The Impact of Risk Management:

An Analysis of the Apollo and CEV Guidance, Navigation and Control Systems

Katherine H. Allen

Robbie C. Allen

Ilana L. Davidi

Elwin C. Ong

9 May 2005

16.895J/STS.471J/ESD.30J - Engineering Apollo

Introduction

Building the Guidance, Navigation and Control (GNC) System was among the most significant challenges for the Apollo program. In the 1960s, computers were far from commonplace and were still relatively new and untested. Few of the astronauts were eager to trust their lives to a sewn-together bundle of wires.

The team responsible for building the GNC System was aware of this mistrust. Because of it, they felt that they had to produce a perfect system. However, more was at stake than trust: because it was critical to mission success and the safety of the crew, the system had to work perfectly every time it ran. This requirement was made even more challenging to fulfill considering the GNC team had to rely on unproven technologies (such as integrated circuits) and had to live with many potential single-point failures in order to meet budget and time constraints.

While the term “risk management” was not used during the Apollo program, the engineers were performing excellent risk management by today's standards during the design of the system. For the purpose of this paper, we are defining risk management as follows:

The GNC team was very careful to evaluate the risks associated with various design choices and each decision was scrutinized and checked by many sources. Due to this intense focus on safety and testing, the astronauts and those that directed them were ultimately willing to trust their lives to a computer, which in turn made the Apollo missions successful. The team did not fear risk; they sought only to mitigate it through innovative, yet not-overly-complex technologies.

This report will examine some of the most challenging decisions made during the design of the Apollo GNC System. After we provide a brief overview and history of the GNC System, we will describe the technologies used for the hardware and software and discuss the main risk factors associated with these design choices. We will close by looking at how the lessons learned from the successes and failures of Apollo can be applied to the design and implementation of the Crew Exploratory Vehicle (CEV).

Apollo Guidance, Navigation, and Control System Overview

History

The MIT Instrumentation Laboratory 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. Whether one large rocket or a small lunar module descended to the moon, the spacecraft would need the ability to autonomously guide itself to the moon, land safely, and return the astronauts back to Earth.

The Instrumentation Lab was the pioneer of inertial guidance, navigation, and control. 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].

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 used memory registers which stored 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). However, 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].

Further development of digital computer technologies was performed at the Instrumentation Lab for the MIT Mars Probe project. Although the Mars Probe was cancelled before being fully built, the engineers continued work on the computer and it provided the necessary knowledge and experience needed for the design of the AGC hardware. There was also no doubt, following all the work and achievements of MIT that the AGC design would be digital rather than analog [KLABS,HALL]. Although digital technology was less well known, and hence more risky than analog systems, within the Instrumentation Lab this tradeoff was probably never fully analyzed, as digital technology was a natural progression from all their previous experience in building aerospace computing systems.

Apollo Guidance Computer Team

Work on the design of the AGC was led by Eldon Hall at the Instrumentation Lab. Major contributions were made by many different people, including Ramon Alonso, Albert Hopkins, Hal Laning, and Hugh Blair-Smith.

Eldon Hall had completed an AB in Mathematics at Eastern Nazarene College, an AM in Physics at Boston University, and was completing his PhD at Harvard when the Instrumentation Lab recruited him in 1952. He was key in encouraging the Instrumentation Lab and the Navy to adopt digital computing equipment on the Polaris missile [KLABS,HALL]. He was responsible for the development of the digital differential analyzers used on Polaris. Soon after, Hall was promoted to group leader and formed the Digital Development Group where he led the work on the Mod 1B Mars computer. After the successful flight of the Polaris missile in 1960 and having completed a bread-board version of the Mars computer, his group was poised for the challenge of designing the AGC, and in 1961, the contract was awarded to the Instrumentation Lab.

Hal Laning spent a long career at MIT, having earned an undergraduate degree in Chemical Engineering and a PhD in Applied Mathematics in 1947. He began his tenure at the Instrumentation Lab in 1945 and was soon put in charge of a small group, formally called the Applied Mathematics Group. Laning wrote what is arguably the world’s first compiler, George, for MIT’s Whirlwind computer in 1952 [LOH,SB]. He was vital to the development of the AGC, being responsible for the design of the operating system among many notable contributions.

Ramon Alonso, Albert Hopkins, and Hugh Blair-Smith were all recruits from down the road at Harvard. Alonso joined the Instrumentation Lab in 1957 from Harvard’s Computation Laboratory where he had earned a PhD in computer science [SB]. Along with Hopkins and Laning, they were responsible for the overall architecture of the AGC. Alonso was also responsible for the decision to use of core-rope memory [ALO].

[Elwin: {Need more information on Al Hopkins, am waiting for email back from Hugh}]

Hugh Blair-Smith earned an AB in Engineering and Applied Physics from Harvard in 1957. In 1959, he began work at the Instrumentation Lab working on the cross-compiler program for the Mars Computer and later, the AGC. With Alonso and Hopkins, they designed the instruction set used on the AGC.

Primary Guidance, Navigation, and Control System Architecture

The Primary Guidance, Navigation, and Control System (PGNCS) architecture on board the Lunar Module (LM) included two major components as shown in Figure 1. The AGC was the centerpiece of the system. It was responsible for calculating the state vector (position and velocity) of the vehicle and interfaced with the crew and other systems on board. The second part of the PGNCS was the Inertial Measurement Unit (IMU). The IMU provided inertial measurements from gyros and accelerometers. These measurements were integrated to derive the vehicle's position and velocity.

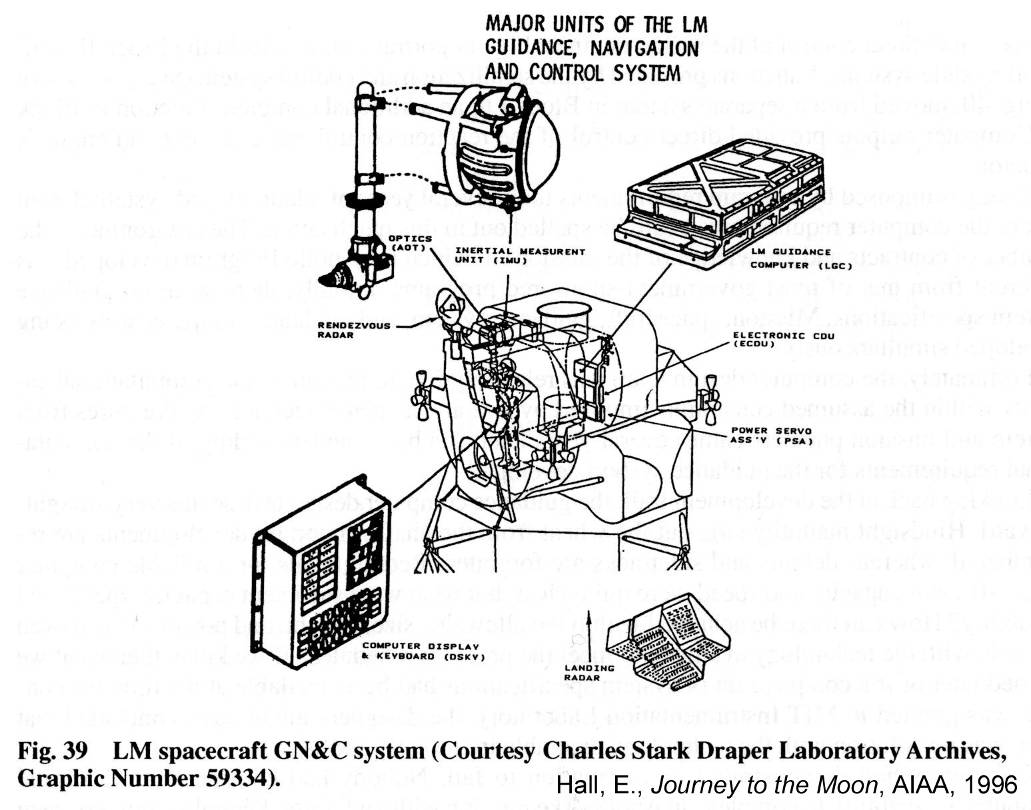


FIGURE 1

Apollo Guidance Computer Hardware Architecture

There were two versions of the AGC flown on Apollo, referred to as Block I and Block II. The Block I version flew in the unmanned missions, while an improved Block II version was used on all subsequent missions. The Block II computer was the heart of the PGNCS used on every LM. 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 as shown in Figure 2. The computer weighed 70.1 lbs, and required 70 watts at 28 volts DC [TOM].

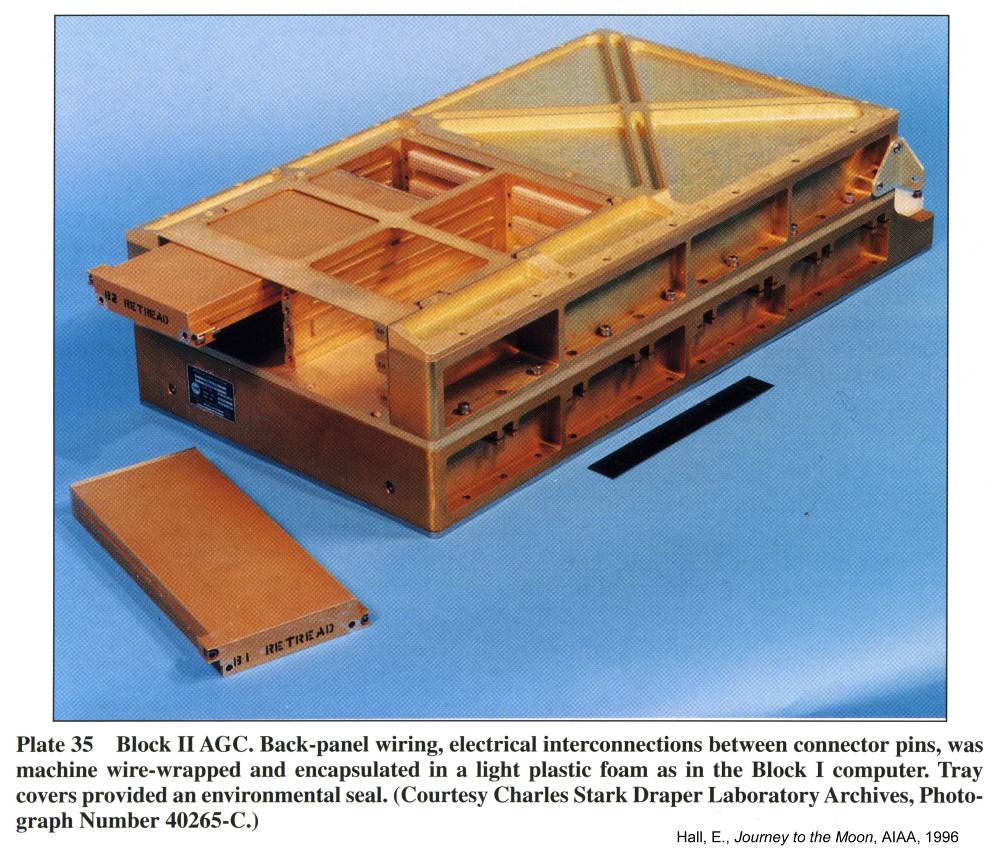
****

FIGURE 2

Display Keyboard

The primary human interface to the AGC was the Display Keyboard (DSKY). It was composed of three parts: the numeric display, the error lights, and the keypad (see Figure 3). The display used an eight-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 was a row of error and status lights, to indicate such important conditions as gimbal lock and operator error. Below the lights and the display panel was a 19-button keyboard. This keyboard featured a nine-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 is provided in Appendix B.



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

FIGURE 3

The design and interface of the AGC may seem crude by today's standards, but this hardware ran without a major failure throughout all of the Apollo missions. We'll now look at how some of the design decisions were made and what tradeoffs and risk mitigation techniques were employed by the AGC team.

Risk Management in Apollo

As demonstrated in the preceding section, the Apollo GNC System was complex. However, steps were taken in the building of each aspect of the GNC to ensure that despite its complexity, it would remain a reliable system. The Instrumentation Lab engineers examined the risks involved with all aspects of the computer, so that they could design the strongest, most reliable machine possible.

Apollo Guidance Computer Hardware

Eldon Hall's team saw the potential applicability of digital computing techniques to space systems because of their work on the Polaris missile and the Mars Probe. However, Apollo required a much more powerful computation system than any of the projects the Instrumentation Lab had previously been involved with. Therefore, the decision to use a digital computer was one of the first major decisions with bearing on risk management. While it is conceivable that an analog computer could have accomplished the requirements of Apollo, the system would have been much bigger and heavier than the eventual digital computer developed by MIT [HHBS]. An analog computer (like those used in airplanes of the day) was preferred by the astronaut-pilots, but had significant drawbacks. It would have been much more difficult to program, and the tasks it performed would have been much more limited than a comparable digital system. For example, an analog computer would not have been reprogrammable in flight. This would have been a disaster for the Apollo 14 mission, where the abort switch had to be reprogrammed or it would have "spoiled Alan Shepard and Ed Mitchell's landing" [EYL]. Though astronaut preference was of key import, the difficulty in programming an analog computer would have left more room for error, and less computer capability increasing risk only to defer to astronaut preference.

One of the ways in which Apollo engineers managed the risk of using a digital computer was to encourage commonality between systems. Two identical digital computers were used on Apollo: one in the CM, and the other in the LM. NASA required the hardware on the two computers to be the same. This simplified production and testing procedures: if a problem were found in one machine, both the LM and CM computers would be adjusted for correction. However, this benefit came with several drawbacks. The commonality requirement meant that the design of the computer was more difficult, as the computer had to interface with equipment in the CM which was not in the LM, and vice-versa. For example, the CM was not intended to ever land on the moon so would not have seemed to need the landing software. Of course, the LM was not intended to be the primary vehicle for the crew. That apparently unnecessary redundancy became suddenly necessary on Apollo 13, where the LM’s ability to act as a lifeboat saved the lives of three astronauts. In addition, since different contractors built the CM and LM, any changes to the computer meant that North American, Grumman, MIT, and NASA had to agree to the changes. The complexity added by having four different groups involved with such a decision were partially allayed by the advantages brought by having both systems the same, but created significant hardships for the design teams.

Apollo Guidance Computer Processor

The AGC processor contained some of the first integrated circuits (IC), a new and unproven technology. Integrated circuits—thin chips consisting of at least two interconnected semiconductor devices—were initially introduced in 1961 [WIK,IC]. ICs permitted a drastic reduction in the size and number of logic units needed for a logic circuit design. (See Figure 4)

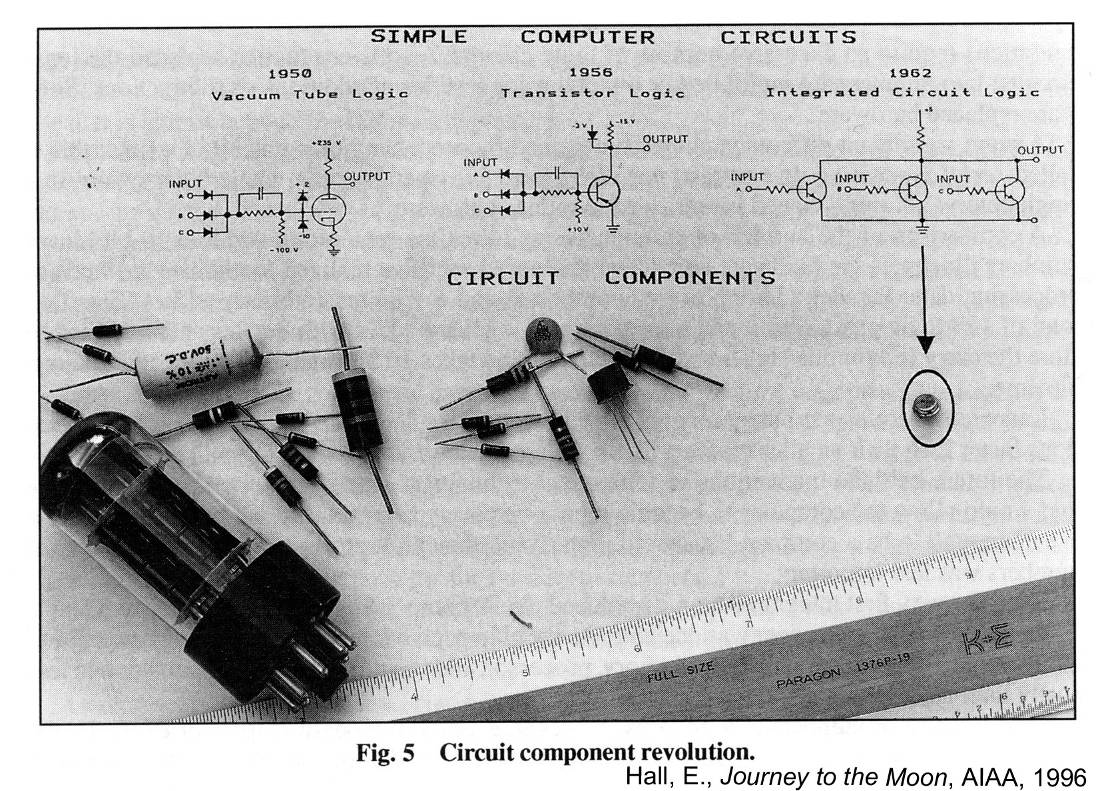


FIGURE 4

The Instrumentation Lab and NASA evaluated the benefits and risks of using ICs thoroughly before making their decision. ICs had never been flown in space; in fact, they had never been used in any computer. More importantly, there was only a single source, Fairchild, which could provide the necessary quantities of ICs for the PNGCS. It was not known whether the rate of production could be maintained throughout the entire program. Hall was able to persuade NASA that the advantages of ICs outweighed the risks involved [HALL,108,109]. This was a significant accomplishment, as “there was resistance both from NASA and people within the Lab who had invested much of their work in core-transistor logic.” [EH] Chief among the advantages were the aforementioned much-needed weight and volume savings. ICs also allowed a significant reduction in the number of electronic components needed (See Figure 5 HALL). One IC component replaced several circuit components for an equivalent core-transistor unit. Fewer components needed meant that more effort could be concentrated on providing strict qualifications and procurement of the single component.

As Hall recalls, the team was quite aware of the risks involved with the decision to use ICs [EH]. The engineers paid attention to the proper qualification and testing of the components at every level of the design. Strict procurement procedures were designed to ensure that the manufacturer provided the best product. These procedures ranged from formal lot screening to sending astronauts on visits to the factory [EH]. By 1963, Fairchild introduced the second generation Micrologic gate, which put two 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.” These new ICs were incorporated into the design of the Block II computer, producing further savings in weight and volume, which allowed more room for the expansion of the memory. The risk taken in implementing ICs was already paying dividends in size and weight reduction.

The pace of IC development was progressing steadily. Before the first Block II computer was produced, Fairchild dropped production of the Micrologic line, electing instead to concentrate production on more advanced chips. This was a risk foreseen by the Instrumentation Lab, and they were fortunate to obtain the services of the Philco Corporation Microelectronics Division who maintained production of the IC for the life of the program [HALL,23].

Apollo Guidance Computer Memory

The AGC had two types of memory. Erasable memory was used to store results of immediate calculations during program execution, while programs were stored in permanent read-only memory banks. The memory used on Apollo was perhaps the least risky component in the AGC. The erasable memory was made from coincident-current ferrite cores, so unlike modern erasable memories (which use electricity to store data in transistors) the erasable memory in the AGC stored information magnetically, such that it would not need to be powered to maintain information storage. [JON]. The main disadvantages of ferrite core memories were that they were relatively large and heavy, requiring more power than the alternative technologies.

Memory capacity was limited on the AGC, as space was strictly limited. Although the challenges of limiting memory capacity posed a significant risk for the program, the decision to use ferrite core memories posed less of a risk than other components of the AGC such as the ICs used on the processor. Ferrite core technology had a longer reliability record than integrated circuits. Ferrite core memories were first used on the Whirlwind computer at MIT in 1951 and later on the Gemini computer [TOM]. Additionally, core memories were much more reliable than drum memory, the only real alternative at the time. Drum memories, which are also based on magnetic principles, were used throughout the 1950s on large mainframe computers. In these memories, information was stored and read by rotating drums of magnetic-coated cylinders with tracks around its circumference [WIK,DRUM-MEMORY]. Drum memories required mechanical movements and were therefore susceptible to mechanical fatigues and failures. In contrast, core memory required no moving parts and was much more compact. Both characteristics gave great advantages in terms of space flight requirements.

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 determining whether the wires passed through the core (representing a one) or outside the core (representing a zero). 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 wires or ropes. The code was literally tangible.

The chief advantage of rope core memory was that it stored a lot of information in a relatively small amount of space [TOM]. However, the memory could not be easily changed after the ropes were manufactured. MIT contracted Raytheon to manufacture the units. Due to the lead time required for manufacturing and testing, the software had to be completed and delivered to Raytheon six weeks in advance [BAT]. Since last-minute changes to the software were almost impossible to implement, the MIT I/L had strong motivation deliver a quality product the first time.

Memory capacity was an issue and an unforeseen risk throughout the design 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 underestimate 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 and increasing the amount of memory required as the project progressed [HBS]. When NASA realized the implications of the issue, they implemented strict control and oversight of the software design process [BT]. This is a good example of how NASA took a proactive approach to managing identified risks.

Apollo Guidance Computer Software

As with the design of the hardware and human interfaces, decisions made during the design of the software were carefully analyzed before implementation. This process resulted in such positive decisions that the experience gained by NASA during their overseeing of the Apollo software development would directly influence the development of the Space Shuttle software [TOM].

AGC Software Architecture

The AGC software was a priority interrupt system. Unlike a round-robin system where jobs are run sequentially, a priority interrupt system was capable of handling several jobs running simultaneously on a single processor. Tasks were assigned a priority and the computer would always execute the job with the highest priority, intervening a more important job when required.

One of the main advantages of a priority-interrupt system is that it is very flexible. Once an operating system was written, new programs could be added quite easily. Conversely, the software was nondeterministic, which made testing much more difficult. Unlike a round-robin system, the sequences of jobs tasked by the computer are infinite. The combination of jobs and their requirements for system resources such as memory could not be predicted; therefore jobs could not be guaranteed completion. To counter the risks posed by these unknown and potentially detrimental sequences, the software developers added protection software that would reset the computer when it detected a fault in the execution of a program.

Hal Laning 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. 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 the other jobs required. Every 20 milliseconds, the Executive checked its queue for jobs with higher priorities [TOM].

The software had some simple fault protection built-in, whereby it would continuously check the amount of resources being used. The computer would reboot itself if it encountered a potentially fatal problem or found that it was running out of computation cycles. After restarting, it would run the most important jobs first. This was a deliberate design feature meant to manage the risks involved with the software. Even when it failed, the AGC had a means of recovery that allowed it to work almost seamlessly. This fault protection software was vital in allowing Eagle to land instead of aborting the mission in the final minutes of the Apollo 11 lunar landing [EYL].

Writing software for the AGC could have been done using machine code, but the software developers at MIT often used an interpretive language because it provided higher-level instructions such as addition, subtraction, multiplication, and division, as well as more advanced instructions including square roots and (vector) dot and cross products. When executed on the computer, each interpretive instruction was translated at run-time into basic computer instructions.

The use of an interpretive language was a new and as–yet-unproven technique at the time. However, the risks associated with using this unproven technique were outweighed by their advantages. Interpretive languages allowed software developers to be far more efficient. Developers could code an equation in a natural form using arithmetic instructions instead of translating the equation into binary form. This process had a more significant advantage in that it facilitated the review process. It is much easier to spot an error in the code when it is written clearly and in a form natural for humans to read. The interpretative language provided these benefits, greatly outweighing the risk of using new technology.

Software Design and Implementation

“Risk management” may not have been a term used in the Sixties, but the care that was applied while developing software for the AGC showed exceptional management of risk. A key design goal of the AGC was simplicity. Margaret Hamilton, one of the lead developers, recalled, “Here, it was elegant, it was simple. But it did everything…no more no less,” as opposed to the more distributed, procedurally-influenced code of today in which “you end up with hodge podge, ad hoc.”[MHA] As the number of lines of code increase, so does the number of potential bugs, so by keeping the code more compact, there was less room for hidden errors.

In many cases, the software developers were forced to be more efficient due to the limitations 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. So in a way, having a handicap gave us a benefit. [MHA]

Due to the long lead time required for the production of the flight software, “there was not the carelessness at the last minute. We went through everything before it went [to be built into the fixed memory].”[MHA]

A large 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 learned were documented in technical memos, many of which are still available and applicable today.

Of the overall Apollo system errors, almost approximately 80 percent were real-time human errors, over 70 percent were recoverable by using software (just prior to landing the software was used in one mission to circumvent the hardware’s erroneous signals to abort in order to save the mission), 40 percent were known about ahead of time but the workaround was inadvertently not used. [ERR]

Risk was also effectively managed by maximizing the commonality of 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 instance, 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.” [MHA]

Software Review and Testing

On Apollo, the combination of a restriction of space and numerous peer reviews kept the code tight and efficient. The pain threshold for each bug discovered was a sufficient deterrent for programmers to do their best to get it right the first time.

Part of the peer reviews involved programmers eyeballing thousands of line of raw code. John Norton was the lead for this task, and the process 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 the code.” [MHA] This included 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 that approximation was used [SAF].

“While in traditional systems engineering, desired results are obtained through continuous system testing until errors are eliminated (curative), the 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].

With the extensiveness of the testing and simulations MIT did on the software, it is surprising any bugs appeared in the code at all. However, it did happen. Dan Lickly, who programmed much of the initial re-entry software, noted 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] Simulations were designed to find as many of these rare occurrences as possible, so they could be identified earlier and eradicated before flight. This was another excellent risk-management procedure.

Human Interface Design

One of the areas most prone to risk is that of human interface. While computers do not need to eat and sleep, their human counterparts must do both and can suffer greatly if they do not. Great care had to be taken to ensure that the human interface was able to mitigate the risks posed by its operators.

DSKY Design

The DSKY had to compensate for two relatively new technologies: space travel and human-interfaces to digital computers. Many times, it was unclear what information the astronauts would find useful while flying or how to best display that information.

Everybody had an opinion on the requirements. As¬tronauts preferred controls and displays similar to the meters, dials, and switches in military aircraft. Digital designers proposed keyboard, printer, tape reader, and numeric displays. [HALL,71]

Although the astronauts’ opinions were greatly valued, their preference for analog displays had to be balanced against the capabilities of a 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.” [HALL,71] This made it difficult for designers to satisfy the astronauts’ desire for aircraft-like displays while still meeting NASA’s deadlines and other requirements. The human interface designers needed to find ways to create a safe, non-risky interface.

Astronauts were not the only ones with high demands for the interface design. 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 of deciphering commands, but would have been very difficult on the astronauts who already had a busy training schedule. The increased training combined with the fatigue of being in space would likely have resulted in mistakes by the astronauts. Allowing a machine to execute these commands was much less risky.

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. [HALL]

This was also rejected, as the fragile vacuum tubes would have been unlikely to survive the G-forces of launch and re-entry. Once again, the examination of risk was a primary factor in development of the AGC.

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 used a novel software concept: “Numeric codes identified verbs or nouns. Computer software interpreted the codes and took action.” [HALL,73] 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.” [HALL,73] Many of the pilots, including Scott, 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 joint development effort ensured that the astronauts would be able to operate the system effectively in flight. It minimized the risks involved with introducing such novel and as–yet-unproven techniques.

Manual Control Hardware and Software

Control System Design

Inside the LM, two hand controllers gave the astronauts the ability to issue commands to the Reaction Control System. However, in order to prevent accidental thruster firings, the control stick used a deadband, a threshold for control stick input below which commands are ignored. In practice, this meant that whenever the hand controller’s deflection exceeded the soft stop at 11 degrees, the manual override switch closed and allowed the astronauts to directly command the thrusters. In this manner, they succeed in enabling human participation—the manual control mode was always available to the pilot and commander, regardless of the guidance mode otherwise selected—while mitigating the risk of accidental inputs wasting reactor propellant.

Another danger inherent in a manually-controlled system is task saturation, a situation where the astronaut is overloaded with information and tasks. To help prevent this, whenever the control stick was not deflected beyond the soft stop, the Digital AutoPilot (DAP) took over, and the astronaut could concentrate on other tasks.

This type of control system division-of-labor epitomizes the risk-minimizing design philosophy of the PNGCS—using digi¬tal autopilot control where it was useful and reasonable to implement, and using manual control where human interaction was beneﬁcial or simplifying.

The PNGCS control system used in Apollo 9, internally called SUNDANCE, used a nonlinear combination of two attitude rates: 20 deg/s for normal maneuvering, and 4 deg/s for fine control. SUNDANCE system had a large frequency deadband which helped to prevent limit cycling, a condition where the system begins to oscillate due to controller phase lag. The cycling could endanger the mission and the crew. Although it increased system stability, and therefore safety, the deadband tended to decrease pilot satisfaction with the system’s handling qualities, since a larger controller input was required to achieve the minimum allowed thrust pulse. This was a problem since it tended to encourage larger pulses than the minimum possible, which wasted reaction control fuel. Astronaut dissatisfaction with the control system was considered to be a large risk, as an astronaut who did not comfortable with the control responses of his craft was much less likely to be able to recover from a dangerous situation.

To address these conflicting risks, the MIT/IL team investigated the correlation of handling qualities with various control system parameters using the LM control stick. The 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) with¬out inducing limit cycles.

Environmental Constraints

A major human factors concern was the maneuverability of an astronaut wearing a pressure suit. The suits of the time were quite bulky, and because they were filled with pressurized gas, they were resistant to bending motions, making it difficult to operate in the crowded spacecraft. Zero-G and high-G environments were of concern to physicians, but also to engineers—the astronauts would have to operate the same controls in both environments. Vibration, also a concern during launch and re-entry, could also make the controls difficult to read and needed to be investigated.

The human factors of each design were investigated primarily by using astronauts and volunteers at MIT 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 Instrumentation Lab had an exact mockup of the CM and LM panels, which allowed proposed hardware displays, switches, and buttons to be evaluated on the ground in a variety of levels of realism. The rigorous experimental testing helped to mitigate the risk of designing systems for environments which were not entirely understood.

Control: Manual vs. Autonomous vs. Automatic

One of the larger risks associated with the entire Apollo project was posed by interference by the Soviets. In order to mitigate the risk to the control of Apollo, the designers wanted to have a ship that was fully autonomous.

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]

However, the Instrumentation Lab engineers wanted more than simple autonomy.

An auxiliary goal of guidance system engineers was a completely auto-matic 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 as¬tronauts became disabled. [HALL59]

Returning to earth with an automatic guidance system would provide a significant boost to astronaut safety, but it might have increased risk due to the increased system complexity.

The astronauts wanted the system autonomous, but with as much capacity for manual control as possible. 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 caused significant disagreement during the control system design and even into the first flights. The designers built automatic systems in, but the astronauts were loathe to trust them unless pressed, which reduced their safety impact.

Nevins related an anecdote about a situation in which Walter Shirra, one of the most automation-resistant of the astronauts, was forced to trust his life to the automatic re-entry system. On Shirra’s Apollo 9 flight, as they were preparing for reentry, the flight checklists were running behind, and, in particular “they didn’t get the seat adjusted properly. They spent a long time making sure those seats were secured, because if they broke, these things are big metal rods, and you’d have a nice hamburg, if you will, of the crew when they get down.” This emergency prevented the crew from properly preparing for re-entry. “They were getting to a point where they could get killed, so Wally saluted the boys up North (MIT/IL) and switched the re-entry mode to automatic. Wally told this story at the crew debriefing—he couldn’t say enough good things about the MIT system after that.”[NEV] The design of the automatic system had mitigated the risk posed to the crew, and saved the mission.

The astronauts were also reluctant to embrace new types of manual control technologies, even when they were safer. The Instrumentation Lab engineers had to prove the safety improvements of their innovations to the astronauts and NASA. To prove that the sidearm controller was superior, they tested the astronauts with a traditional system and the sidearm system “The NASA people made movies of test pilots under 9, 10, 15 Gs, using both systems. With stick-rudder controls they flopped all over the cockpit and they did not with the sidearm.’ Even with that kind of data they still didn’t want [the sidearm controller device].” [NEV]

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

• The system will effectively employ human participation whenever it can simplify or improve the operation over that obtained by automatic se¬quences 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 to keep the astronauts comfortable with the system’s technology, while utilizing the latest control technologies to reduce overall system risk.

System Level Risk Management Decisions

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 GNC system needed to have hundreds of electronic parts, all of which had to operate simultaneously for not only the over 300 hours of the mission, but for the entire mission preparation period. The decision of whether to provide the ability for in-flight maintenance was one that had significant risk-associated implications. The decision was intricately connected to the reliability of the hardware and the ability of the crew to perform the necessary tasks in flight. NASA was aware of the risks posed by having a single string computer and pushed the idea of having a replaceable unit onboard to mitigate the risk of a failed computer in-flight.

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 still thought to be the only way to achieve the necessary level of confidence. [HALL 92].

A more realistic plan to deal with reliability issues was to train the astronauts to replace components in-flight. The engineers at the Instrumentation Lab were quite skeptical.

We thought [in flight-maintenance] was nonsense, 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 un-insulated terminals. The solution for Apollo had been to coat all electronic connections with RTV, which performed admirably as an insulator. [AHO]

This potting prevented moisture from getting into the electronics, but made in-flight repair essentially impossible.

Ultimately, the decision against in-flight maintenance was forced upon NASA by technical infeasibility, but the risk associated with a computer failure in flight was never disregarded. This risk was managed by system-level redundancy. In effect, ground control direction and in-flight computer became parallel systems, each capable of providing the capability to complete the mission. During phases of mission where ground control was ineffective, provisions were made to provide a backup for the AGC. The Abort Guidance System (AGS) was designed for this specific purpose.

Abort Guidance System

The AGS was unique to the LM. Built by TRW, it served as a backup to the PGNCS. If 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. During the final phases of lunar landing, the three-second communication delay meant that guidance from the ground would have been too late to be useful. The AGS was designed and built solely to fill the backup role for this single phase of the mission, but because the PGNCS worked, it was never used in flight.

Summary

Even though formal risk management techniques had not been developed yet in the early 1960s, the designers, developers, and engineers of Apollo were excellent at learning from mistakes and applying this knowledge to prevent future errors. The foresight exhibited by these teams helped create a spacecraft that flew without a major accident.

There are five guiding principles we believe were core to building such a successful system:

1. Do it right the first time: Due to the technology limitations of the time and the understanding that failure could mean loss of life, the focus was getting the system right the first time around and not counting on later revisions to fix major problems.

2. Design – Build – Test: Adequate time was allocated for these three important aspects of building a system. Notably, plenty of time and resources went into testing.

3. Better is the enemy of good: Over-designing a system leads to delays and more complexity than is needed. Limitations imposed by the technology of the period mandated that some systems be simple, but the Apollo management also made good choices about where to innovate and when to stop building in new features.

4. Take risks when necessary: NASA and the Instrumentation Lab weren't risk adverse. They took their fair share of risk, but only where the risk was justified by the benefit, and only after lower-risk alternatives had been evaluated.

5. Build for reliability: Apollo was built to be highly reliable, not necessarily highly redundant. Redundancy encourages the idea that bugs and hardware failures are okay, and can therefore introduce more risks than it eliminates.

To make future missions to space successful, we feel the designers of the next generation spaceship must re-examine these principles and apply them to today's practices.

Risk Management of Crew Exploratory Vehicle (CEV)

One of the easiest metrics by which to judge Apollo’s risk management is success. During Apollo, every man sent into space returned home safely. Later programs were not as successful: the Space Shuttle lost two entire crews. Both accidents can be attributed to various forms of negligence and bad operations or engineering practice—and therefore bad risk management.

The United States is now embarking on building its next generation of spacecraft. President Bush has announced that we will be returning a human to the Moon and will proceed to Mars and beyond. Landing on the Moon is a more complex mission than those undertaken by the Space Shuttle, and it remains to be seen if the new Crew Exploratory Vehicle (CEV) will follow the pattern of success of Apollo or the mixed-results of the Shuttle.

In order to succeed, the Guidance, Navigation and Control System team for the CEV would do well to follow the principles practiced during Apollo. First and foremost, they need to reduce complexity and keep things as simple as possible. Of course, we live in a different world where we often pride ourselves on the complexity of our solutions. To be successful, the CEV must be built efficiently and simply, with a strong eye set to Apollo risk management techniques.

CEV Computing Hardware

The best example of a modern spacecraft computing system is the Space Shuttle Primary Computer System. 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 in parallel. 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 computing architecture will likely resemble the Space Shuttle’s rather than Apollo’s, due to the advances in computing technology since Apollo first launched. 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

When improperly applied, risk management can actually serve to increase risk rather than mitigate it. While we have much more knowledge of computing systems 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]

Because of the nature of the Apollo software we had the unenviable (or enviable) opportunity to make just about every kind of error possible, especially since the flight software was being developed concurrently with hardware, the simulator, the training of the astronauts ,etc., and no one had been to the moon before. In addition we were under the gun with what today would have been unrealistic expectations and schedules. This and what was accomplished (or not accomplished) provided us a wealth of information from which to learn. [HTI2]

The CEV mission software would be one of the most complex 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, but emphasis should be given to simplicity and thorough evaluation and validation. Although tremendously successful, the Space Shuttle software is prohibitively expensive and complex [MAD]. Asingle software system, rather than two separate systems, will be more reliable and easier to operate. 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 and making it more reliable. The requirements for two separate software systems would significantly add to the complexity of the overall system [KL].

In addition, redundant software systems are not guaranteed to be effective. If two groups build from the same specification, and the specification is incorrect, both groups will produce faulty systems. In the words of Margaret Hamilton, "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.” [MHA] Redundancy is not always good risk mitigation; often it only gives the appearance of mitigation and the underlying problems remain.

Today, we can use concurrent and parallel efforts to design the CEV, similar to the methods Apollo used to design the LM and CM. Reuse is assuredly more formalized, but by keeping the design simple without many bells and whistles, the design-sharing should be easy. Said Hamilton, “We would learn from what we did then and make it inherent…[during Apollo] 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 system 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. In addition, “many things that are unexpected are not known.” [MHA], that is, the information available on the mission and the operation of the CEV is necessarily incomplete, as it is intended to be a multipurpose vehicle. Therefore, the architecture should remain open so that modules can be added, removed, or modified as needed. The Shuttle lacked this capability, and suffered because of it.

Modern software business practices are another risk faced by the CEV designers. Today, it is expected that a new Microsoft product will have bugs in it—it will require patches and security updates for basic functionalities. This gives developers the freedom to think it is acceptable to release software with bugs. Although safety critical system designers suffer less from this fallacy than other software developers, the assumption that every software system will have bugs in its first iteration is endemic. Rather than looking for the perfection and ability to “do it right the first time” that Hamilton had demanded of her team, today’s standards have sadly fallen and are more permissive of inadequacies. To be successful, the CEV team will have to take on the failure-intolerant mentality of Apollo, and disregard the failure-permissive mentality pervasive in industry.

Part of what made the AGC team successful was its ability to form a coherent group and to remain in the same company for many years. Employees of today do not show the same commitment and loyalty to their companies that they did in the Sixties. To be successful on the next moon mission, NASA needs to form a team of engineers and designers who are willing to commit to the project for its duration. They should start “auditions” for such teams as quickly as possible. Give the teams smaller projects that are not as critical to the role of the CEV; perhaps they can do other jobs at NASA or be pulled form existing groups there. In order to get this sort of commitment, NASA would need to create lucrative contracts with a pay structure that would guarantee the engineers desired salaries for a number of years; perhaps fixing on a standard and guaranteeing that salary plus a bonus. In addition, encouraging promising young engineers to join the project and giving them significant responsibilities for and input into the design will ensure that the early architects of the system are still active in industry when the CEV is ready to go to the moon in 2020, and for the lifetime of the system.

CEV Automation

As with Apollo, the final decision between a human pilot and a machine pilot will certainly be a political decision, rather than a purely an engineering one. Engineering advances have made automation more feasible—in the forty years since Apollo began, automated systems have become more sophisticated. In combination, the political and technical factors suggest that the CEV will have an 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.

Culture of Safety

Not all of the challenges related to building the CEV are technical. According to Professor Nancy Leveson, an expert in the field of software and system safety, Apollo had a much stronger safety culture than that of the Space Shuttle. NASA is so performance-driven today that safety requirements are often the first thing to be cut when the delivery timeline becomes compressed [NAN]. Concern for safety issues is not constant and is often inversely proportional to concerns for performance. As illustrated in Figure 5, immediately following accidents, NASA's concern for safety noticeably increases. However, the level of concern quickly tapers back down to near pre-accident levels.

Source: Nancy Leveson

Figure 5

Leveson believes that NASA has to anchor its safety efforts externally and move control of the implementation of safety requirements from internal program managers to an external party. That way, when a tradeoff has to be made between safety, performance, or schedule, safety will not be the first cut. Figure 6 estimates the level of risk that is present with and without an independent technical authority guaranteeing safety.

Source: Nancy Leveson

Figure 6

Conclusion

The Apollo missions were a success in large part because everything associated with the GNC was done right the first time. This accomplishment was enabled by early evaluation and mitigation of a variety of risks, from the basic to the most complex to the completely obscure. If CEV is going to succeed, it needs to have similarly rigorous means of risk-aware designs and implementation, risk management, and goal-focused architectures. It needs to be designed to accomplish its tasks rather than to simply showcase the technologies of the variety of companies involved in the project.

Test pilots, the pool from which many astronauts have been drawn, seem to be great risk-takers. However, they are aware of their risks, and do what they can do mitigate them. Those who design and implement the safety-critical systems should maintain the same respect for high-risk activities as the operators of those systems. The Apollo designers maintained this risk awareness. If the CEV designers can do the same, the CEV will be a success.

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 item the verb acts upon. The programs are delineated with “P” and their number (e.g. “P64”), which is how they were referred to throughout the Apollo program. Verbs and nouns are listed only with their number.

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: Definitions

A software error is an unintended phenomenon in tan implementation of the specification for a computer

Appendix C: 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. The design of the manual control software is discussed later in section xxx.

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 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 from 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.

Bibliography

[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,

[HTI] Hamilton, Margaret. “The Heart and Soul of Apollo: Doing it Right the First Time.” MAPLD International Conference, September 9, 2004.

[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 TBD, 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

[ERR] Hamilton, Margaret. “Just what is an Error Anyway.”

[HTI2] Hamilton Technologies, Incorporated. Proposal submitted for shuttle, resubmitted for CEV. May 20, 2004

[EH] Hall, Eldon, Presentation to Engineering Apollo, Cambridge, MA, April 20, 2005.

[BT] Tindall, William, Tindalgrams.

[SAF]

[EYL] Eyles, Don, “Tales from the Lunar Module Guidance Computer,” Paper presented to the 27th annual Guidance and Control Conference of the American Astronautical Society, Brekenridge, Colorado, February 6, 2004, AAS 04-064.

[KLABS,HALL] Hall, Eldon, Biographical Introduction, as Presented at MAPLD 2004, http://klabs.org/mapld04/invited/hall\_bio.htm.

[SB] Brown, Alexander, “When MIT Went to Mars: Academic Entrepreneurs, MIT and the Apollo Program,” available from author.

[LOH] Lohr, Steve, Go TO (New York: Basic Books, 2001), p. 23.

[ALO] Alonso, Ramon, “Evolutionary Dead-Ends,” Presented at MAPLD, 2004.