39]. Data storage was as fractions (all numbers were less than one)[40]. An instruction word used bits 15–13 (they were numbered descending left to right) as an octal operation code. The address used bits 12–1. Direct addressing was limited, so a "bank register" scheme (discussed below) existed to make it possible to address the entire memory[41].

The Apollo computer had a simple packaging system. The computer circuits were in two trays consisting of 24 modules. Each module had two groups of 60 flat packs with 72-pin connectors. The flatpacks each held two logic gates[42]. Tray A held the logic circuits, interfaces, and the power supply, and tray B had the memory, memory electronics, analog alarm devices, and the clock, which had a speed of one megahertz[43]. All units of the computer were hermetically sealed[44]. The memory in Block II consisted of a segment of erasable core and six modules of core rope fixed memory. Both types are discussed fully below.

The Apollo computer used few flip-flop registers due to size and weight considerations[45], but seven key registers in the computer did use flip-flops:

- The accumulator, register 00000, referenced as "A".
- The lower accumulator, 000001, L".
- The return address register, 000002, "Q".
- The erasable bank register, 000003, "EB".
- The fixed bank register, 000004, "FB".
- The next address, 000005, "Z".
- The both bank register, 000006, "BB" (data stored in EB and FB were automatically together here)[46].

The use of bank registers enabled all of the machine's memory to be addressed. The largest number that can be contained in 12 bits is 8,192. The fixed memory of the Apollo computer contained over four times that many locations. Therefore, the memory divided into "banks" of core, and the addressing could be handled by first indicating which bank and then the address within the bank. For example, taking the metaphor "address" literally, there are probably hundreds of "100 Main Street" addresses in any state, but by putting the appropriate city on an envelope, a letter can be delivered to the intended 100 Main Street without difficulty.

The computer banks were like the cities of the analogy. The erasable bank register held just 3 bits that were used to extend the direct addressing of the erasable memory to its "upper" region, and

the fixed bank register held 5 bits to indicate which core rope bank to address. In addition, for the addresses needing a total of 16 bits, a "super bank bit" could be stored and concatenated to the fixed bank data and the address bits in the instruction word[47]. This scheme made it possible to handle the addressing using a 16-bit word, but it placed a greater burden on the programmers, who, in an environment short of adequate tools, had to attend to setting various bit codes in the instructions to indicate the use of the erasable bank, fixed bank, or super bank bit. Although this simplified the hardware, it increased the complexity of the software, an indication that the importance of the software was not fully recognized by the designers.

To further reduce size and weight, the Apollo computer was designed with a single adder circuit, which the computer used to update incremental inputs, advance the next address register, modify specified addresses, and do all the arithmetic[48]. The adder and the 16 I/O channels were probably the busiest circuits in the machine.

Memory

The story of memory in the Apollo computer is a story of increasing size as mission requirements developed. In designing or purchasing a computer system for a specific application, the requirements for memory are among the most difficult to estimate. NASA and its computer contractors have been consistently unable to make adequate judgments in this area. Apollo's computer had both permanent and erasable memory, which grew rapidly over initial projections.

Apollo's computer used erasable merry cells to store intermediate results of calculations, data such as the location of the spacecraft, or as registers for logic operations. In Apollo, they also contained the data and routines needed to ready the computer for use when it was first turned on. Fixed memory contained programs that did not need to be changed during the course of a mission. The cycle times of the computer's memories were equal for simplicity of operation[49].

MIT's original design called for just 4K words of fixed memory and 256 words of erasable (at the time, two computers for redundancy were still under consideration)[50]. By June 1963, the figures had grown to 10K of fixed and 1K of erasable[51]. The next jump was to 12K of fixed, with MIT still insisting that the memory requirement for an autonomous lunar mission could be kept under 16K[52]! Fixed memory leapt to 24K and then finally to 36K words, and erasable memory had a final configuration of 2K words.

Lack of memory caused constant and considerable software de-

velopment problems, despite the increase of fixed memory 18 times over original estimates and erasable memory 16 times. Part of the software difficulties stemmed from functions and features that had to be dropped because of program size considerations, and part because of the already described addressing difficulties. If the original designers had known that so much memory would be needed, they might not have chosen the short word size, as a 24-bit word could easily directly address a 36K bank, with enough room for a healthy list of instruction codes.

One reason the designers underestimated the memory requirements was that NASA did not provide them with detailed specifications as to the function of the computer. NASA had established a need for the machine and had determined its general tasks, and MIT received a contract based on only a short, very general requirements statement in the request for bid. The requirements started changing immediately and continued to change throughout the program. Software was not considered a driving factor in the hardware design, and the hardware requirements were, at any rate, insufficient.

The actual composition of the memory was fairly standard in its erasable component but somewhat unique in its fixed component. The erasable memory consisted of coincident-current ferrite cores similar to those on the Gemini computer, and the fixed memory consisted of core rope, a high-density read-only memory using cores of similar material composition as the erasable memory but of completely different design. MIT adopted the use of core rope in the original Mars probe computer design and carried it over to the Apollo[53]. Chief advantage of the core rope was that it could put more information in less space, with the attendant disadvantages that it was difficult to manufacture and the data stored in it were unchangeable once it left the factory (see Box 2-1).

Box 2-1: Core Rope: A Unique Data Storage Device

Each core in an erasable memory could store one bit of information, and each core in the core rope fixed memory could store four *words* of information. In the erasable memory, cores are magnetized either clockwise or counterclockwise, thus indicating the storage of either a one or a zero. In fixed memory, each core functions as a miniature transformer, and up to 64 wires (four sets of 16-bit words) could be connected to each core. If a wire passed through a particular core, a one would be read. If a particular wire bypassed the core, a zero would be read. For example, to store the data word 1001000100001111 in a core, the first, fourth, eighth, and thirteenth through sixteenth wires would pass through that core, the rest would bypass it. A 2-bit select code would identify which of the four words on a core was being read, and the indicated 16 bits would be sent to the appropriate register[54]. In this way, up to 2,000 bits could be stored in a cubic inch[55].

The computer contained core rope arranged in six modules, and each module contained 6,144 16-bit words[56]. The modules further divided into "banks" of 1,024 words. The first two banks were called the "fixed-fixed memory" and could be directly addressed by 12 bits in a instruction word. The remaining 34 were addressable as described in the text, using the 5-bit contents of the fixed bank register and the 10 bits in a instruction word[57].

The use of core rope constrained NASA's software developers. Software to be stored on core rope had to be delivered months before a scheduled mission so that the rope could be properly manufactured and tested. Once manufactured, it could not be altered easily since each sealed module required rewiring to change bits. The software not only had to be finished long in advance, but it had to be perfect.

Even though common sense indicates that it is advantageous to complete something as complex and important as software long before a mission so that it can be used in simulators and tested in various other ways, software is rarely either on time or perfect. Fortunately for the Apollo program, the nature of core rope put a substantial amount of pressure on MIT's programmers to do it right the first time. Unfortunately, the concept of "bug"-free software was alien to most programmers of that era. Programming was a fully iterative process of removing errors. Even so, many "bugs" would carry over into a delivered product due to unsophisticated testing techniques. Errors found before a particular system of rope was complete could be fixed at the factory[58], but most others had to be endured. Raytheon, the subcontractor that built the ropes, could eliminate hardwiring errors introduced during manufacture by testing the rope modules against the delivery tape of the programs. The company built a device to do

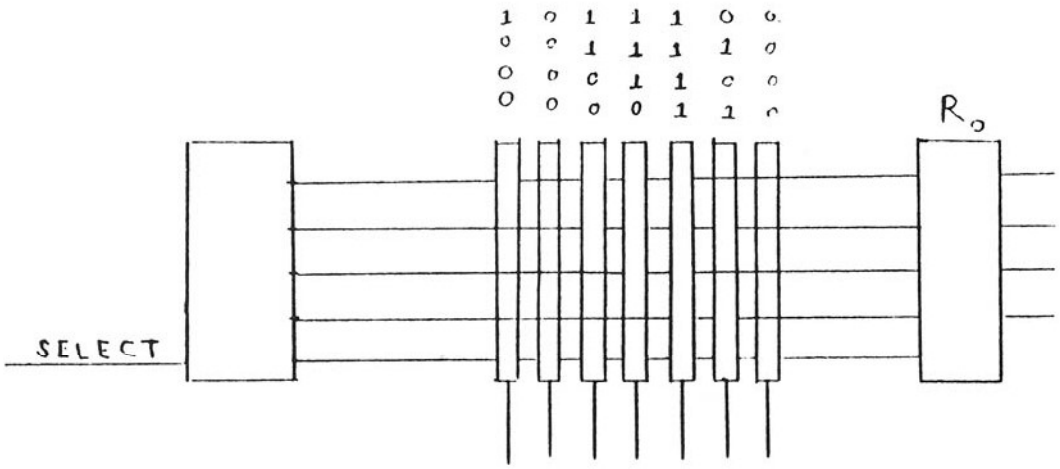

Figure 2–1. This diagram shows the principle behind core rope. Suppose that the data shown above the cores in the drawing is to be stored in the specific core. Thus 1000 is stored in the first core on the left by attaching the top wire from the select circuit to the core and bypassing it with the next three wires. When that core is selected for reading, the wire attached to the core will indicate a "one" because all cores in a rope are permanently charged as ones; the wires bypassing the core will indicate zeroes.

this[59].

Production Problems and Testing

Development and production of the Apollo guidance, navigation, and control system reflected the overall speed of the Apollo program. Design of the system began in the second quarter of 1961, and NASA installed a Block I version in a spacecraft on September 22, 1965. Release of the original software (named CORONA) was in January 1966, with the first flight on August 25, 1966[60]. Less than 3 years after that, designers achieved the final program objective. Even though fewer than two dozen spacecraft flew, NASA authorized the building of 75 computers and 138 DSKYs. Fifty-seven of the computers and 102 of the crew interfaces were of the Block II design[61]. This represents a considerable production for a special-purpose computer of the type used in Apollo. The need to quickly build high-quality, high-reliability computers taxed the abilities of Raytheon.

Through AC Electronic Circuits (contractor for the entire guidance system), Raytheon was chosen to build the computers MIT had designed largely because of its Polaris experience, but it had never built

a computer as complex as the one for Apollo. The Polaris machine was much simpler. Despite the use of experienced Polaris personnel, Raytheon's production division for the Apollo computer went from 800 to 2,000 employees in a year's time in order to handle the increased difficulties and speed of production[62].

Rapid growth, underestimation of production requirements, and reliability problems dogged Raytheon throughout the program. Changes in design made by MIT in late 1962 caused the company its initial trouble. The original request for proposal had featured Polaris techniques, so Raytheon bid low, expecting to use the same tools and production line for the Apollo machine. The changes in component types and memory size caused cost estimates to nearly double, resulting in considerable friction with NASA[63]. NASA was also worried when two computers and fully 50% of the Block I DSKYs failed vibration tests[64]. These failures turned out to be largely caused by contaminated flat packs and DSKY relays. Particles would shake loose during vibration testing[65]. The Block II computers would not work at first due to excessive signal propagation time in the micrologic interconnection matrix. The solution was to switch from nickel ribbon connectors to a circuit board, causing an increase of $500,000 in production costs[66].

These sorts of problems caused the Manned Spacecraft Center to authorize a complete design review of the AGC in February 1966. The lack of adequate support documentation was found to be the most significant fault of the Block II computer[67]. This sort of problem is usually the result of speeding up development to the point at which changes are not adequately documented.

Continuous and careful attention to reliability led to the discovery of problems. Builders flight-screened components lot by lot[68]. Post-production hardware tests included vibration, shock, acceleration, temperature, vacuum, humidity, salt fog, and electronic noise[69]. As D.C. Fraser, an engineer on the project, later remarked, "reliability of the Apollo computer was bought with money"[70].

THE APOLLO GUIDANCE COMPUTER: SOFTWARE

Development of the on-board software for the Apollo program was an important exercise both for NASA and for the discipline of software engineering. NASA acquired considerable experience in managing a large, real-time software project that would directly influence the development of the Shuttle on-board software. Software engineering as a specific branch of computer science emerged as a result of experiences with large-size military, civilian, and spaceborne systems. As one of those systems, the Apollo software effort helped provide

examples both of failure and success that could be incorporated into the methodology of software engineering.

In the Apollo program, as well as other space programs with multiple missions, system software and some subordinate computer programs are only written once, with some modifications to help integrate new software. However, each mission generates new operational requirements for software, necessitating a design that allows for change. Since 1968, when designers first used the term software engineering, consciousness of a software life cycle that includes an extended operational maintenance period has been an integral part of proper software development.

Even during the early 1960s, the cycle of requirements definition, design, coding, testing, and maintenance was followed, if not fully appreciated, by software developers. A Bellcomm report prepared for the Apollo program and dated November 30, 1964 could serve as an excellent introduction to the concept today[71]. The important difference from present practice was the report's recommendation that modules of code be limited to 200 to 300 lines, about five times larger than current suggestions. The main point of the report (and the thrust of software engineering) was that software can be treated the same way as hardware, and the same engineering principles can apply. However, NASA was more used to hardware development than to large-scale software and, thus, initially failed adequately to control the software development. MIT, which concentrated on the overall guidance system, similarly treated software as a secondary occupation[72]. This was so even though MIT manager A.L. Hopkins had written early in the program that "upon its execution rests the efficiency and flexibility of the Apollo Guidance and Navigation System"[73]. Combined with NASA's inexperience, MIT's non-engineering approach to software caused serious development problems that were overcome only with great effort and expense. In the end NASA and MIT produced quality software, primarily because of the small-group nature of development at MIT and the overall dedication shown by nearly everyone associated with the Apollo program[74].

Managing the Apollo Software Development Cycle

One purpose of defining the stages in the software development cycle and of providing documentation at each step is to help control the production of software. Programmers have been known to inadvertently modify a design while trying to overcome a particular coding difficulty, thus making it impossible to fulfill the specification. Eliminating communication problems and preventing variations from the designed solution are among the goals of software engineering. In the Apollo

program, with an outside organization developing the software, NASA had to provide for quality control of the product. One method was a set of standing committees; the other was the acceptance cycle.

Three boards contributed directly to the control of the Apollo software and hardware development. The Apollo Spacecraft Configuration Control Board monitored and evaluated changes requested in the design and construction of the spacecraft itself, including the guidance and control system, of which the computer was a part. The Procedures Change Control Board, chaired by Chief Astronaut Donald K. Slayton, inspected items that would affect the design of the user interfaces. Most important was the Software Configuration Control Board, established in 1967 in response to continuing problems and chaired for a long period by Christopher Kraft. It controlled the modifications made to the on-board software[75]. All changes in the existing specification had to be routed through this board for resolution. NASA's Stan Mann commented that MIT "could not change a single bit without permission"[76].

NASA also developed a specific set of review points that paralleled the software development cycle. The Critical Design Review (CDR) resulted in acceptance of specifications and requirements for a given mission and placed them under configuration control. It followed the preparation of the requirements definition, guidance equation development, and engineering simulations of the equations. Next came a First Article Configuration Inspection (FACI). Following the coding and testing of programs and the production of a validation plan, it marked the completion of the development stage and placed the software code under configuration control. After testing was completed, the Customer Acceptance Readiness Review (CARR) certified that the validation process resulted in correct software. After the CARR, the code would be released for core rope manufacture. Finally the Flight Readiness Review (FRR) was the last step in clearing the software for flight[77]. The acceptance process was mandatory for each mission, providing for consistent evaluation of the software and ensuring reliability. The unique characteristic of ICs of the Apollo software appeared at each stage of the software life cycle.

Requirements Definition

Defining requirements is the single most difficult part of the software development cycle. The specification is the customer's statement of what the software product is to do. Improperly prepared or poorly defined requirements mean that the resulting software will likely be incomplete and unusable. Depending on the type of project, the customer may have little or a lot to do with the preparation of the specification. In most cases, a team from the software developers

works with the customer.

MIT worked closely with NASA in preparing the Guidance and Navigation System Operations Plan (GSOP), which served as the requirements document for each mission. NASA's Mission Planning and Analysis Division at the Manned Spacecraft Center provided detailed guidance requirements right down to the equation level[78]. Often these requirements were in the form of flow charts to show detailed logic[79]. The division fashioned these requirements into a controlled document that contained specific mission requirements, preliminary mission profile, preliminary reference trajectory, and operational requirements for spacecraft guidance and navigation. NASA planned to review the GSOP at launch minus 18 months, 16 months, 14 months and then to baseline or "freeze" it at 13.5 months before launch. The actual programs were to be finished at launch minus 10.5 months and tested until 8 months ahead, when they were released to the manufacturer, with tapes also kept at MIT and sent to Houston, North American (CM manufacturer), and Grumman (LEM manufacturer) for use in simulations. At launch minus 4 months the core ropes were to be completed and used throughout the mission[80].

In software engineering practice today, the specification document is followed by a design document, from which the coding is done. Theoretically, the two together would enable any competent programmer to code the program. The GSOPs contained characteristics of both a specification and design document. But, as one of the designers of the Apollo and Shuttle software has said, "I don't think I could give you the requirements for Apollo and have you build the flight software"[81]. In fact, the plans varied both in what they included and in the level of detail requirements. This variety gave MIT considerable latitude when actually developing the flight software, thus reducing the chance that it would be easily verified and validated.

Coding: Contents of the Apollo Software

By 1963, designers determined that the Apollo computer software would have a long list of capabilities, including acting as backup to the Saturn booster, controlling aborts, targeting, all navigation and flight control tasks, attitude determination and control, digital autopilot tasks, and eventually all maneuvers involving velocity changes[82]. Programs for these tasks had to fit in the memories of two small computers, one in the CM and one in the LEM. Designers developed the programs using a Honeywell 1800 computer and later an IBM 360, but never with the actual flight hardware. The development computers generated binary object code and a listing[83]. The tape containing the object code

would be tested and eventually released for core rope manufacture. The listing served as documentation of the code[84].

Operating System Architecture

The AGC was a priority-interrupt system capable of handling several jobs at one time. This type of system is quite different from a "round-robin executive." In the latter programs have a fixed amount of time in which to run before being suspended while the computer moves on to the remaining pending jobs, thus giving each job the same amount of attention. A priority-interrupt system is always executing the one job with the highest priority; it then moves on to others of equal or lower priority in its queue.

The Apollo control programs included two related to job scheduling: the Executive and the Waitlist. The Executive could handle up to seven jobs at once while the Waitlist had a limit of nine short tasks[85]. Waitlist tasks had execution times of 4 milliseconds or less. If a task ran longer than that, it would be promoted by the Waitlist to "job" status and moved to the Executive's queue[86]. The Executive checked every 20 milliseconds for jobs or tasks with higher priorities than the current ones[87]. It also managed the DSKY displays[88]. If the Executive checked the priority list and found no other jobs waiting, it executed a program called DUMMY JOB continuously until another job came into the queue[89].

The Executive had other duties as part of controlling jobs. One solution to the tight memory in the AGC was the concept of time-sharing the erasable memory[90]. No job had permanent claim to any registers in the erasable store. When a job was being executed, the Executive would assign it a "coreset" of 12 erasable memory locations. Also, when interpretive jobs were being ran (the Interpreter is explained below), an additional 43 cells were allocated for vector accumulation (VAC). The final lunar landing programs had eight coresets in the LEM computer and just seven in the CM. Both had five VACs[91]. Moreover, memory locations were given multiple assignments where it was assured that the owning processes would never execute at the same time. This approach caused innumerable problems in testing as software evolved and memory conflicts were created due to the changes.

Programming the AGC

One can program a computer on several levels. Machine code, the actual binary language of the computer itself, is one method of specifying instructions. However, it is tedious to write and prone to error. Assembly language, which uses mnemonics for instructions (e.g., ADD in place of a 3-bit operation code) and, depending on its sophistication, handles addressing, is at a higher level. Most programmers in the early 1960s were quite familiar with assembly languages, but such programs suffered from the need to put too much responsibility in the hands of the programmer. For Apollo, MIT developed a special higher order language that translated programs into a series of subroutine linkages, which were interpreted at execution time. This was slower than a comparable assembly language program, but the language required less storage to do the same job[92]. The average instruction required two machine cycles—about 24 milliseconds—to execute[93].

The interpreter got a starting location in memory, retrieved the data in that location, and interpreted the data as though it were an instruction[94]. Instead of having only the 11 instructions available in assembler, up to 128 pseudoinstructions were defined[95]. The larger number of instructions in the interpreter meant that equations did not have to be broken down excessively[96]. This increased the speed and accuracy of the coding.

The MIT staff gave the resulting computer programs a variety of imaginative names. Many, such as SUNDISK, SUNBURST, and SUNDIAL, related to the sun because Apollo was the god of the sun in the classical period. But the two major lunar flight programs were called COLOSSUS and LUMINARY. The former was chosen because it began with "C" like the CM, and the latter because it began with "L" like the LEM[97]. Correspondence between NASA and MIT often shortened these program names and appended numbers. For example, SOLRUM55 was the 55th revision of SOLARIUM for the AS501 and 502 missions. BURST116 was the 116th revision of SUNBURST[98]. Although these programs had many similarities, COLOSSUS and LUMINARY were the only ones capable of navigating a flight to the moon. On August 9, 1968, planners decided to put the first released version of COLOSSUS on *Apollo 8*, which made the first circumlunar flight possible on that mission[99].

Handling Restarts

One of the most significant differences between batch-type computer

systems and real-time systems is the fact that in the latter, an abnormal termination of a program is not acceptable. If a ground-based, non-real-time computer system suffers a software failure ("goes down") due to overloads or mismanagement of resources, it can usually be brought up again without serious damage to the users. However, a failure in a real-time system such as that in an aircraft may result in loss of life. Such systems are backed up in many ways, but considerable emphasis is still placed on making them failure proof from the start. Obviously, the AGC had to be able to recover from software failures. A worst-case example would be a failure of the computer during an engine burn. The system had to have a method of staying "up" at all times.

The solution was to provide for restarts in case of software failures. Such restarts could be caused by a number of conditions: voltage failures, clock failure, a "rupt lock" in which the system got stuck in interrupt mode, or a signal from the NIGHT WATCHMAN program, which checked to see if the NEWJOB register had not been tested by the EXECUTIVE, indicating that the operating system was hung up in some way[100].

An Apollo restart transferred control to a specified address, where a program would begin that consulted phase tables to see which jobs to schedule first. These jobs would then be directed to pick up from the last restart point. The restart point addresses were kept in a restart table. Programmers had to ensure that the restart table entries and phase table entries were kept up to date by the software as it executed[101]. The restart program also cleared all output channels, such as control jet commands, warning lights, and engine on and off commands, so that nothing dangerous would take place outside of computer control[102].

A software failure causing restarts occurred during the Apollo 11 lunar landing. The software was designed to give counter increment requests priority over instructions[103]. This meant that if some item of hardware needed to increment the count in a memory register, its request to do so would cause the operating system to interrupt current jobs, process the request, and then pick up the suspended routines. It had been projected that if 85,000 increments arrived in a second, the effect would be to completely stop all other work in the system[104]. Even a smaller number of requests would slow the software down to the point at which a restart might occur. During the descent of Apollo 11 to the moon, the rendezvous radar made so many increment requests that about 15% of the computer systems resources were tied up in responding[105]. The time spent handling the interrupts meant that the interrupted jobs did not have enough computer time to complete before they were scheduled to begin again. This situation caused restarts to occur, three of which happened in a 40-second period while program P64 of LUMINARY ran during descent[106]. The restarts caused a series of warnings to be displayed both in the

spacecraft and in Mission Control. Steven G. Bales and John R. Garman, monitoring the computer from Mission Control, recognized the origin of the problem. After consultation, Bales, reporting to the Flight Director, called the system GO for landing[107]. They were right, and the restart software successfully handled the situation. The solution to this particular problem was to correct a switch position on the rendezvous radar which, through an arcane series of circuitry, had caused the analog-to-digital conversion circuitry to race up and down[108]. This incident proved the need for and effectiveness of built-in software recovery for unknown or unanticipated error conditions in flight software—a philosophy that has appeared deeply embedded in all NASA manned spaceflight software since then.

Verification and Validation

There could be no true certification of the Apollo software because it was impossible to simulate the actual conditions under which the software was to operate, such as zero-G. The need for reliability motivated an extensive testing program consisting of simulations that could be accomplished before flight. Three simulation systems were available for verification purposes: all-digital, hybrid, and system test labs. All-digital simulations were performed on the Honeywell 1800s and IBM 360s used for software development. Their execution rate was 10% of real time[109]. Technicians did hybrid simulations in a lab that contained an actual AGC with a core rope simulator (as core rope would not be manufactured until after verification of the program) and an actual DSKY. Additionally, an attached Beckman analog computer and various interfaces simulated spacecraft responses to computer commands[110]. Further ad hoc verification took place in the mission trainers located in Houston and at Cape Canaveral, which would run the released programs in their interpretive simulators.

The simulations followed individual unit tests and integrated tests of portions of the software. At first, MIT left these tests to the programmers to be done on an informal basis. It was very difficult at first to get the Instrumentation Laboratory to supply test plans to NASA[111]. The need for formal validation rose with the size of the software. Programs of 2,000 instructions took between 50 and 100 test runs to be fully debugged, and full-size mission loads took from 1,000 to 1,200 runs[112].

NASA exerted some pressure on MIT to be more consistent in testing, and it eventually adopted a four-level test structure based largely on the verification of the Gemini Mission Control Center developed by IBM in 1964[113]. This is important because formal release of the

program for rope manufacture was dependent on the digital simulations only. Raytheon performed the hybrid and system tests after they had the release tape in hand[114]. At that time, MIT would have released an AGC Program Verification Document to NASA. Aside from help from IBM, NASA also had TRW participate in developing test plans. Having an outside group do some work on verification is a sound software engineering principle, as it is less likely to have a vested interest in seeing the software quickly succeed, and it helps prevent generic errors.

Apollo Software Development Problems

Real-time flight software development on this scale was a new experience for both NASA and the MIT Instrumentation Laboratory. Memory limitations affected the software so that some features and functions had to be abandoned, whereas tricky programming techniques saved others. Quality of the initial code was sometimes poor, so verification took longer and was more expensive. Despite valiant validation efforts, software bugs remained in released programs, forcing adjustments by users. Several times, NASA administrators put pressure on MIT, to reduce software complexity because there were real doubts about MIT's ability to deliver reliable software on time. Apparently, few had anticipated that software would become a pacing item for Apollo, nor did they properly anticipate solutions to the problems.

By early 1966, program requirements even exceeded the Block II computer's memory. A May software status memo stated that not only would the programs for the AS504 mission (earth orbit with a LEM) exceed the memory capacity by 11,800 words but that the delivery date for the simpler AS207/208 programs would be too late for the scheduled launch[115]. Lack of memory and the need for faster throughput resulted in complicating and delaying the program development effort[116]. One of MIT's top managers explained

> If you are limited in program capacity ... you have to fix. You have to get ingenious, and as soon as you start to get ingenious you get intermeshing programs, programs that depend upon others and utilize other parts of those, and many things are going on simultaneously. So it gets difficult to assign out little task groups to program part of the computer; you have to do it with a very technical team that understands all the interactions on all these things[117].

The development of obscure code caused problems both in understanding the programs and validating them, and this, in turn, caused delays. MIT's considerable geographic distance from Houston caused additional problems. Thus, NASA's contract managers had to commute

often. Howard W. "Bill" Tindall, newly assigned from the Gemini Project as NASA's "watchdog" for MIT software, spent 2 or 3 days a week in Boston starting in early 1966[118].

Tindall was well known at the Manned Spacecraft Center due to his legendary "Tindallgrams"—blunt memos regarding software development for Apollo. One of the first to recognize the importance of software to mission schedules, he wrote on May 31, 1966 that "the computer programs for the Apollo spacecraft will soon become the most pacing item for the Apollo flights[119]. MIT was about to make the standard emergency move when software was in danger of being late: to throw more bodies into the project, a tactic that often backfires. As many as 50 people were to be added to the programming staff, and the amount of interaction between programmers and, thus, the potential for miscommunication increased along with the time necessary to train newcomers. MIT tried to protect the tenure of its permanent staff by using contractors who could be easily released. The hardware effort peaked at 600 workers in June of 1965 and fell off rapidly after that, while software workers steadily increased to 400 by August of 1968. With the completion of the basic version of COLOSSUS and LUMINARY, the number of programmers quickly decreased[120]. This method, although in the long-term interests of the laboratory, caused considerable waste of resources in communication and training.

Tindall's memo also detailed many of NASA's efforts to improve MIT's handling of the software development. Tindall had taken Lynwood Dunseith, then head of the computer systems in Mission Control, and Richard Hanrahan of IBM to MIT to brief the Instrumentation Laboratory on the Program Development Plan used for management of software development in the Real-Time Computing Center associated with Mission Control. The objective was to give MIT some suggestions on measuring progress and detecting problem areas early. One NASA manager pointed out that the Instrumentation Laboratory was protective of the image of MIT, and one way to control MIT was to threaten its self-esteem[121]. The need to call on IBM for advice was apparently a form of negative motivation. A couple of weeks later, Tindall reported that Edward Copps of MIT was leading the development of a Program Development Plan based on one done by IBM[122]. However, by July he was complaining that MIT was implementing it too slowly[123]. In fact, some aspects of configuration control such as discrepancy reporting (when the software does not match the specification) took over a year for MIT to implement[124].

NASA had to be very careful in approving cuts in the program requirements to achieve some memory savings. Some features were obviously "frosting," and could easily be eliminated; for example, the effects of the oblate nature of the earth, formerly figured into lunar orbit rendezvous but actually minimal enough to be ignored[125]. Also

cut were some attitude maneuver computations. They therefore left Reaction Control System (RCS) burns to the "feel" of the pilot, which meant slightly greater fuel expenditure[126]. Overall, the cuts resulted in software that saved money and accelerated development but could not minimize fuel expenditures nor provide the close guidance tolerance that was within the capability of the computer, given more memory[127].

Flight AS-204: A Breaking Point

Despite efforts by both MIT and NASA, by the summer of 1966, flight schedules and problems in development put both organizations in a dangerous position regarding the software. A study of the problems encountered with the software for flight AS-204, which was to be the first manned Apollo mission, best demonstrates the urgency. On June 13, Tindall reported that the AS-204 program undergoing integrated tests had bugs in *every* module. Some *had not been unit tested* prior to being integrated[128]. This was a serious breach of software engineering practice. If individual modules are unit tested and proven bug-free, then bugs found in integrated tests are most likely located in the interfaces or calling modules. If unit testing has not been done then bugs could be *anywhere* in the program load, and it is very difficult to identify the location properly. This vastly increases the time and, thus, the cost of debugging. It causes a much greater slip in schedule than time spent on unit tests. Even worse, Tindall said that the test results would not be formally documented to NASA but that they would be on file if needed.

The AS-204 software schedule problems affected other things. All the crew-requested changes in the programs were rejected because including them would cause even further delays[129]. The AS-501 program and others began to slip because the AS-204 fixes were saturating the Honeywell 1800s used in program development[130]. MIT also added another nine programmers to the team, all from AC Electronic, thus increasing communication and training problems.

The eventual result was that the flight software for the mission was of dubious quality. Tindall predicted such would be the case as early as June 1966, saying that we have every expectation that the flight program we finally must accept will be of less than desirable quality[131]. In other words, it would contain bugs, bugs that would not actually threaten the mission directly but that would have to be worked around either by the crew or by ground control. They found one such bug less than a month before the scheduled February 21, 1967, launch date. Ground computers and the Apollo guidance computer calculated

the time for the de-orbit burn that preceded re-entry. Simulations performed during January 1967 and reported on the 23rd indicated that there was a discrepancy between the two calculations of as much as 138 seconds! Since the core rope was already installed in the spacecraft, the only possible fix (besides a delay in the launch time) would be to have the crew ignore the Apollo computer solution. The ground would transmit the Real-Time Computing Center solution, after which an astronaut would have to key the numbers into the Apollo computer[132]. This situation, and other discrepancies, led one NASA engineer to later remark that we were about to fly with flight software that was really suspect[133].

AS-204 did not fly, so that software load was never fully tried. On January 27, 1967, during a simulation with the crew in the spacecraft on the pad, a fire destroyed the CM, killed the crew, and delayed the Apollo program for months. The changes in managing software development put into effect by NASA and MIT during 1966 had not had enough time to take effect before the fire. In the ensuing period, with manned launches on indefinite delay, MIT was under the direction of the NASA team led by Tindall and was able to catch up on its work and take steps to make the software more reliable. NASA and MIT split the effort among three programs: CM earth orbit, CM lunar orbit, and lunar module lunar landing (LM earth orbit was dropped)[134]. By October 17, 1967, the SUNDISK earth orbit program was complete, verified, and ready for core rope manufacture, a year before the first manned flight[135]. The time gained by the delay caused by the fire allowed for significant improvements in the Apollo software. Tindall observed at the time, “It is becoming evident that we are entering a new epoch regarding development of spacecraft computer programs.” No longer would programs be declared complete in order to meet schedules, requiring the users to work around errors. Instead quality would be the primary consideration[136].

The Guidance Software Task Force

Despite postfire improvements, Apollo software had more hurdles to clear. NASA was aware of continuing concern about Apollo’s computer programs. Associate Administrator for Manned Spaceflight George E. Mueller formed a Guidance Software Task Force on December 18, 1967 to study ways of improving development and verification.* The group met 14 times at various locations before its final report in September 1968[137].

*Members of the Task Force included Richard H. Battin, MIT; Leon R. Bush, Aerospace Corp.; Donald R. Hagner, Bellcomm; Dick Hanrahan, IBM; James S. Martin, NASA–Langley; John P. Mayer, NASA-MSC; Clarence Pitman, TRW; and Ludie G. Richard, NASA–Marshall. Mueller was the chairman.

Even while the Task Force was investigating, Mueller took other steps to challenge MIT. A Software Review Board re-examined the software requirements for the lunar mission in early February 1968. The board judged the programs to be too sophisticated and complex, and Mueller requested that they aim for a 50% reduction in the programs, with increased propellant consumption allowed as a tradeoff[138]. An aide reported that Mueller was convinced that MIT "might not provide a reliable, checked-out program on schedule" for the lunar landing mission[139].

The recommended 50% scrub did not occur, and the final report of the Task Force was very sympathetic to the problems involved in developing flight software. It recommended standardization of symbols, constants, and variable names used at both Houston and Huntsville to make communication and coding easier[140]. The Task Force acknowledged that requirements would always be dynamic and that development schedules would always be accelerated, but rather than using this for an excuse for poor quality, the group recommended that software not be slighted in future manned programs. Adequate resources and personnel were to be assigned early to this "vital and underestimated area"[141]. This realization would have great effect on managing later software development for the Space Transportation System.

Mueller remained concerned about software even after the Task Force dissolved. On March 6, 1969, he wrote a letter to Robert Gilruth, NASA deputy administrator, complaining that software changes were being made too haphazardly and should receive more attention, equal to that given to hardware change requests. Gilruth replied five days later, disagreeing, noting that the Configuration Control Board and other committees formed an interlocking system adequate for change control[142].

Lessons of the Apollo Software Development Process

Overcoming the problems of the Apollo software, NASA did successfully land a man on the moon using programs certifiably adequate for the purpose. No one doubted the quality of the software eventually produced by MIT nor the dedication and ability of the programmers and managers at the Instrumentation Lab. It was the process used in software development that caused great concern, and NASA helped to improve it[143]. The lessons of this endeavor were the same learned by almost every other large system development team of the 1960s: *(a)* documentation is crucial, *(b)* verification must proceed through several levels, *(c)* requirements must be clearly defined and carefully managed, *(d)* good development plans should be created and executed, and *(e)*

more programmers do not mean faster development. Fortunately, no software disasters occurred as a result of the rush to the moon, which is more a tribute to the ability of the individuals doing the work than to the quality of the tools they used.

USING THE AGC

The Apollo computer system made great demands on the crew. It took about 10,500 keystrokes to complete a lunar mission; not much in the life of an airline reservations clerk but still indicative of how computer centered the crew had to be[144]. During the period in which the software was criticized for its complexity, designers attempted to reduce the number of keystrokes required to execute various programs. When possible, they also eliminated built-in halts as data were displayed for astronaut approval. However, the "fourth crew member" never abandoned center stage[145].

Apollo's crew employed its computer through the use of the DSKYs. In the CM one was on the main control panel opposite the commander's couch. The other was at the navigator's station in the lower equipment bay, where the computer itself was located. Block I had a different DSKY at the navigator's station than on the main panel[146], but they were identical in the Block II series. DSKY and computer activity could be monitored from the ground as the computer transmitted data words to drive real-time displays in Mission Control[147].

The crew could communicate with the computer through keys, displays, and warning lights on the DSKY. Additionally, the uplink telemetry could provide input to the machine, and so could the preflight checkout equipment[148]. The computer, in turn, could communicate with the crew by flashing the PROGram, VERB, and NOUN displays[149]. The DSKY displays included 10 warning lights, a computer activity light, a PROGram display, VERB and NOUN displays, three five-digit numeric displays with signs, and 19 keys including VERB, NOUN, CLEAR, KEY RELEASE, PROCEED, RESET, ENTER, PLUS, MINUS, and the digits 0–9. See Boxes 2-2 and 2-3 for functions and use.

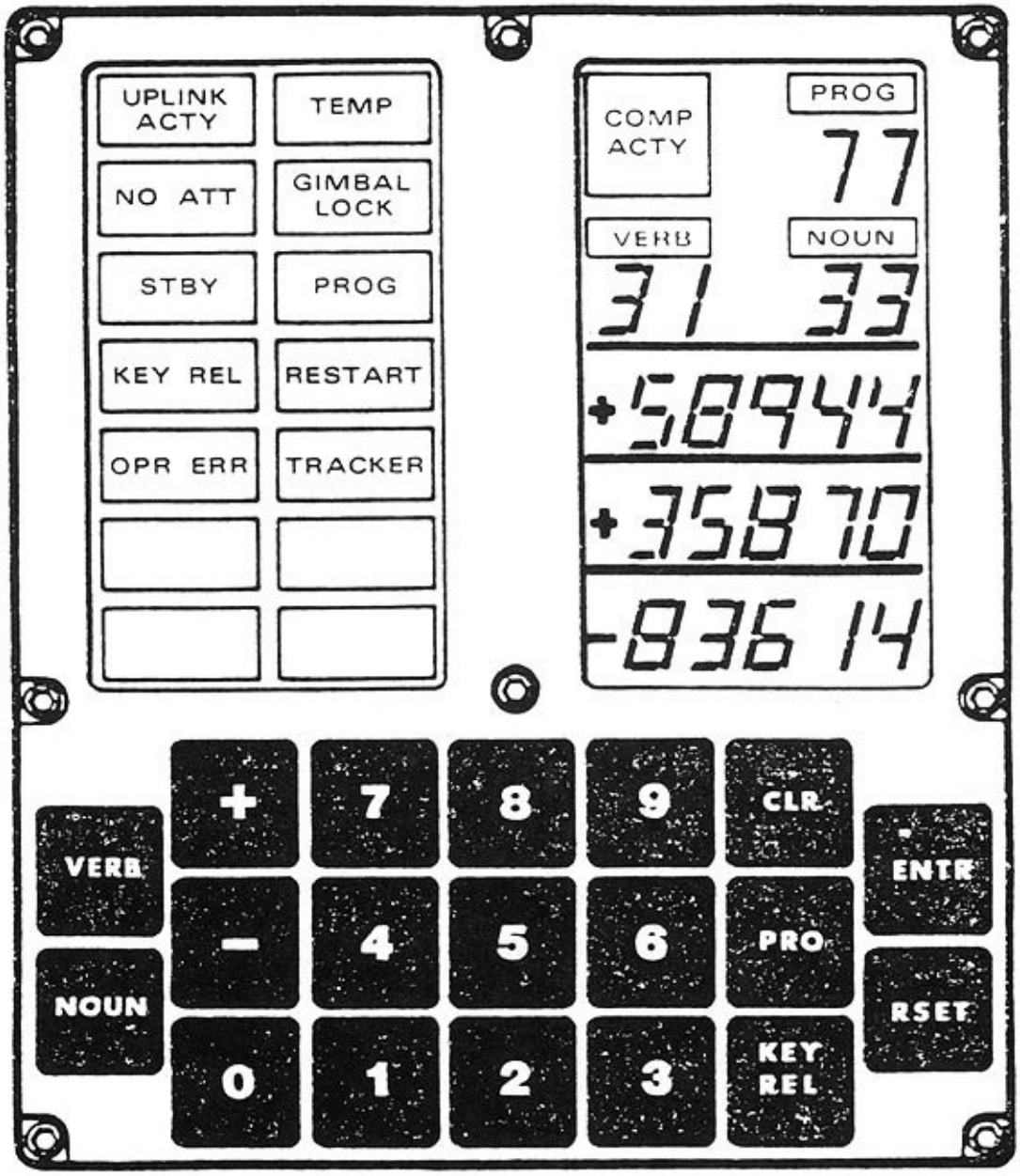

Figure 2–2. The Display and Keyboard (DSKY) of an Apollo spacecraft. (prepared by the Wichita State University Media Services)

Box 2-2: Apollo Display and Keyboard Lights

Ten DSKY warning lights had the following functions:

- COMP ACTY: This lit up when the computer was turning a program.
- UPLINK ACTY: Lit when data was being received from the ground.
- TEMP: Lit when the temperature of the stable platform was out of tolerance.
- NO ATT: Lit when the inertial subsystem could not provide attitude reference.
- GIMBAL LOCK: Lit when the middle gimbal angle was greater than 70 degrees.
- STBY: Lit when the computer system was on standby.
- PROG: Lit when the computer was waiting for additional information to be entered by the crew to complete the program.
- KEY REL: Lit when the computer needed control of the DSKY to complete a program. Sometimes display information could be "buried" under other routines or by a priority interrupt. The crew could press the KEY REL key to release the keyboard to the requesting program[150]. When the KEY REL light went on, that signaled the crew to press the key.
- RESTART: Lit when the computer was in the restart program. This was the light that kept coming on during the *Apollo 11* landing.
- OPR ERR: Lit when the computer detected an error on the keyboard.
- TRACKER: Lit when one of the optical coupling units failed.

The LEM DSKY had three additional lights: NO DAP, ALT, and VEL, which were related to failures of the digital autopilot and to warn of altitude and velocity readings outside of the predetermined limits.

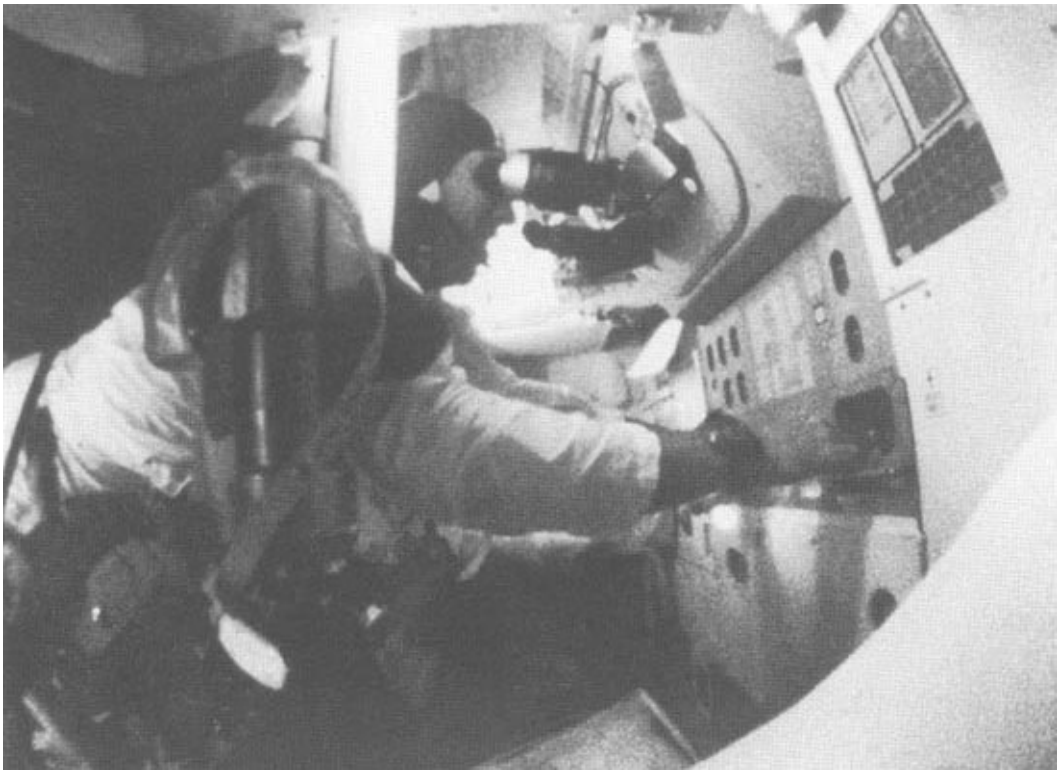

Figure 2–3. Another DSKY was located at the navigator's station in the command module. Astronaut James A. Lovell takes a star sighting during the Apollo 8 mission. (NASA photo S-69-35099)

Box 2-3: Apollo Display and Keyboard Displays

Seven displays were available on the DSKY:

- PROG: This was a two-digit display indicating what numbered program the computer was currently executing.
- VERB: A two-digit display of the verb number being entered (the verb–noun system is discussed below).
- NOUN: A two-digit display of the noun number being entered.
- Three five-digit numeric displays, which showed numbers in either decimal or octal (base eight). When a sign was shown with the number, the number was decimal; otherwise, it was octal[151].

Figure 2–4. The interior of an Apollo Command Module, showing the location of the DSKY on the main control panel at the left. Apollo 15 crewmen shown include Alfred M. Worden (center) and David R. Scott (left). James B. Irwin is mostly obscured to the right. (NASA photo SS-71-29952)

Using the Keys and the Verb-Noun System

Astronauts used keys to enter information and select programs and actions. Key inputs caused automatic interrupts in the software[152]. The astronauts would activate a program and then interact with it by requesting and entering information; a typical software load consisted of about 40 programs and 30 simultaneous routines[153]. Changing programs and making other requests involved using the verb–noun system. Those familiar with current computer keyboards will notice

the lack of alphabet keys on the DSKY. Whereas most computer commands are entered by typing in the text of the command, the Apollo computer command list specified verb and noun pairs. There were 100 two-digit numbers available for each, and most were used on any given flight. Examples of verb–noun pairs are "display velocity" and "load angle." Verb 37, for example, was "Change Prog," which enabled the crew to set up a new program for execution.

If, for example, the crew wanted to execute the rendezvous targeting program, an astronaut would first press the VERB key followed by the digits 3 and 7, and then the ENTER key. That sequence informed the computer of a request for a program change. The astronaut would then press 3, 1, and ENTER to tell the computer to execute program P31. Within the program the crew could request maneuver angles (verb 50, noun 18), monitor the changes while a maneuver was in progress (verb 06, noun 18), or request the velocity change required for the next maneuver (verb 06, noun 84), among other functions. The CSM G&C Checklist, a set of "cue cards" on three rings changed for each mission by the Crew Procedures Division in Houston, described all these sequences in detail. The document contained reference data, such as a star list, verb list, noun list, alarm codes, error handling and recovery, and the checklists for each program carried in the computer.

Despite the 100 verb–noun pairs, 70-odd programs and routines, and a very limited user interface that alternated decimal and octal and blinked for attention, the consensus is that the Apollo computer was easy to use. As with other aspects of flying space missions, hours in simulators made operating the computer second nature. NASA engineer John R. Garman commented that "it's like playing the piano—you don't have to see your fingers to know where they are"[154]. Familiarity with the computer, remarked astronaut Eugene Cernan, meant that pressing a wrong key simply and immediately "felt" wrong[155]. Others also confirmed that using the machine eventually became relatively natural[156]. Apollo astronauts were also willing to adapt to design foibles that would frustrate others. There were concerns that a crewman initiating a maneuver from the navigator's station would not be able to return to his couch before the burn started. In response, Virgil Grissom was accommodating: "Well, we'll just lie down on the floor"[157]. Astronauts also tolerated non-life-threatening software errors not cleared up before flight as merely something else to endure[158]. They did, however, complain about the annoying number of keystrokes required during a rendezvous, so designers modified the software to make a "minkey" (minimum keystroke) option available, in which the computer could perform some functions without constant crew approval[159]. This change contributed to an even more compact, straightforward system.

THE ABORT GUIDANCE SYSTEM

The computer in the Abort Guidance System (AGS) is probably the most obscure computing machine in the manned spaceflight program to date. The 330-page "Apollo Spacecraft News Reference" prepared for the first lunar landing mission does not contain a single reference to it, compared with several pages of description of the Primary Guidance, Navigation, and Control System (PGNCS) computer and its interfaces. The invisibility of the AGS is a tribute to PGNCS, since the AGS was never needed to abort a landing. It was, however, an interesting and pioneering system in its own right.

The AGS owed its existence to NASA's abort policy; an abort is ordered if one additional system failure would potentially cause loss of crew[160]. Hence, the failure of *either* the PGNCS *or* the AGS would have resulted in an abort. The AGS operated in an open loop, parallel to the PGNCS in the LEM, and gave the crew an independent source of position, velocity, attitude, and steering information[161]. It could verify navigation data during the periods when the LEM was behind the moon and blacked out from ground control. The Apollo program first exercised this capability during Apollo 9 and Apollo 10 leading up to the first landing mission[162].

The AGS was a pioneer in that it was the first "strapped-down" guidance system. The system used sensors fixed to the LEM to determine motion rather than a stable platform as in conventional inertial guidance systems[163]. The entire system occupied only 3 cubic feet and consisted of three major components: *(a)* an Abort Electronic Assembly (AEA), which was the computer, *(b)* an Abort Sensor Assembly (ASA), which was the inertial sensor, and *(c)* a Data Entry and Display Assembly (DEDA), which was the DSKY for the AGS.

AEA and DEDA: The Computer Hardware

As with the PGNCS computer, the AGS computer went through an evolutionary period in which designers clarified and settled the requirements. The first design for the system did not include a true computer at all but rather a "programmer," a fairly straightforward sequencer of about 2,000 words fixed memory, which did not have navigation functions. Its job was simply to abort the LEM to a "clear" lunar orbit (one that would be higher than any mountain ranges) at which point the crew would wait for rescue from the CM, with its more sophisticated navigation and maneuvering system[164]. The requirements changed in the fall of 1964. To provide more autonomy and safety, the AGS had to provide rendezvous capability without outside sources

of information[165]. TRW, the contractor, then decided to include a computer of about 4,000 words memory. The company considered an existing Univector accumulation machine but, instead, chose a custom designed computer[166].

The computer built for the AGS was the MARCO 4418 (for *Ma*n *R*ated *Co*mputer). It was an 18-bit machine, with 17 magnitude bits and a sign bit. It used 5-bit op codes and 13-bit addresses. Numbers were stored in the two's complement form, fixed point, same as in the primary computer. Twenty-seven instructions were available, and the execution time varied from 10 to 70 microseconds, depending on the instruction being performed[167]. The computer was 5 by 8 by 23.75 inches, weighed 32.7 pounds, and required 90 watts[168]. The memory was bit serial access, which made it slower than the PGNCS computer, and it was divided into 2K of fixed cores and 2K of erasable cores[169]. The actual cores used in the fixed and erasable portions were of the same construction, unlike those in the PGNCS computer. Therefore, the ratio of fixed memory to erasable in the MARCO 4418 was variable[170]. TRW was obviously thinking in terms of adaptability to later applications.

The DEDA was much smaller and less versatile than the DSKY. It was 5.5 by 6 by 5.19 inches and was located on the right side of the LEM control panel in front of the pilot, about waist height[171]. Sixteen pushbutton keys were available: CLEAR, READOUT, ENTER, HOLD, PLUS, MINUS, and the digits 0–9. It had a single, nine-window readout display. Three windows showed the address (in octal), one window the sign, and five, digits[172]. This was similar to the readout in the Gemini spacecraft for its computer.

Software for the AGS

Since hardware in the AGS evolved as in PGNCS, software also had to be "scrubbed" (reduced in size) in the AGS. Mirroring the memory problems of PGNCS, by 1966, 2 full years before the first active mission using the LEM, only *20 words* remained of the 4,000 in the AGS memory[173]. Careful memory management, became the focus of TRW and NASA. Tindall recalled that the changes all had to be made in the erasable portion, as the fixed portion was programmed early and remained set to save money. However, changing the erasable memory turned out to be very expensive and a real headache, the developers fighting to free up storage literally one location at a time[174]. Also, some software decisions had to be altered in light of possible disastrous effects. The restart program for the PGNCS has been described. In it, a restart clears all engine burns. The first versions of the AGS software also caused engine shutdown and an attitude hold to go into

effect when a restart occurred. This would be potentially dangerous if a restart began with the LEM close to the lunar surface. The solution was to give the crew responsibility to manually fire the engines during a restart if necessary[175].

Software development for the AGS followed a tightly controlled schedule:

1. 12.5 months before launch: NASA delivers the preliminary reference trajectory and mission requirements to TRW.
2. 11 months: Program specification and AGS performance analysis is complete.
3. 10.5 months: NASA conducts the Critical Design Review (CDR).
4. 8 months: The final mission reference trajectory is delivered.
5. 7 months: The equation test results, verification test plan, and preliminary program goes to NASA for approval.
6. 6.5 months: The First Article Configuration Inspection (FACI) conducted.
7. 5 months: The verified program and documentation is delivered to NASA.
8. 4.5 months: NASA conducts the Customer Acceptance Readiness Review (CARR).
9. 3 months: The operational flight trajectory is delivered by NASA to the contractor.
10. 2 months: The final Flight Readiness Review (FRR) is held.
11. 1.5 months: The tape containing the final program is delivered[176].

One method of software verification was quite unique. To simulate motion and thus provide more realistic inputs to the computer, planners used a walk-in van containing the hardware and software. Technicians drove the van around Houston with the programs running inside it[177].

Use of the AGS

The AGS was never used for an abort, but it did contribute to the final rendezvous and docking with the CM on the Apollo 11 mission, probably to avoid the problems encountered with the rendezvous radar during landing[178]. It did monitor PGNCS performance during all missions in which it flew. The only criticism of its performance was from astronaut John Young, who remarked that "one mistake in a rendezvous, and the whole thing quit"[179]. Apparently, restarts occurred as part of the recovery from some operator errors. The AGS was actually like a parachute—absolutely necessary, but presumably never needed.

LESSONS

What did NASA learn from its experiences with the Apollo computer system? At the management level, NASA learned to assign experienced personnel to a project early, rather than using the start of a project for training inexperienced personnel; many NASA managers of software and hardware were learning on the job while in key positions. Also, more participation by management in the early phases of software design is necessary so that costs can be more effectively estimated and controlled.

From the standpoint of development, NASA learned that a more thorough, early effort at total systems engineering must be made so that specifications can be adequately set. NASA contractors in the Apollo program faced changing specifications long after final requirements should have been fixed. This was expensive and caused such problems as Raytheon's retooling, memory shortages, and design insufficiencies.

The realization that software is more difficult to develop than hardware is one of the most important lessons of the Apollo program. So the choice of memory should be software driven, and designers should develop software needed for manned spaceflight near the Manned Spacecraft Center. The arrangement with MIT reduced overall quality and efficiency due to lack of communication. Also, more modularization of the software was needed[180].

The AGC system served well on the earth-orbital missions, the six lunar landing missions, the three Skylab missions, and the *Apollo-Soyuz* test project. Even though plans existed to expand the computer to 16K of erasable memory and 65K of fixed memory, including making direct memory addressing possible for the erasable portion, no expansion occurred[181]. The Apollo computer did fly on missions other than Apollo. An F-8 research aircraft used a lunar module computer as part of a "fly-by-wire" system, in which control surfaces moved

by servos at the direction of electronic signals instead of traditional cables and hydraulics. In that way, the Apollo system made a direct research contribution to the Shuttle, which is completely a fly-by-wire craft. The most important legacy of the AGC, however, was in the way NASA applied the lessons it was beginning to learn in developing ground software to the management of flight software.