The Apollo On-Board Flight Software

Margaret H. Hamilton

March 2019

The on-board flight software team developed the software[1] that executed on the Apollo Guidance Computer (AGC) that took humans to the moon and back. Everyone has a unique perspective on how things "were" within the software part of the MIT Apollo project; depending on when a person joined the project, how long one was on the project, what one's experiences were within the "field" before joining the project, what one did while on the project, and where this fit within the overall structure of the project. "Knowledge" was often passed down from others who were involved in the project in earlier times under different circumstances then when the project was in its "heyday".

The biggest challenge: our software was man-rated...astronauts lives were at stake. Because of the never-ending focus on making everything as perfect as possible, anything to do with the prevention of errors was not only not off the table, it was top priority both during development and during real time. Not only did it have to be ultra-reliable, it would need to have the flexibility to detect anything unexpected (e.g., a hardware or astronaut error) and recover from it in real time; that is, at any time during all the time in a real mission. To meet the challenge, the software was developed with an ongoing, overarching focus on finding ways to capitalize on the asynchronous and distributed functionality of the system, at large, in order to perfect the more systems-oriented aspects of the flight software.

A matrix management approach was in place for the organization of people working on the manned missions. There were the line managers, each of whom was in charge of the experts within a particular subject matter area (e.g., the software); and there were the project managers (one for the Command Module, CM, and one for the Lunar Module, LM, for each mission), each of whom served as a mission related interface between NASA and MIT. An invaluable position within the team was that of a "rope mother", the designated caretaker of the software submitted for a particular Apollo mission for either the LM or the CM. For example, one rope mother was the caretaker for the CM software for Apollo 8, another for the CM for Apollo 11, another for the LM for Apollo 11, etc. A rope mother monitored and analyzed all the code in his or her designated area; throughout all the official software releases and their interim updates, from implementation through testing, looking for problems such as interface errors and violations of coding rules. Often, several rope mothers were on the team at a given time, since software for more than one mission was often being developed concurrently. In addition to the group leaders of the functional areas within the software team, the rope mothers reported directly to the leader of the team, who in addition to other responsibilities was by default the lead rope mother.

The software was developed using the AGC assembly language and an interpreted mathematical

[1] Note: "software" ≡ the "on-board flight software".

language. It executed on the AGC with the AGC's operating system. The software was developed by the software people. The AGC and its operating system were developed by the hardware people. The task at hand for the software people (the "software engineers"): develop the software for the CM, the LM and the systems software (areas used by and impacting all the software). The systems software was shared by and resided within both the CM and the LM; it included the design and development of the error detection and recovery programs (for example, restarts and the Display-Interface-Routines' Priority Displays) and the overall design of the software structure (the “glue”) that held everything together as an integrated system of systems and defined the relationships between, among and within mission phases; ensuring that all aspects of the modules such as those related to timing, data and priority were completely integrated.

Our team also designed "software engineering" techniques that included rules, methods, tools and processes for ensuring that the software would be ultra-reliable; both during development and during real time. Techniques evolved for the development, testing and management of the software. Six formal levels of testing took place within a system-of-systems environment. Daily formal releases contained and documented everyone's most recent changes (and the reason for the changes) for each and every mission.

In addition to the software developed by our team, there were others whose "code" fell under our purview. "Outside" code could be submitted to our team from someone in another group(s) to become part of the official software program (e.g., from an engineer in the navigation analysis group). Once submitted to our team for approval, code immediately fell under our supervision; it was then "owned by" and updated by the software engineers to become part of, and integrated with, the rest of the software; and, as such, had to go through the strict rules required of the software; enforced by and tested as a system (within a set of integrated systems) by the software engineers who were now in charge of that area of the software and the software areas related to it.

Updates were continuously being submitted into the software from hundreds of people over time and over many releases for each and every mission (when software for one mission was often being worked on concurrently with software for other missions); making sure everything would play together and that the software would successfully interface to and work together with all the other systems including the hardware, peopleware and missionware for each mission.

The onboard-flight software's systems software was ready. The fact that our software was asynchronous for the manned missions was a godsend for what was able to be accomplished. That is, it had the flexibility to handle the unpredictable: higher priority jobs interrupted lower priority jobs, based on events as they happened. It was up to us developers to determine "before the fact" the relative importance of each process in the software and to assign to it a unique priority to ensure that all events would take place in the correct order and at the right time relative to everything else going on.

Steps earlier taken within the software to create solutions within an asynchronous software environment became the basis for solutions within a distributed systems-of-systems environment. That is, even though only one process is actively executing at a given time in a multi-programming environment, other processes in the same system—sleeping or waiting—exist in parallel with the executing process. With this as a backdrop**,** the Display-Interface-Routines' Priority Displays were able to be created, changing the interface between the flight software and the astronauts from synchronous to asynchronous (the software and astronauts becoming parallel processes within a system of systems). This had never been done before**.** Such was the case with the flight software's error detection and recovery techniques that included its system-wide “kill and recompute” from a "safe place" restart approach to its snapshot and rollback techniques and its Priority Displays together with its man-in-the-loop capabilities.

The Priority Displays warned the astronauts in case of an emergency by interrupting the astronauts' normal mission displays and replacing them with priority alarm displays, providing them with emergency related options from which to select. Such was the case on Apollo 11 just before landing on the moon when the computer, as a result of the rendezvous radar switch having been left in the wrong position, became overloaded. The priority alarm displays were a reminder to the astronauts to put the radar switch back to where it belonged.

The story about the Apollo 11 landing and the Priority Displays was one of error detection and recovery in real time. It was about the astronauts, mission control, the software and the hardware; and how they all worked together during an emergency as an integrated system of systems. It was about creating new, man-machine and software engineering concepts to do things never done before. Unlike a system where the software (or hardware) could "know" of a serious problem without the pilot's knowing it, the Priority Displays were able to determine right away if a particular alarm had occurred that fell within the category of an "*emergency alarm*" and they let the astronauts know about it too.

Since it was not possible (certainly not practical) on Apollo to test the software "before the fact" by ”flying” an actual mission, it was necessary to test the software by developing a mix of hardware and digital simulations of every (and all aspects of an) Apollo mission which included man-in-the-loop simulations (with real or simulated human interaction); and variations of real or simulated hardware and their integration in order to make sure that a complete mission from start to finish would behave exactly as expected.

Collaboration was a given. It took place among co-workers, between groups within MIT-IL7 and between the organizations involved. This included working with others, interacting with others, interfacing to others, learning from each other, working on things together, working out things together; it was more often than not an iterative effort that was in play such as that between developers or between the developers and users (both on a lower and higher level such as between MIT and NASA). We all worked together as a team; partly because of the dedication of everyone involved and, partly because of the project's evolution based on lessons learned along the way. As an example, the Display-Interface-Routines' Priority Displays would not have been possible without an integrated system-of-systems (and teams) approach and contributions made by many other groups working together with our team to support this becoming a reality [1,2,3]. The hardware team at MIT changed their hardware and the mission planning team in Houston changed their astronaut procedures; both working closely with us to accommodate the Priority Displays for both the CM and the LM; for any kind of emergency and throughout any mission. Mission Control was well-prepared to know what to do if and when the Priority Displays interrupted the astronauts.

Engineers responsible for the software's development began the practice of giving each mission flight program an identifying name. The practice was informal to begin with, but has since been adopted as official nomenclature associated with Project Apollo. The program that controlled an unmanned command module in the suborbital Apollo flight test was called CORONA[4]. Other programs were:

SOLARIUM	- unmanned earth orbital command module flight program.
SUNBURST	- unmanned earth orbital lunar module flight program.
SUNDISK	- manned earth orbital command module flight program.
SUNDANCE	- manned earth orbital lunar module flight program.
COLOSSUS	- the flight program for the command module for manned flight to the moon.
LUMINARY	- the flight program for the lunar module for manned flight to the moon.

As developers, we were given the opportunity of a lifetime—to make every kind of error humanly possible. We were handicapped by the computer's time and space constraints, giving "software experts" the license to be “creative", resulting in tricky programming. Requirements were "thrown over the wall" by "non-software experts" who assumed that all the software programs would somehow "magically" interface together. Fortunately, this was not the case. For, if it had been, we would never have learned what we did about errors and how to prevent them. Further, because of the computer's constraints and because of the asynchronous environment, responsibilities and data storage locations were shared among and within programs where lower priority processes were continuously being interrupted by higher priority processes during each and every mission phase.

Although there were more than enough opportunities to make errors, there were now the opportunities to come up with new ways to prevent them. We evolved “software engineering” rules and techniques with each relevant discovery. Although many errors were found during the software’s pre-flight phases, no software errors were known to have occurred during flight on any of the Apollo missions.

References

[1] Hamilton, Margaret H., "The Language as a Software Engineer", International Conference of Software Engineering (ICSE 2018) that celebrated the 50th Anniversary of Software Engineering and the 40th of the conference itself: https://www.youtube.com/watch?v=ZbVOF0Uk5lU

[2] Hamilton, Margaret H., "What the Errors Tell Us", September/October issue of the IEEE Software Magazine for the 50th anniversary of Software Engineering: https://www.computer.org/csdl/mags/so/2018/05/index.html

[3] Hamilton, Margaret H., "Computer Got Loaded", letter to the editor of Datamation, March 1971

[4] Apollo Moon Show celebration, IL/MIT/NASA, 1968-1969