Analysis of the Apollo and CEV Guidance and Control Systems and the Impact of Risk Management

Elwin C. Ong
Katherine H. Allen
Ilana Davidi
Robbie C. Allen

May 9th 2005
16.895J/STS.471J/ESD.30J - Engineering Apollo

Introduction

In 1961, President Kennedy challenged the nation to put a man on the moon by the end of the decade. This gave the nascent space program a gigantic boost in political support and finding, but created a host of engineering problems to be very rapidly solved. One of the largest challenges was how to safely and precisely navigate a several ton spacecraft from the surface of the earth to the moon and back. The guidance, navigation, and control (GNC) systems which would perform this task represented one of the biggest risk factors for the entire Apollo project: enough thrust will launch a craft into space, but doing so in an exact manner is a much more challenging engineering feat.

At the time of Kennedy's speech, no one had built a system similar in scope to Apollo. With the relative newness of computers in the early 1960s, reliability of hardware and software were chief concerns at NASA. Every step of the way, engineers had to consider whether to use older technology that was more proven but had limitations or newer technology that could allow for more robust functionality but with decreased reliability. They also had to consider what to do in case of system failures, including abort modes and real-time diagnostics and recovery. The end result was a system which successfully operated without a major failure throughout the seventeen Apollo missions.

Looking forward to the next generation spacecraft referred to as the Crew Exploratory Vehicle (CEV), many lessons can be learned from the success of Apollo. In the 33 years since the last Apollo mission, space vehicles have become significantly more complex. The field of risk management also has evolved considerably. It is unlikely that Apollo would pass today's safety requirements. Apollo got away with a lot of single point failures because of the tight timeline Kennedy had put on NASA. Apollo nonetheless succeeded because much attention was paid to detail, and the engineers were dedicated to their work.

In addition, the political and social context in which the Apollo project operated made funding and popular support insignificant constraints on the program. The environment in which CEV is being built is considerably different. Due to the recent Columbia disaster, NASA is being scrutinized closely and there is no Soviet threat to motivate development of a space system. In order to appear safe, CEV may end up being so redundant and fault tolerant that it will be too complex to manage effectively, and hence, there will be a failure because nobody will understand the system well enough to predict how it will work.

Apollo Computing Systems

The MIT Instrumentation Lab under Charles Stark (Doc) Draper received the contract to provide the primary navigation, guidance, and control for Apollo in August of 1961. At the time, NASA was still debating how to land on the moon:

Lunar Orbit Rendezvous, Earth Orbit Rendezvous, and Direct Ascent were all still valid possibilities. Regardless of that decision, however, there was no doubt that a powerful digital computer would be the center piece of the complex system required to guide the spacecraft to the moon, land it safely, and return the astronauts back to Earth.

The MIT Instrumentation Laboratory was the pioneer of inertial guidance and navigation, so it was a natural choice to design the Apollo GN&C system. Doc Draper had first applied the use of gyros on the Mark 14 gun sight during WWII. The effectiveness of the system led to more advanced applications including self-contained inertial systems on aircraft and missiles. By the mid 1950's, the Instrumentation Lab was working on a number of applications of inertial guidance including the Air Force's Thor missile, the Navy's Polaris missile, and a robotic Mars Probe [HALL40].

A computation system, either analog or digital, was required to apply the guidance and control equations. There were already plenty of computers around by the 1950s including the ENIAC, Whirlwind, and IBM 604 [TCP], but these computers were much too big and heavy for use in aerospace applications. To apply the guidance and control equations for the Polaris missile, MIT developed a set of relatively simple equations that were implemented using digital differential analyzers. The digital differential analyzer designed by MIT was nothing more than some memory registers to store numbers and adders that produced the result of the incremental addition between two numbers. Although simple by computational standards, the work on the Polaris digital system provided the necessary base of technology needed for the Apollo Guidance Computer (AGC). Wire interconnections, packaging techniques, flight test experience, and the procurement of reliable semiconductor products were all required for the successful delivery of the AGC [HALL44].

In the late 1950's, the Instrumentation Laboratory was granted a contract to study a robotic mission to Mars. They designed a probe that would fly to Mars, snap a single photo, and return it safely to Earth [BAT], using a digital computer, the Mod 1B. The computer would have been responsible for navigation and control of the probe through its mission. The resulting computer used core-transistor logic and core memories. It was a general-purpose computer, meaning it could be programmed, unlike the Polaris system. While the Polaris computer could only calculate one set of equations, the Mod 1B computer could be programmed to perform any number of calculations. Although the Mars probe was canceled before it was built, the computer designed to photograph Mars evolved into the computer which landed the Lunar Module.

Apollo Computers

Two identical computers were used on Apollo–one in the Command Module and another in the Lunar Module. The hardware and software on each were exactly

the same, as required by NASA. This requirement made the design of the computer more difficult as the computer had to interface with different and unique equipment for the CM and LM. In addition, since different contractors built the CM and LM, any changes to the computer meant that all three groups, plus the NASA supervisory group, had to agree to the changes. However, having the same type of computer on both spacecraft simplified production and testing procedures.

Lunar Module Landing System Architecture

Hall, E., *Journey to the Moon*, AIAA, 1996

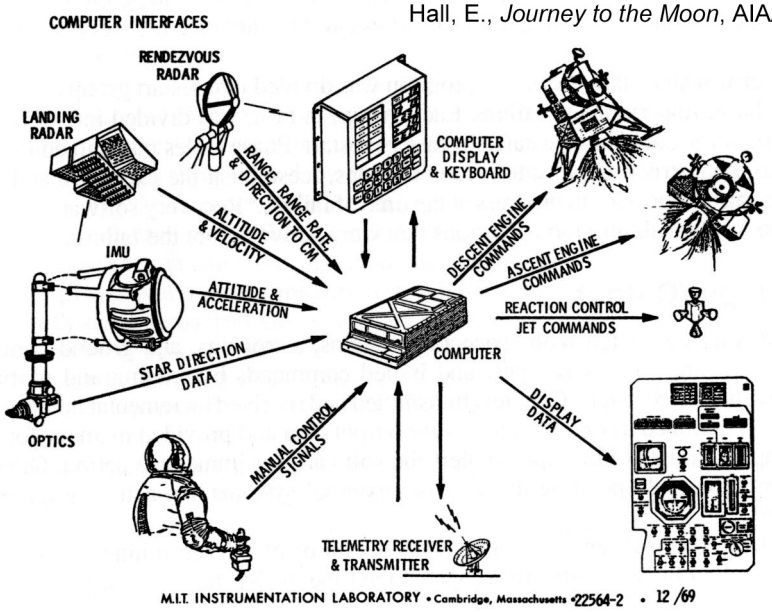

Fig. 94 LM system interfaces (Courtesy MIT Instrumentation Laboratory Report, Graphic Number R700-1-54).

The Lunar Module (LM) landing guidance system consisted of several major components. Among them were the Primary Guidance, Navigation and Control System (PGNCS), the Abort Guidance System (AGS), the landing radar, the LM descent engine, RCS jets, and crew interfaces. The PGNCS included the fixme (IMU) for inertial guidance, and the digital computer. Within the computer was a digital autopilot program (DAP) and manual control software. The AGS was responsible for safely aborting the descent and returning the LM ascent stage back to lunar orbit if the PGNCS were to fail. (It was never used in flight.) The landing radar (LR) on board the LM provided ranging and velocity information relative to the lunar surface as the LM descended.

The LM descent engine was a single throttleable rocket engine. It had a maximum thrust of approximately 10,000 lbs and could operate at full thrust or throttled between 10-60% of maximum power. In addition, it rested on gimbals which provided 6 degrees of rotation. The gimbal was controlled by the DAP for low rate attitude adjustments [BEN]. For high-rate attitude changes, the LM used

a fully redundant RCS system featuring 8 thrusters in each set. Each thruster was capable of providing 100 lbs of thrust.

In order to take input from the crew during landing, the capsule had several manual control interfaces. Among these were the DSKY, used by the astronauts to call various programs stored on the computer, a sidearm control stick for manual control of the attitude control jets, and a grid on the commander's forward window called the **fixme: expand**LPD. The window was marked on the inner and outer panes to form an aiming device or eye position. The grid was used by the computer to steer the LM to the desired landing site. By using a hand controller, the commander could change the desired landing spot by lining up a different target through the grid on his window.

LM Landing Strategy

After separation from the Command and Service Module (CSM), the LM began descent to the moon by first performing a descent orbit insertion maneuver. This maneuver was performed with the descent engine at full thrust to lower the LM's altitude from 60 nautical miles to 50,000 feet. Once the spacecraft reached 50,000 ft, the powered descent phase was initiated. The LM powered descent strategy was divided into 3 separate phases. The first phase, called the braking phase, was designed primarily for efficient propellant usage while reducing orbit velocity. At 60 seconds to touchdown and approximately 7000 feet, the approach phase was initiated. This phase was designed to allow the astronauts to monitor their descent and pick an alternate landing spot if required. The final phase, or landing phase, was designed to allow manual control of the descent as the LM approached the lunar surface [BEN].

PGNCS Architecture

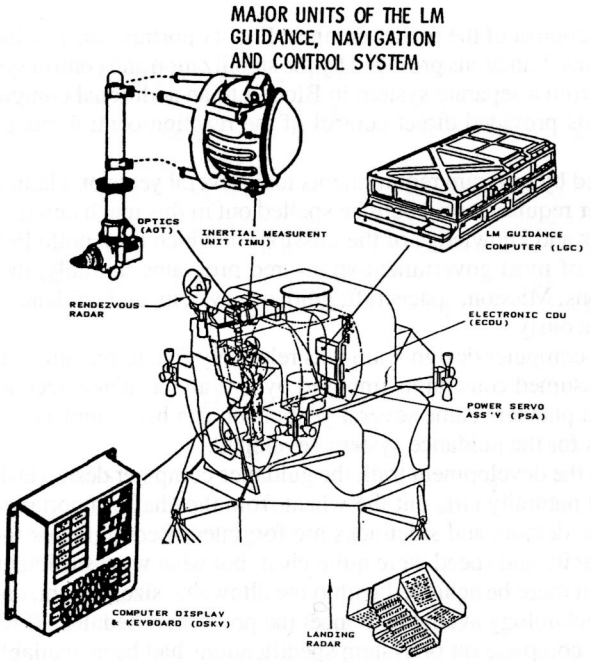

Fig. 39 LM spacecraft GN&C system (Courtesy Charles Stark Draper Laboratory Archives, Graphic Number 59334). Hall, E., *Journey to the Moon*, AIAA, 1996

The Primary Guidance, Navigation, and Control System (PGNCS) architecture on board the LM included two major components: the Apollo Guidance Computer (AGC) and the Inertial Measurement Unit (IMU) (See Figure 39 HALL). The AGC was the center piece of the system. It was responsible for calculating the state vector (position and velocity) of the vehicle at all times, and interfaced with the crew and other systems on board. The IMU provided inertial measurements from gyros and accelerometers. These measurements were integrated to derive the vehicle's position and velocity.

AGC Architecture

The Mod 1B evolved into the Mod 3C, then later into the Block I computer, and finally into the Block II. The Block II computer was the heart of the PGNCS used on every LM. The Command Module (CM) used the same computer. The final Block II design consisted of an architecture with a 16 bit word length (14 data bits, 1 sign bit, and 1 parity bit), 36,864 words of fixed memory, 2,048 words of erasable memory, and a special input/output interface to the rest of the spacecraft. See Appendix A for more on the significance of word length and arithmetic precision with Apollo.

The completed Block II computer was packaged and environmentally sealed in a case measuring 24 by 12.5 by 6 inches. The computer weighed 70.1 lbs, and required 70 watts at 28 volts DC [TOM]. Work on the computer design was led

by Eldon Hall. Major contributions were made by Ramon Alonso, Albert Hopkins, Hal Lanning, and Hugh Blair-Smith among others.

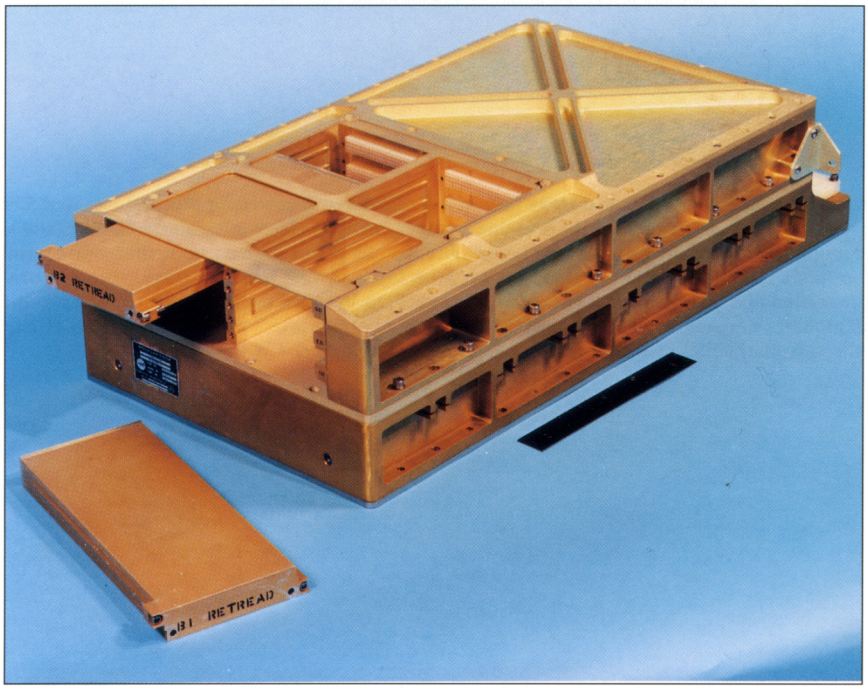

Plate 35 Block II AGC. Back-panel wiring, electrical interconnections between connector pins, was machine wire-wrapped and encapsulated in a light plastic foam as in the Block I computer. Tray covers provided an environmental seal. (Courtesy Charles Stark Draper Laboratory Archives, Photograph Number 40265-C.)

Hall, E., *Journey to the Moon*, AIAA, 1996

AGC Processor

The AGC processor was a trail-blazer in digital computing. It was the first to use integrated circuits (IC), which was a new and unproven technology at the time. Integrated circuits were first introduced in 1961. An IC is a thin chip consisting of at least two interconnected semiconductor devices, mainly transistors, as well as passive components like resistors [WIK,IC]. ICs permitted a drastic reduction in the size and number of logic units needed for a logic circuit design. (See figure 5) The first ICs were produced by Texas Instruments using Germanium junction transistors. Silicon-based transistors soon followed, with the first IC developed by the Fairchild Camera and Instrument Corporation [HALL18].

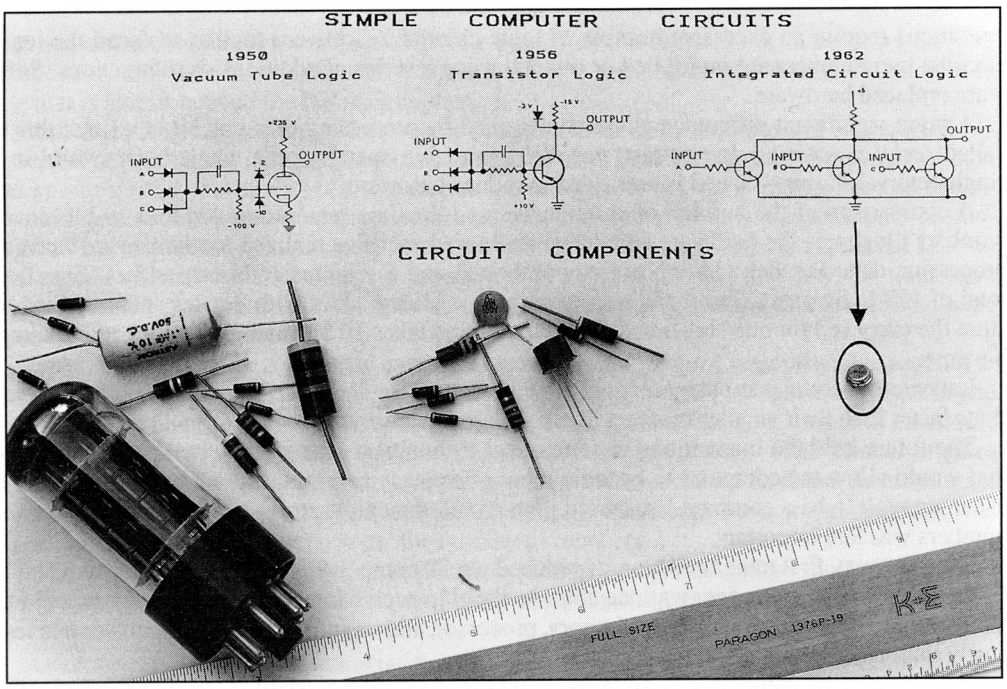

Fig. 5 Circuit component revolution.
Hall, E., *Journey to the Moon*, AIAA, 1996

In 1962, the Instrumentation Lab obtained permission from NASA to use the Fairchild's Micrologic IC on the AGC [HALL18]. The Fairchild Micrologic IC was a three input NOR gate. The output of the NOR gate was a 1 if all three inputs were zeros. Otherwise, the output was a zero. The AGC processor was created entirely from this one basic logic block.

There were many risks involved with using ICs on the AGC. The Instrumentation Lab and NASA evaluated the benefits and risks of using ICs thoroughly before making their decision. As Eldon Hall recalls, there was resistance both from NASA and people within the Lab who had invested much of their work in core-transistor logic. In the end, Hall was able to persuade NASA that the advantages of ICs outweighed the risks involved [HALL,108,109].

By 1963, Fairchild introduced the second generation Micrologic gate, which put 2 NOR gates on a single chip. In addition to doubling in gate capacity, the chip also operated at a faster speed, used less power, and had an improved packaging design known as a “flat-pack.” (See Figure 73 HALL) These new ICs were incorporated into the design of the Block II computer, producing a savings in weight and volume, and allowing more room for the expansion of the memory.

Even in 1962, the pace of IC development was progressing steadily. However, this was not always to the benefit of the Apollo program. Before the first Block II computer was produced, Fairchild had dropped production of the Mircologic line, electing instead to concentrate production on more advanced chips. Fortunately

for the Instrumentation Lab, Philco Corporation Microelectronics Division maintained production of the IC for the life of the program [HALL23].

The final Block II computer included approximately 5700 logic gates. They were packaged into 24 modules. Together, they formed the brain of the computer, providing instructions for addition, subtraction, multiplication, division, accessing memory, and incrementing registers, among others.

AGC Memories

The AGC had two types of memories. Erasable memory was used to store results of immediate calculations during program execution, while programs were stored in permanent read-only memory banks. The erasable memory was based on those used on the Gemini spacecraft, and was made from coincident-current ferrite cores [TOM]. Unlike modern erasable memories, which are usually made with transistors, the erasable memory in the AGC was based on magnetic principles rather than electrical. Ferrite core memories were first used on the Whirlwind computer at MIT in 1951 and were the standard technology used for erasable memories at the time the AGC was developed.

The ferrite cores were circular rings that, by virtue of its ferromagnetic properties, could store a bit of information (that is, a one or a zero) by changing the direction of the magnetic field. A wire carrying a current passing through the center of the ring changed the direction (clockwise vs. counter-clockwise) of the magnetic field, and hence, changed the information stored in the ferrite core. The primary advantage of this type of technology is that the memory retains its data even when power is removed [JON]. The main disadvantages of ferrite core memories are that they were relatively large and heavy and required more power than electronically-based memories.

The fixed memory for the AGC was based on the same principles as the erasable memory, except all the ferrite cores were permanently magnetized in one direction. The signal from a wire which passed through a given core would then be read as a one, while those that bypassed the core would be read as a zero. Information was stored and read from memory in the form of computer words by selecting the correct core and sensing which wires represent ones and zeros. Up to 64 wires could be passed through a single core [WIK,CR]. In this way, the software for the AGC was essentially stored in the form of wire ropes. Because of this, the fixed memory soon came to be referred as core-rope memory. MIT originally invented the core-rope technology for use on the Mars probe. Its chief advantage was that it stored a lot of information in a relatively small amount of space, but it was very difficult to manufacture [TOM]. The memory could not be easily changed after the ropes were manufactured. MIT contracted Raytheon to manufacture the units using a unique process based on a weaving machine programmed by punchcards. Due to the lead time required

for manufacturing and testing, the software had to be completed and delivered by MIT six weeks in advance of the NASA hardware deadlines [BAT].

Memory capacity was an issue throughout the design phases of the AGC. The initial memory design called for only 4000 words of fixed memory and 256 words of erasable. The final Block II design had 36,000 words of fixed memory and 2000 words of erasable. The underestimation of memory capacity was mainly due to difficulties in the software development [HOP]. As Hugh Blair-Smith recalls, MIT continually underestimated the task of developing software [HBS]. The engineers at MIT also had a predisposition to add more and more complex requirements to the software, as long as they seemed like apparently good ideas [HBS]. Eventually, hard work, some ingenious schemes to save memory capacity, and oversight from NASA helped stem the out-of-control memory growth.

DSKY Design

> "How do you take a pilot, put him in a spacecraft, and have him talk to a computer?" -Dave Scott, Apollo 15 commander

In the early 1960s, there were very few options for input and output devices which meant human interaction with computers was limited to highly-trained operators. "Computers were not considered user-friendly," [ELD] explained Eldon Hall, one of the Apollo GNC system designers. For example, one of the premier computers of the time, the IBM 7090, read and wrote data from fragile magnetic tapes, and took input from its operator on a desk-sized panel of buttons. It was one of the first computers using transistors, rather than vacuum tubes—a huge advantage in robustness, but a relatively unproven technology. The 7090 used to control the Mercury spacecraft also occupied an entire air-conditioned room at Goddard Spaceflight Center [FRO]. As a result, the Apollo GNC system designers faced a quandary: a room of buttons and switches would not fit inside the CM—a simpler interface would be necessary.

No one had experience putting human beings in space, much less having them control complicated machines. It was unclear what information the astronauts would find useful while flying, or how best to display that information. "Everybody had an opinion on the requirements. Astronauts preferred controls and displays similar to the meters, dials, and switches in military aircraft. Digital designers proposed keyboard, printer, tape reader, and numeric displays." [HALL71] Although the astronauts' opinions held a lot of weight, some of the simple analog displays they were accustomed to were simply infeasible in a craft run by a 1960s-era digital computer. "Astronauts and system engineers did not understand the complicated hardware and software required to operate meters and dials equivalent to those used in military airplanes." [HALL71] This made it difficult for designers to satisfy the astronauts' desire for aircraft-like controls while still meeting NASA's deadlines and other requirements.

Astronauts were not the only ones asking for miracles from the GNC software. Jim Nevins, an Instrumentation Lab engineer, says that ”back in the ’62 time period, the computer people came to me and proposed that they train the crew to use octal numbers.” [NEV] This would have simplified the computer’s job deciphering commands, but would have been very difficult on the astronauts, who already had a busy training schedule learning the various aspects of their job. Eldon Hall does not remember that suggestion, but recounted that “the digital designers expressed a great deal of interest in an oscilloscope type of display...a vacuum tube, a fragile device that might not survive the spacecraft environment. It was large, with complex electronics, and it required significant computing to format display data.” This, also, was rejected—the fragile vacuum tubes would have been unlikely to survive the G-forces of launch and re-entry.

Eventually, a simple, all-digital system was proposed which included a small digital readout with a seven-segment numeric display and a numeric keyboard for data entry. The simple device referred to as DSKY used a novel software concept: ”Numeric codes identified verbs (display, monitor, load, and proceed) or nouns (time, gimbal angle, error indication, and star id number). Computer software interpreted the codes and took action.” [HALL73] The pilots were happy with the new device. David Scott, Apollo 15 commander, commented that “it was so simple and straightforward that even pilots could learn to use it.” [HALL73] Many of the pilots, including Scott, actually helped to develop the verb-noun interface. “The MIT guys who developed the verb-noun were Ray Alonzo and [A.L.] Hopkins, but it was interactively developed working with the astronauts and the NASA people.” [NEV]

The display keyboard, or DSKY (Figure 1) is composed of three parts: the numeric display, the error lights, and the keypad. The display uses an 8-bit register to display up to 21 digits (two each for the program, verb, and noun selected, and three rows of five digits for data). Next to the display is a row of error and status lights, to indicate important conditions like gimbal lock (an engine problem where the gimballed thrusters lock into a certain configuration), and operator error, among others. Below the lights and the display panel is a 19-button keyboard. This keyboard features a now-standard 9-button numeric keypad as well as a “noun” button to indicate that the next number being entered is a noun, a “verb" button, a “prg” button, for program selection, a "clear" button, a key release, an “enter” button, and a "reset" button. The crew could enter sequences of programs, verbs, and nouns to specify a host of guidance and navigation tasks. A selection of programs, verbs, and nouns from Apollo 14’s GNC computer are provided in Appendix B.

Figure 1. A Close-up of the DSKY device as mounted in the Apollo 13 CSM, Odyssey.

The design of the DSKY changed very little between Block I and Block II—the commands changed, but the interface device stayed essentially the same.

Anthropometry, Displays, and Lighting

The field of anthropometry was relatively new in 1960. Some work had been done at Langley, quantitatively describing the handling qualities of aircraft (and leading to the development of the Cooper-Harper scale for rating handling qualities) but the majority of human factors issues were still addressed by trial and error. Jim Nevins, in a briefing in April 1966, summarized the Instrumentation Lab's areas of human factor activity with the following list:

1. Anthropometry and gross configuration associated with:
• display and control arrangement
• zero g tethering
• general lighting and caution annunciators

2. Visual and visual-motor subtasks for the
• optics (space sextant, scanning telescope, and alignment optical telescope)
• computer (display keyboard)
• data and data handling

3. Evaluation of relevant environmental constraints associated with
• pressure suit
• zero g
• high g
• interior illumination
• vibration, acoustic noise
• gaseous environment
• physiologic stress, fatigue

The human factors of each design were investigated primarily by using astronauts and volunteers at the MIT I/L and elsewhere to test the designs for LM hardware—both in “shirtsleeves” tests and full-up tests in pressure suits, to ensure that the relatively rigid suits with their glare and fog-prone bubble helmets would not interfere with the crew’s ability to perform necessary tasks. The I/L had a mockup of the CM and LM panels, which, in addition to the simulators at Cape Canaveral and Houston, allowed proposed hardware displays, switches, and buttons to be evaluated on the ground in a variety of levels of realism.

AGC Software

The AGC mission software was a large and complex real-time software project. (The experience gained by NASA during their oversight of the Apollo software development would directly influence the development of the Space Shuttle software later [TOM]). The architecture of the AGC software was a priority-interrupt system capable of handling several jobs at a time: the computer would always execute the job with the highest priority. All other jobs would be queued and executed once higher priority jobs were completed. The main advantage of a priority-interrupt system is that it is very flexible. Once an operating system was written, new programs could be added quite easily. On the other hand, the software was nondeterministic, which made testing much more difficult. The Apollo 11 alarm during lunar landing was just one of an infinite number of unpredictable sequences of jobs in the computer. To combat this, the software designers added protection software that would reset the computer when it detected a fault in the execution of a program. This fault protection software was vital in allowing Eagle to land instead of aborting the mission in the final minutes of the lunar landing.

Hal Lanning led the development of the AGC operating system. The tasks of the operating system were divided into two programs: 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 [TOM]. The Waitlist handled jobs that required a short amount of time to execute, on the order of 4 milliseconds or less, while the Executive handled most of the computer’s workload. Every 20 milliseconds, the Executive checked its queue for jobs with higher priorities.

Writing software for the AGC could be done using machine code, calling basic computer instructions at each step. Although code was often more efficient when written this way, the development process was tedious and more error prone. Often, software designers at MIT used an interpretive language that provided higher level instructions such as addition, subtraction, multiplication, and division. More advanced instructions included square roots and vector dot and cross products. When executed on the computer, each interpretive instruction was translated at run-time into the low-level instructions needed by the computer.

Digital Autopilot

Programs were organized and numbered by their phase in the mission. The programs related to the descent and landing of the LM were P63-67. P63 through P65 were software responsible for guiding the LM automatically through the powered descent and braking phases of the lunar descent. P66 and P67 were optional programs that were called by the astronauts at any time during the descent. They provided the astronauts with manual control of the LM attitude and altitude. In all phases of the descent, the digital autopilot was responsible for maintaining the spacecraft attitude through firing RCS jets and gimballing the LM descent engine [COC]. Even during manual control, all commands from the astronauts were first sent to the computer. It was a one of the first fly-by-wire system ever designed.

P63 Function

P63 was the first of a series of sequential programs used to guide the LM from lunar orbit down to the surface. The task of P63 was to calculate the time for the crew to initiate ignition of the descent engine for powered descent. This time was calculated based on the position of the LM relative to the planned landing site. Upon ignition of the engine, P63 used guidance logic to control the LM descent towards the approach phase. The braking phase was designed for efficient reduction of orbit velocity and used maximum thrust for most of the phase [BEN]. When the calculated time to target reached 60 seconds, at an approximate altitude of 7000 feet and 4.5 nautical miles from the landing site, P63 automatically transitioned to P64 to begin the approach phase.

P64 Function

P64 carried on the descent, adjusting the spacecraft attitude for crew visual monitoring of the approach to the lunar surface. Measurements from the landing radar became more important in this phase, as the spacecraft approached the lunar surface. Measurements from the radar were more accurate closer to the surface, which counter balanced the effects of drift in the IMU. P64 also allowed the commander to change the desired landing spot by using the hand controller and LPD.

P65 Function

At a calculated time to target of 10 seconds, P65 was called to perform the final landing phase of the descent. P65 nulled out velocity changes in all three axes to preselected values, allowing for automatic vertical descent onto the lunar surface if desired [BEN]. Probes, which extended 5.6 feet below the landing pads signaled contact with the surface and activated a light switch on board the spacecraft, signaling the crew to shut off the descent engine.

Manual Control Software

Control System Design [1]

The LM PGNCS uses the AGC to execute the control laws for a digital control system by timesharing the computing resources. (This is how modern digital computers execute most computations, but it was relatively novel in the 1960s.)

Inside the LM, there are two hand controllers—one for the Commander and one for the LM Pilot—which can each issue attitude commands in six directions. Whenever the hand controller's deflection exceeds the "soft stop" at 11 degrees , it closes the manual override switch and allows the astronauts to directly command the thrusters. In this manner, they succeed in enabling human participation—the manual control mode is always available to the pilot and commander, regardless of the guidance mode otherwise selected. If no deflections are input to the hand controller, the Digital AutoPilot (DAP) is executed 10 times per second to control the LM based on the state vector information in the PGNCS. The DAP uses a filter similar to a Kalman filter to estimate bias acceleration, rate, and attitude. However, the gains used are not the Kalman gains---they are nonlinearly-extrapolated from past data stored in the PGNCS, as well as data on engine and thruste**rs**. The nonlinearities in this control allow the system to exclude small oscillations due to structural bending and analog-to-digital conversion errors.

Within the realm of manual control, there are two sub-modes which respond to motion of the side-arm controller stick. The first, "Minimum Impulse Mode", provides a single 14-ms thruster pulse each time the controller is deflected. This is particularly useful in alignment of the inertial measurement unit (IMU). The second mode is PGNCS Rate Command/Attitude Hold Mode, which allows the astronauts to command attitude rates of change (including a rate of zero, that is, attitude hold).

The system used in Apollo 9, internally called SUNDANCE, used a nonlinear combination of two attitude rates (Manual Control Rates, or MCRs): 20 deg/s for "Normal" maneuvering, and 4 deg/s for "Fine" control. In addition, SUNDANCE system has a large deadband: a section of motion where control inputs create no system response. This deadband helps to prevent limit cycling, a condition where the system begins to oscillate due to controller phase lag. Although it increases system stability, a deadband tends to decrease pilot satisfaction with the system's handling qualities, since a larger controller input is required to achieve the minimum allowed thrust pulse. This is particularly a problem since it tends to encourage larger pulses than the minimum possible, which wastes reaction control fuel.

---

[1] Summarized based on Stengel, Robert F. "Manual Attitude Control of the Lunar Module", June 1969

In the LUMINARY system, the GN&C designers discovered that they could achieve a well-controlled system, with almost ideal theoretical handling qualities (i.e. those which would occur in a system with very small or no deadband) without inducing limit cycles. To do this, the designers reduced the Manual Control Rates of the normal control system from 20 deg/s to 14 deg/s, and had pilots operate the system and rate its handling qualities on the Cooper scale. To the surprise of the investigators, the Cooper ratings improved as MCR decreased. They continued to decrease MCR, to 8 deg/s , and continued to see the Cooper ratings of pilot satisfaction with handling quality increase. However, in order to allow a maximum control rate of 20 deg/s (the rate considered necessary for emergency maneuvers) the l/L engineers had to implement a linear-quadratic scaling system for MCR. In addition, to simplify the task of controlling the LM, the PNGCS system adds a “pseudo-auto” mode. This mode maintains attitude automatically in two axes (using minimum impulses of the RCS), so that the astronaut only has to close a single control loop to control the spacecraft in the remaining axis. This type of control system epitomizes the design philosophy of the PNGCS—using digital autopilot control where it simplifies the astronaut’s task, and using manual control where human interaction is beneficial and/or simplifying.

Specification	**SUNDANCE value**	**LUMINARY value**
Minimum Firing Time	14 ms	14 ms
DAP sampling delay	130-250 ms	130-250 ms
Fine MRC	4 deg/s	4 deg/s
Normal MRC	20 deg/s	$w_c = \frac{MCR}{40} sgn(\delta) \left[\lvert\delta\rvert - 2 + \frac{(\lvert\delta\rvert - 2)^2}{2} \right]$

Table 1: Apollo PNGSC Systems: SUNDANCE (Apollo 9) and LUMINARY

In-Flight Maintenance

``In 1964, if you could get 100 hours MTBF on a piece of electronics, that was a good piece of electronics" [NEV] Unfortunately, the Apollo GN&C system needed to have hundreds of electronic parts, all of which had to operate simultaneously for not only the two weeks (~300 hours) of the mission, but for the entire mission preparation period, which might be several months, with tens of simulated missions.

There were a host of suggestions for how the GN&C computer might be made more robust against electronics failures. At the bidder's conference in the spring of 1962, one bidder on the computer's industrial support contract made a suggestion that summed up the difficulty. ``The bidder proposed that the spacecraft carry a soldering iron. Repair would involve removing and replacing individual components. Although the proposal seemed extreme, a provision for

in-flight repair was the only way to achieve the necessary level of confidence" (HALL 92).

A slightly more realistic plan to deal with reliability issues was to train the astronauts to replace components in-flight. This would still require the development of reliable connectors which could be mounted on printed circuit boards, but would only require the astronauts to replace whole modules. MIT I/L folks were still skeptical. "We thought [in flight-maintenance] was nonsense" recalled Jim Nevins, who was at the I/L at the time, ``but we had to evaluate it. We laid out a program for the crew based on the training of an Air Force Navigator: normal basic training, plus maintenance training, plus basic operational flight, and there was a tremendous cost to do all this---it took over three years. The crew training people were horrified. This went down like thunder, and we got invaded---all the six of the astronauts came down to the Instrumentation Lab. The end result was that you can't go to the moon and do all the things you want to do, so the requirement for in-flight maintenance was removed."[NEV]

The idea of replaceable components did not entirely disappear, however, until the engineers began to discover the problems with moisture in space. “In Gordon Cooper's Mercury flight, some important electronic gear had malfunctioned because moisture condensed on its uninsulated terminals. The solution for Apollo had been to coat all electronic connections with RTV, which performed admirably as an insulator.” [AHO] This potting (replaced with a non-flammable material after the Apollo 1 fire) prevented moisture from getting into the electronics, but made in-flight repair essentially impossible.

Abort Guidance System

The Abort Guidance System (AGS) was unique to the LM. Built by TRW, it served as a backup to the PGNCS. In case the PGNCS failed during landing, the AGS would take over the mission and perform the required engine and RCS maneuvers to put the LM into an appropriate orbit for rendezvous. (A backup computer was not needed in the CM as the ground controllers provided the guidance and navigational information for the crew. In operation, the PGNCS essentially was the backup for the ground controllers.) For the LM, however, especially during the final phases of lunar landing, the three second communication delay meant that guidance and control from the ground would have been useless. The AGS was designed and built solely to fill the backup role for this single phase of the mission, but because the PGNCS worked so well, it was never used in flight.

AGS Hardware

Similar to the PGNCS, the AGS had three major components: the Abort Electronic Assembly, which was the computer, the Abort Sensor Assembly, a

strap down inertial sensor, and a Data Entry and Display Assembly, where commands were entered by astronauts [TOM]. The AGS computer architecture had 18-bits per word with 27 machine instructions. It had 2000 words of fixed memory and 2000 words of erasable memory. The completed package was 5 by 8 by 24 inches, weighed 33 pounds, and required 90 watts [TOM].

AGS Software

As with the PGNCS, memory capacity was the major issue in the development of the AGS software. Unlike the PGNCS however, the operating system was based on a round-robin type architecture. Every job was assigned a time slot during each round, and the computer would process jobs sequentially, repeating the process every round. The AGS software provided the crew with the same state vector information as the PGNCS, derived independently from its own inertial units. It had software to guide the LM through an abort and safe rendezvous with the CM. Like the PGNCS, the software development effort for the AGS faced similar issues including memory capacity and changing requirements.

Control: Manual, Autonomous, or Automatic?

According to Eldon Hall, “Autonomous spacecraft operation was a goal established during [MIT’s initial Apollo] study. Autonomy implied that the spacecraft could perform all mission functions without ground communication, and it justified an onboard guidance, navigation, and control system with a digital computer. The quest for autonomy resulted, at least in part, from international politics in the 1950s and 1960s, specifically the cold war between the Soviet Union and the United States. NASA assumed that autonomy would prevent Soviet Interference with US space missions”. [HALL59] The threat of Soviet interference in the US manned spaceflight programs made autonomous operation a requirement—but the Instrumentation Lab engineers were not satisfied with autonomy.

“An auxiliary goal of guidance system engineers was a completely automatic system, a goal that was more difficult to justify. It arose as a technical challenge and justified by the requirement for a safe return to Earth if the astronauts became disabled”. [HALL59] The guidance system engineers were understandably optimistic about the possibility of automatic guidance—their experience designing the guidance for the US Navy’s Polaris ballistic missile and the recently-cancelled Mars project, both fully-automatic systems, indicated that automatic lunar missions were reasonable—but feasibility was not the only constraint on system design.

One of the other constraints was the preferences of the system operators. The astronauts were relatively happy with an autonomous system—no pilot wants his craft flown from the ground—-but were quite unhappy with the idea of an entirely automatic system. They wanted the system autonomous, but with as much

capacity for manual control as possible. Jim Nevins observed that “the astronauts had this 'fly with my scarf around my neck' kind of mentality. The first crew were real stick and rudder people— not engineers at all”. [NEV] This difference in mentality—between the operators of the system and the designers who really know the details and “funny little things”[2] about the system—caused significant disagreement during the control system design and even later, into the first flights.

Jim Nevins tells a story about Astronaut Walter Shirra that illustrates the mindset of the astronauts:

“My first exposure to astronauts was in the fall of 1959. A student of mine, Dr. Robert (Cliff) Duncan, was a classmate of Walter Shirra at the Naval Academy. After a NASA meeting at Langley, Cliff invited me to lunch with Wally.” Although their conversation ranged over many topics, “the memorable one was Wally’s comments related to astronaut crew training and the design of the spacecraft control system for the Mercury and Gemini spacecrafts.”

“Wally wanted rudder pedals in the Mercury," explained Jim. The Mercury, Gemini, and Apollo systems all had a side-arm controller, which was not only stable in a control sense, but utilized a dead-band controller (a control system where motion of the control stick less than a certain threshold angle is ignored) to reduce the effects of accidental stick motion. The astronaut was still in control, but traditionalists considered this type of control risky—in order to make the system stable if the man let go, it was also made less reactive to the controls.

Jim continued, “A couple years later, when we became involved with Apollo, the NASA training people told me that the only way they could satisfy Wally was to promise him we’d have rudder pedals in Gemini, where they had more room. Of course this was never done—by that time Wally had become more comfortable with the side-arm controller.” [NEV]

To prove that the sidearm controller was superior, they tested the astronauts with a traditional system and the sidearm system “under 9, 10, 15 Gs. I told people ’You know, they’re not going to have control [of the stick and rudder]. They’re going to be flopping over.’ Even with that kind of data they still didn’t want [the sidearm control device].” [MIN]

“The meeting was a ’stage-setter’ for me in that it defined the relationship between ‘us’ (the designers) and the ’crew’ (the real-time operators). It meant that we could only achieve the program’s goals by involving the crew in all facets and depths of the design process.”[NEV]

[2] At the Instrumentation Lab, people used this as a technical term, abbreviated as “FLTs.”

Eventually, a set of guidelines were established for the Instrumentation Lab engineers working on Apollo, which were called General Apollo Design Ground Rules: [JNE]

- The system should be capable of completing the mission with no aid from the ground; i.e. self-contained
- The system will effectively employ human participation whenever it can simplify or improve the operation over that obtained by automatic sequences of the required functions
- The system shall provide adequate pilot displays and methods for pilot guidance system control
- The system shall be designed such that one crew member can perform all functions required to accomplish a safe return to earth from any point in the mission.

These guidelines allowed the engineers to include the appropriate levels of autonomy, automation, and manual control in the Apollo GNC system.

Risk Management and Apollo

Risk management may not have been a term used in the 1960s, yet the care that was applied while developing software for the AGC showed exceptional risk management. Many of the risk management tasks during Apollo were imposed on the team by the technology available at that time.

"When we would send something off to the computer, it took a day to get it back. So what that forced us into is I remember thinking 'if I only get this back once a day, I'm going to put more in to hedge my bets. If what I tried to do here doesn't work…maybe what I try here. I learned to do things in parallel a lot more. And what if this, what if that. So in a way, having a handicap gave us a benefit." [MHA]

A key design goal of the AGC was simplicity. Margaret Hamilton recalls how many of the applications in those days were designed by groups sitting in places like bars, using cocktail napkins where today we would use whiteboards in conference rooms. "Here, it was elegant, it was simple. But it did everything…no more no less (to quote Einstein)," as opposed to the more distributed, procedurally-influenced code of today in which "You end up with hodge podge, ad hoc." [MHA]

"While in traditional systems engineering, desired results are obtained through continuous system testing until errors are eliminated (curative), the [MIT I/L]Team was focused on not allowing errors to appear in the first place (preventative)." [CUR4] All onboard software went through six different levels of testing. Each level of testing would result in additional components being tested together [SAF].

Due to having the code “sewn” everything needed to be working when it was integrated. “There was not the carelessness at the last minute that sometimes occurs today. We went through everything before it went there.” On Apollo, the combination of a restriction of space and numerous peer reviews kept the code tight and efficient. This pain threshold for each bug was a sufficient deterrent to cause programmers to do their best to get it right the first time around.

Part of the peer management involved programmers eyeballing thousands of line of raw code. John Norton was the lead for this code review process, which was sometimes called “Nortonizing.” “He would take the listings and look for errors. He probably found more problems than anybody else did just by scanning and half of what went on there was simulation for testing.” [MHA] This includes a potentially dangerous bug where 22/7 was used as an estimation of pi. The guidance equations needed a much more precise approximation, so Norton had to scour the code for all locations where the imprecise fraction was used [SAF].

A big part of Apollo’s success was that the programmers learned from their errors. “We gradually evolved in not allowing people to do things that would allow those errors to happen.” [MHA] These lessons were documented in technical memos, many of which are still available today.

With all the testing and simulations MIT did on the software, it is surprising any bugs crept into the code at all. But it did happen---the non-deterministic nature of the code made it impossible to test all cases. Dan Lickly, programmer for much of the initial re-entry software thinks that “errors of rare occurrence—those are the ones that drive you crazy.” With these kinds of bugs, you can run simulations a thousand times and not generate an error.” [SAF] To combat this, the AGC had excellent error detection, and the computer could reboot itself if it encountered a potentially fatal problem. When it started up again, it would reconfigure itself and start processing from the last saved point. This built a failsafe into the system to account for any missed bugs.

Apollo also managed risk by maximizing the commonality of hardware and software components. All the system software–the procedures for reconfiguration, for restart, for displaying---were the same between the CM and LM. Variations were permitted only where the CM and LM had different mission requirements. For example, the CM did not have to land on the moon, so it did not have the capacity to do that. The conceptual stuff was the same.

In addition, there were some software variations because of the different programmers in charge of the CM and LM softwares. “The personalities felt very different about what they had to do: the command module was more traditional, the LM less traditional in its approach.” Commonality was encouraged, so wherever they could be, they were the same, but “the gurus in charge didn’t discuss…just did it their own way.”[MHA] This might be considered risky, since it

increases the amount of different software paradigms with which the crew must interact.

Thoughts on CEV Landing System Design

Whatever form the final landing system design will take, it will surely require a powerful computing system to implement the guidance and control systems. Space-based computing systems have evolved tremendously since the Apollo program, but there are still many challenges to overcome. These include fault tolerance, human-automation interfaces, advanced control law design, and software complexities.

CEV Computing Hardware

The current state-of-the-art in spacecraft computing systems is the Space Shuttle Primary Computer System. Although it has been in operation for over 20 years, the system still sets the standard for space-based real-time computing, fault tolerance, and software design. The Space Shuttle Primary Computer System uses a total of five general purpose computers, with four running the Primary Avionics Software System, and the fifth running an independent backup software [ONG]. The four primary computers run synchronously–each computer is constantly checking for faults in its own system as well as the other three computers. The added fault tolerance capability comes at a cost, as the algorithms for ensuring synchronous operation and fault checking is extremely complex. (The first Space Shuttle flight was postponed due to a fault in the synchronization algorithm, which was only discovered during launch.)

The CEV computer will likely resemble the Space Shuttle system more than that of Apollo. Partially, this is because the systems used in Apollo were at the leading edge of technology at the time, but computing technologies have advanced since 1970. The larger reason, however, is that these increases in technology allow for more margin and more redundancy to be built into the system than was possible in Apollo. Safety concerns and popular/political perceptions of safety concerns make it very difficult to remove redundancy from a system–even if that redundancy makes the system less safe overall.
The tradeoff between risk mitigation and increased complexities will have to be balanced effectively to maximize the reliability and complexity of the system as a whole. A synchronous triple modular redundant computing system should provide the necessary fault tolerance required, while maintaining a reasonable level of complexity. Similar systems are employed daily on safety-critical fly-by-wire commercial aircraft like the Boeing 777 [YEH] and Airbus A3XX family [BER].

CEV Mission Software

The CEV mission software would be one of the most complex and daunting software projects ever undertaken. Much insight can be gained by emulating successful programs such as the Space Shuttle software and fly-by-wire aircraft software. Emphasis should be given to simplicity and thorough evaluation and validation. Although tremendously successful, the Space Shuttle Software is prohibitively expensive and complex [MAD]. The CEV will be more reliable and easier to operate with a single software system, rather than two separate systems. The backup software has never been used on the Space Shuttle, and it can be argued that the cost and effort of producing the backup software could be better spent on validating the primary software. The requirements for two separate software systems would significantly add to the complexity of the system [KL].

Builders of the CEV guidance system should still code for reliability over redundancy. For example, in a redundant atmosphere, if two groups build off the same specification, and the specification is incorrect, both groups will produce problematic end results. In addition, Hamilton said, "There's a primary and a secondary. So if something goes wrong with the primary, it could go to a worse place when it goes to secondary. If you make a bad assumption in the spec, they're both going to still be bad"

CEV Automation

As in Apollo, the level of automation in the CEV will have significant political overtones. The final decision between a human pilot and a machine pilot will certainly be a political decision, not an engineering decision. However, since automated systems have become more reliable in the intervening 40 years since the Apollo project began, the CEV will likely have a sophisticated automated piloting and landing system. Although automated landing systems have been employed for many years in robotic missions, the CEV will be the first to employ such a system on a manned mission. To prevent a disastrous accident like the one experienced by the Mars Polar Lander [MPL], the automation software will require extensive and thorough review and testing. The Apollo software should serve as an excellent starting point for the proposed design. The sophisticated landing software used on the LM was in fact capable to landing the craft on its own, with the crew serving as system monitors [BEN]. New technologies such as the use of more powerful computers and advanced control law designs should be added when necessary, but the overall objective will be to maintain simplicity and avoid unnecessary complexities.

Human Factors and CEV

If the CEV does utilize a human pilot, improvements in manual control systems can also be applied to the basic Apollo design. New hardware provides the option to take factors like handling quality into the initial design, which will make

the manual control system easier to design. Advances in ergonomics and published ergonomic standards (like MIL-STD-1472D, the government standard for ergonomics) provide a compilation of the modern understanding of human-machine interaction requirements. Testing will still be required, as ergonomics will, of course, be only one facet of display and control system placement, but predefined requirements should assist designers in optimizing cabin layout.

Risk Management and CEV

Today, risk management can actually serve to increase risk rather than mitigate it. While we have much more knowledge of software today and tools available at our disposal, the designers of the AGC may have had an advantage. "Creative people were given the freedom to do it without any legacy distracting them or influencing them." [MHA]

Among the most important requirements for the CEV is that the software should be asynchronous and multi-threaded. In a synchronous environment, "If you touch it, everything falls out of place." This was one of the problems with the Shuttle software. Going forward, the CEV should have asynchronous processing so that events and objects are easy to add or reconfigure.

After the Shuttle disaster, NASA called for ways to improve the shuttle. Many submissions were made, and forty-four were selected for further research. "The resultant 44 proposals, internally known at NASA as 'The Famous 44' were made available to NASA management only 90 days after the [Columbia] disaster."[CUR5] Three of these were based on Apollo's guidance system. Eventually, the field was narrowed to 13, and then to one. The final one was written by Margaret Hamilton and her team, and was based on taking all of the technologies from Apollo and applying them directly (Appendix C) **[Ilana: Add this appendix]**.

One of the goals listed in the final paper was "to reuse systems and software with no errors to obtain the desired functionality and performance, thereby avoiding the errors of a newly developed system." [CUR4]

Many software development tasks which were done manually during Apollo can now be automated. Today, we can use their methods of concurrent and parallel coding efforts that the Apollo used to design the LM and CM at the same time. Reuse is assuredly more formalized, but by keeping the code simple without many bells and whistles, the sharing should be easy. Said Hamilton, "We would learn from what we did then and make it inherent…I'd have a human involved in going from specs to code and now we automatically generate the code so we don't have those human errors but we still follow the rules."

A further way to ensure that the software is easy to track and reconfigure is to develop with an open architecture. Spend time doing extensive design and analysis, "defining the system as a system," [MHA] and create it so it works with changeable languages or platforms. Any steps that can ensure a safe integration should be identified and standardized immediately.

Conclusion

<TBD>

Appendix A - Word Length and Arithmetic Precision

A digital computer stores numbers in binary form. To achieve arithmetic precision, there must be enough bits to store a number in a form sufficient for mathematical precision. To increase this precision, a number can be stored using 2 words, with a total of 28 data bits. A binary number stored with 28 bits is equivalent to around 8 decimal digits. To express the distance to the moon, 28 bits would be enough to express the number in 6 foot increments, which was more than enough for the task. [HHBS]

Appendix B – DSKY Commands

The DSKY accepted commands with three parts: a program (a section of code which corresponded to a generic section of the mission), a verb describing what action the computer was to take, and a noun describing what the verb acts on. The following commands were used in the Apollo Guidance Computer on Apollo 14, and correspond to the Luminary 1D program.

Number	Title
Service	
P00	LGC Idling
P06	PGNCS Power
P07	Systems Test (Non-flight)
Ascent	
P12	Powered Ascent Guidance
Coast	
P20	Rendezvous Navigation
P21	Ground Track Determination
P22	RR Lunar Surface Navigation
P25	Preferred Tracking Attitude
P27	LGC Update
Pre-thrusting	
P30	External delta-V
P32	Co-elliptic Sequence Initiation (CSI)
P33	Constant Delta Altitude (CDH)
P34	Transfer Phase Initiation (TPI)
P35	Transfer Phase Midcourse (TPM)
Thrust	

P40 DPS Thrusting
P41 RCS Thrusting
P42 APS Thrusting
P47 Thrust Monitor

Alignments

P51 IMU Orientation Determination
P52 IMU Realign
P57 Lunar Surface Alignment

Descent & Landing

P63 Landing Maneuvre Braking Phase
P64 Landing Maneuvre Approach Phase
P66 Rate of Descent Landing (ROD)
P68 Landing Confirmation

Aborts & Backups

P70 DPS Abort
P71 APS Abort
P72 CSM Co-elliptic Sequence Initiation (CSI) Targeting
P73 CSM Constant Delta Altitude (CDH) Targeting
P74 CSM Transfer Phase Initiation (TPI) Targeting
P75 CSM Transfer Phase Midcourse (TPM) Targeting
P76 Target delta V.

Verb codes

05 Display Octal Components 1, 2, 3 in R1, R2, R3.
06 Display Decimal (Rl or R1, R2 or R1, R2, R3)
25 Load Component 1, 2, 3 into R1, R2, R3.
27 Display Fixed Memory
37 Change Programme (Major Mode)
47 Initialise AGS (R47)
48 Request DAP Data Load Routine (RO3)
49 Request Crew Defined Maneuvre Routine (R62)
50 Please Perform
54 Mark X or Y reticle
55 Increment LGC Time (Decimal)
57 Permit Landing Radar Updates
59 Command LR to Position 2
60 Display Vehicle Attitude Rates (FDAI)
63 Sample Radar Once per Second (R04)
69 Cause Restart
71 Universal Update, Block Address (P27)
75 Enable U, V Jets Firing During DPS Burns
76 Minimum Impulse Command Mode (DAP)
77 Rate Command and Attitude Hold Mode (DAP)
82 Request Orbit Parameter Display (R30)
83 Request Rendezvous Parameter Display (R31)
97 Perform Engine Fail Procedure (R40)
99 Please Enable Engine Ignition

Noun Codes

11 TIG of CSI
13 TIG of CDH
16 Time of Event
18 Auto Maneuvre to FDAI Ball Angles
24 Delta Time for LGC Clock
32 Time from Perigee
33 Time of Ignition
34 Time of Event
35 Time from Event

36	Time of LGC Clock
37	Time of Ignition of TPI
40	(a) Time from Ignition/Cutoff
	(b) VG
	(c) Delta V (Accumulated)
41	Target Azimuth and Target Elevation
42	(a) Apogee Altitude
	(b) Perigee Altitude
	(c) Delta V (Required)
43	(a) Latitude (+North)
	(b) Longitude (+East)
	(c) Altitude
44	(a) Apogee Altitude
	(b) Perigee Altitude
	(c) TFF
45	(a) Marks
	(b) TFI of Next/Last Burn
	(c) MGA
54	(a) Range
	(b) Range Rate
	(c) Theta
61	(a) TGO in Braking Phase
	(b) TFI
	(c) Cross Range Distance
65	Sampled LGC Time
66	LR Slant Range and LR Position
68	(a) Slant Range to Landing Site
	(b) TGO in Braking Phase
	(c) LR Altitude-computed altitude
69	Landing Site Correction, Z, Y and X
76	(a) Desired Horizontal Velocity
	(b) Desired Radial Velocity
	(c) Cross-Range Distance
89	(a) Landmark Latitude (+N)
	(b) Longitude/2 (+E)
	(c) Altitude
92	(a) Desired Thrust Percentage of DPS
	(b) Altitude Rate
	(c) Computed Altitude

Appendix C – TBD

[Ilana: Add Appendix C here]

Bibliography

[AHO] Apollo History Office. “Apollo Expeditions to the Moon” http://www.hq.nasa.gov/office/pao/History/SP-350/ch-4-4.html

[LB] Laning, J. Hal, Battin, Richard H., “Theoretical Principle for a Class of Inertial Guidance Computers for Ballistic Missiles,” R-125, MIT Instrumentation Laboratory, Cambridge, MA, June 1956.

[JON] Jones, James., “Ferrite Core Memories”, Byte Magazine, July 1976.

[HALL] Hall, Eldon., *Journey to the Moon*, AIAA, 1996.

[BAT] Battin, Richard, “Funny Things Happened On the Way to the Moon,” Presentation at Engineering Apollo, MIT,

[BEN] Bennett, Floyd, “Apollo Lunar Descent and Ascent Trajectories,” NASA Technical Memorandum, Presented to the AIAA 8th Aerospace Science Meeting, New York, January 19-21, 1970.

[HHBS] Blair-Smith, Hugh, “Annotations to Eldon Hall's *Journey to the Moon*,” MIT History of Recent Science and Technology, hrst.mit.edu, last updated August, 2002.

[HBS] Hugh Blair-Smith Interview, Cambridge, Massachusetts, April 7, 2005.

[WIK] Wikpedia, www.wikpedia.org

[HOP] Hopkins, “Guidance and Computer Design,” Spacecraft Navigation, Guidance, and Control, MIT, Cambridge, 1965.

[COC] Cherry, George and O'Connor, Joseph, “Design Principles of the Lunar Excursion Module Digital Autopilot,” MIT Instrumentation Laboratory, Cambridge, July, 1965.

[ONG] Ong, Elwin, “From Anonymity to Ubiquity: A Study of Our Increasing Reliance on Fault Tolerant Computing,” Presentation at NASA Goddard, klabs.org, December 9, 2003.

[YEH] Yeh, Y.C., "Safety Critical Avionics for the 777 Primary Flight Controls System," IEEE, 2001.

[BER] Briere, Dominique, and Traverse, Pascal, "Airbus A320/A330/A340 Electrical Flight Controls A Family of Fault Tolerant Systems", IEEE 1993.

[KL] Knight, John and Leveson, Nancy, “An Experimental Evaluation of the Assumption of Independence in Multi-Version Programming,” *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 1, January 1986, pp. 96-109.

[MAD] Madden, W.A., & Rone, K.Y., "Design, Development, Integration: Space Shuttle Primary Flight Software System," ACM, 1984.

[MPL] Euler, E.E., Jolly, S.D., and Curtis, H.H. “The Failures of the Mars Climate Orbiter and Mars Polar Lander: A Perspective from the People Involved”.

Guidance and Control 2001, American Astronautical Society, paper AAS 01-074, 2001.

[ELD] Hall, Eldon. “The Apollo Guidance Computer: A Designer’s View

[NEV] Jim Nevins Interview, Cambridge, Massachusetts, April , 2005.

[FRO] http://www.frobenius.com/7090.htm

[MHA] Margaret Hamilton Interview, Cambridge, Massachusetts, April **TBD**, 2005.

[CUR] Curto, Paul A. and Hornstein, Rhoda Shaller, “Injection of New Technology into Space Systems,” Nautical Aeronautics and Space Administration. Washington, DC.

[MIN] Mindell Interview Transcript, April 26, 2004

[JNE] April 21, 1966 James L. Nevins slides