- Chapter One -
- The Gemini Digital Computer: First Machine in Orbit -

Software

[17] In l962. hardware was still the pacing factor in computer applications. Everything associated with computers-processors, memories and input/output (I/O) peripherals&emdash;was expensive. Many considered software development an incidental part of the overall applications of computing. Specialists wrote most of the software, usually in arcane assembly languages. FORTRAN, a high-level language, had only [19] been available for a few years. Although its use in technical applications was rapidly spreading, it was still considered too inefficient for use on computers like the Gemini digital computer. Many thought its compiler-produced machine code to be less effective in utilizing machine resources than machine language programs written by humans. Experts therefore developed applications programs for Gemini using the tiny set of 16 instructions that the computer could execute[28]. This sort of programming was considered to be more of an art than a science. Whereas the design and construction of computer hardware followed conventional engineering principles, software development was largely haphazard, undocumented, and highly idiosyncratic. Many managers considered software developers to be a different breed and best left alone. This concept of software is a myth, and although it persists in some companies and with some people today, by and large software is now considered as an engineered product, little different from a rocket engine or computer.

(MISSING PICTURE)
Figure 1-5. Auxiliary Tape Memory in test. (IBM photo)

Although the term "software engineering" did not come into common use until 1968, programmers had applied its basic tenets to both large and small software projects for at least 15 years. Software engineering has evolved as programmers learned which techniques worked, which did not, and what actually occurred in the development of software products. The SAGE (Semi-Automatic Ground Environment) air defense system[29], the IBM 360 operating system[30], and NASA’s requirements for both spacecraft software and ground-based software were instances of major software projects that directly contributed to the evolution of software engineering.

Software engineers recognize that software follows a specific development cycle, from formal specification of the product, through the design and coding of the actual program, and then to testing of the product and postdelivery maintenance. This cycle lasts for many years in the case of programs such as operating systems, or a short period of time in the case of specialized, single-use programs. During this development process, strict standards of documentation, configuration control, and managing changes and the correction of errors must be maintained. Also, breaking down the application into smaller, potentially interchangeable parts, or modules, is a primary technique. Communication between programming teams working on different but interconnected modules must be kept clear and unambiguous. It is in these areas that NASA has had the greatest impact on software engineering.

Development of the Gemini software was a learning experience for both NASA and IBM. It was, of course, the first on-board software for a manned spacecraft and was certainly a more sophisticated system than any that had flown on unmanned spacecraft to that point. When the time came to write the software for Gemini, programmers envisioned a single software load containing all the code for the flight, [20] with new unique programs to be developed for each mission[31] . Soon it became obvious that certain parts of the program were relatively unchanged from mission to mission, such as the ascent guidance backup. Designers then introduced modularization, with some modules becoming parts of several software loads.

Another reason for modularization is the fact that the programs developed for Gemini quickly exceeded the memory available for them. Some were stored on the Auxiliary Tape Memory until needed. The problem of poor estimation of total memory requirements has plagued each manned spacecraft computer system. In the case of Gemini, changed requirements and attempts to squeeze the programs into the allotted space resulted in nine different versions of the software[32]. The different versions were referred to by the name "Gemini Math Flow."

Tracing the development of the math flows shows how identifying new functions caused initial memory estimates to be wrong and how the project handled changes. Math Flow One consisted of just four modules: Ascent, Catch-up, Rendezvous, and Re-entry. Math Flow Two was proposed to add orbital navigation and re-entry initialization, but it caused the overall load to exceed the memory size and the Gemini program office canceled the additions[33]. This version of the software flew on spacecraft II in January 1965. By Math Flow Four, the re-entry initialization program had been successfully added, but the load took up 12,150 of 12,288 available words. The plan had been to use this program on spacecraft III and others, but a NASA directive of February, 1964 changed the guidance logic of the re-entry mode to a constant bank angle rather than a proportional bank angle and constant roll rate. Math Flow Five incorporated this change, but it filled the memory and was scrubbed in favor of a modified Math Flow Three on spacecraft III and IV, followed by Math Flow Six containing some changes on spacecraft V through VII [34]. The final version, Math Flow Seven, was used on spacecraft VIII through XII, all of which incorporated the Auxiliary Tape Memory. It had six program modules with nine operational modes. The six program modules of Math Flow Seven were Executor, Prelaunch, Ascent, Catch-Up, Rendezvous, and Re-Entry[35]. The Executor routine selected other routines depending upon mission phases.

The specification procedure for the software required McDonnell-Douglas to prepare the Specification Control Document (SCD). This was forwarded to the IBM Space Guidance Center in Owego, which developed a FORTRAN program to validate the guidance equations. The use of simulations such as the FORTRAN program was endemic to the Gemini software effort and was later applied to software development for other spacecraft computers.

Gemini used three levels of simulations, beginning with the equation-validation system. The second was a man-in-the-loop [21] simulation to help define 1/0 requirements, procedures, and displays. The third level was a refined digital simulation to determine the performance characteristics of the software, useful in error analysis. This third level was carried out in the Configuration Control Test System (CCTS) laboratory, which contained a Gemini computer and crew interfaces. This Mission Verification Simulation (MVS) ensured that the guidance system worked with the operational mission program.

Further tests of the software were done at McDonnell-Douglas and at the pad [36]. NASA and IBM emphasized program verification because there was no backup computer or backup software. The verification process and the tools developed for it were later applied to military projects in which IBM became involved[37].

Even if the software is perfect, errors may occur because of transient hardware or software failures during operation due to power fluctuations or unforeseen demands on real-time programs. Some of these can be spotted by diagnostic subroutines interleaved in the software and used for fault detection [38]. Such routines were put in the Gemini software and are now a part of all IBM computer systems.

The software produced during the Gemini program was highly reliable and successful. Techniques of specification development, verification, and simulations developed for Gemini were later applied to other IBM and NASA projects. NASA was certainly better prepared to monitor software development for the much more difficult Apollo program.
