S. E. James

Evolution of Real-Time Computer Systems for Manned Spaceflight

This paper describes the evolution of ground-based command and control systems used to support NASA's manned spaceflight program; it is an encapsulation of twenty years of development of real-time command and control systems at NASA's Real-Time Computer Complex (RTCC) in Houston, Texas. A brief description of manned spaceflight programs, their accomplishments, and IBM's involvement is provided as background information. Emphasis is given to the development of RTCC systems, as well as to the technological and architectural changes affecting this development. Also described are experiences gained in the management of complex, real-time software systems and the tools and techniques used in the development process.

Introduction

IBM has designed, developed, and supplied computer hardware and software systems to support the U.S. manned space program since its inception. From Project Mercury, America's first venture into space, to the current Space Shuttle program, the primary U.S. space transportation system for the remainder of this century, IBM has contributed to the advancement of the data processing systems. Included have been onboard computers and programming for Gemini spacecraft, Saturn launch vehicles, the Skylab space station, and Space Shuttle orbiters. Additionally, the Corporation has pioneered ground systems to support mission control activities by providing computers and developing software systems for the real-time computer complexes (RTCCs) of every manned spaceflight program to date.

This paper is intended to highlight some of the more significant factors in the evolution of RTCCs and to summarize important advances made in technology, system architecture, and the development process. A brief summary of U.S. manned spaceflight programs is followed by an overview of the RTCCs used to support these programs. The paper further details the evolution of RTCC hardware and software from both architecture and technology perspectives, and concludes with some of the significant lessons learned concerning the software systems development process.

U.S. manned spaceflight programs

Although studies were made by the U.S. Air Force as early as 1956, serious planning for a manned space program did not begin until Project Mercury, which had its technological roots in studies of the National Advisory Committee for Aeronautics (NACA), predecessor of the National Aeronautics and Space Administration (NASA). High points of Project Mercury were the launching of the first American into a 15-minute sub-orbital flight, and the launching of the first American into earth orbit [1].

Project Gemini was the next venture in U.S. manned spaceflight; it used a two-man spacecraft. Significant firsts accomplished during this program included astronaut-initiated spacecraft maneuvers, rendezvous of two spacecraft, extended spaceflights equivalent to a lunar mission, a space walk by an American, and astronaut-controlled reentry and landing of a spacecraft [2].

By 1961, Project Apollo began to take shape to fulfill the national goal of a manned lunar landing by the end of the 1960s. Three Apollo missions especially captured the attention of the world: the flight of Apollo 8, when three Americans became the first human beings to orbit the moon; the Apollo 11 mission, which was the culmination of centuries of anticipation as human beings walked the lunar surface [3]; and the flight of Apollo 13, which jeop-

ardized the mission and lives of the crew when an oxygen tank exploded aboard the spacecraft.

With the success of the lunar landing program, the nation's manned spaceflight efforts turned towards an earth-orbiting space station as the next venture. The Skylab program differed significantly from previous programs in that the objectives were almost entirely science- and engineering-oriented. Despite damage to the huge unmanned space station during the initial launch [4], three separate astronaut crews visited the orbiting laboratory for extended periods of time to establish the feasibility of the commercial uses of space.

An historic occasion in astronautics took place on July 17, 1975, when an American Apollo spacecraft docked with a Soviet Soyuz spacecraft in earth orbit. The route to this historic event was not without problems involving language and engineering practices, as well as the "interface" between the metric and English systems of units [5].

Today, the U.S. manned spaceflight program centers on the reusable Space Shuttle, a more complex spacecraft than its predecessors, designed as a rocket, spacecraft, and glider. Once operational, the shuttle will allow travel to and from space for performing a variety of scientific and engineering activities.

RTCC systems for manned spaceflight

The basic purpose of the RTCC systems for manned spaceflight has remained the same through the years: to perform the necessary computations from tracking and telemetry data to provide Mission Control Center (MCC) flight controllers with the information needed to ensure crew safety and mission success. The fundamental driving force in the evolution of these RTCCs has been the requirement to perform real-time command and control, for increasingly complex spacecraft with increasingly complex missions, with a high degree of reliability. RTCC system architecture changes have been driven by these requirements as well as by advances in hardware and software technology and improvements in the development process.

- *Vanguard RTCC*

Even before America's first astronauts orbited earth, IBM pioneered the requisite computational technology for manned spaceflight through its role in Project Vanguard, the nation's first official artificial satellite program. The Vanguard RTCC was established in June 1957 at the Vanguard Computer Center in Washington, DC. It consisted of an IBM 709 computer with a backup facility located at IBM's research computing center in Poughkeepsie, New York. Working in coordination with Naval Research Laboratory personnel, IBM programmers wrote a 40 000-instruction program for the Vanguard RTCC to support orbit determination calculations and tracking data processing for the unmanned satellites [6].

- *Mercury RTCC*

The Mercury RTCC, established in 1960 at the NASA Goddard Space Flight Center (GSFC), Greenbelt, MD, represented a significant advance in the evolution of RTCC systems. It consisted of three IBM 7090 computers and associated communications and display equipment. Operating in conjunction with NASA's worldwide network of tracking sites and the launch control center at Cape Canaveral, the Mercury RTCC processed real-time tracking data from simultaneous sources to provide a continually updated status of the spacecraft [7].

The computational support for the Mercury program was a radical departure from earlier support of unmanned satellites because it was *real-time*. The RTCC computers had to accept input data from multiple sources simultaneously. The display technology progressed from analog to digital. The critical nature of supporting manned spaceflight meant that spacecraft orbit go/no-go computations had to be completed within ten seconds of booster cutoff.

- *Gemini RTCC*

The distinguishing characteristic of the Project Gemini RTCC was that it had to control two maneuverable spacecraft in orbit at the same time. Since existing ground support capabilities were inadequate, a decision was made to establish the Mission Control Center at the Manned Spacecraft Center (now Johnson Space Center) in Houston to handle all multi-spacecraft missions. In October 1962, IBM received a prime contract from NASA to develop the Gemini RTCC in Houston.

The Gemini RTCC, which began spacecraft monitoring in 1965, consisted of five IBM 7094 computers, interconnected through a switching network, and associated communications and display equipment [8]. The configuration of the five computers allowed two practice missions or a practice mission and an actual mission to be run simultaneously. Processing requirements were expanded to include simultaneous real-time processing of both telemetry and tracking data. The computing load was further increased by the requirements to plan and command orbital maneuvers for the rendezvous of two spacecraft.

- *Apollo RTCC*

Project Apollo placed an even greater demand on the RTCC. To track the Apollo spacecraft, the Lunar Module, and the Saturn launch vehicle, the facilities of the

RTCC would have to maintain constant contact with the astronauts from lift-off through insertion into earth orbit, insertion into cislunar trajectory, then lunar orbit, lunar landing, lift-off from the moon, rendezvous and docking with the orbiting Apollo spacecraft, and finally the return to earth for direct reentry and landing in the Pacific Ocean. To accomplish this task, the RTCC was upgraded to five IBM System 360/Model 75 computing systems and interconnected through a switching network similar to the Gemini 7094 network [9]. The roles of each 360/75 were similar to those associated with the 7094s. However, the processing load was significantly increased. The significance of the Apollo RTCC requirements can be visualized from its seven major real-time program subsystems:

- Launch subsystem—cyclically updated position and velocity utilizing data from the impact-predictor computer at Cape Canaveral, multiple spacecraft sources, and multiple tracking radar sources.
- Telemetry subsystem—received and processed telemetry data indicating performance of vehicle systems, astronaut health, and spacecraft computer status.
- Orbit computation subsystem—calculated ephemerides for the Apollo spacecraft and Lunar Module, determined remote site acquisition, and performed spacecraft attitude and pointing computations.
- Trajectory determination program subsystem—processed radar data to determine the trajectory of the spacecraft, detected radar errors, and provided information on the lunar landing site.
- Mission planning program subsystem—calculated spacecraft maneuvers required for plane changes and phase changes, and to accomplish translunar injection, mid-course corrections, lunar-orbit insertion, descent to the lunar surface, ascent from the surface, rendezvous of lunar module with command module, trans-earth injection, and final mid-course corrections.
- Digital command subsystem—uplinked data to the computers aboard the spacecraft and the Saturn launch vehicle.
- Reentry subsystem—simulated the onboard computer, predicted the reentry trajectory, and alerted remote tracking sites to acquire the spacecraft.

- *Skylab and Apollo Soyuz test project RTCC*

Skylab and ASTP RTCC real-time mission systems were similar to the earth-orbit portion of the Apollo RTCC and will not be discussed here.

- *Space Shuttle RTCC*

The Space Shuttle is proving to be just as demanding as sending astronauts to the moon, though the astronauts of Space Shuttle will not leave earth orbit. The Shuttle RTCC requirements are much more extensive and complex than the Apollo earth-orbit requirements. The complexity of the Shuttle orbiter, the need for pinpoint landing of the orbiter on a runway, the need to process data simultaneously for three orbiters, the need for rapid orbiter turn-around between flights, and the very high flight-rate are the principal Shuttle RTCC requirements. Another significant factor is the need to provide an extensive command and control capability for attached payloads.

IBM's Space Shuttle RTCC involvement began in June 1974 with a contract to design and develop the RTCC programming for the Shuttle Mission Control Center at the Johnson Space Center. Two years later another contract was awarded to IBM to provide the RTCC with three IBM 370/168 computers. Conceptually, the three computers do what their counterparts in Projects Mercury, Gemini, and Apollo had done: provide centralized ground control of the space vehicle from lift-off, through orbit, to reentry. However, they provide three times the computational power of the 360/75 computers used in the RTCC during Apollo. Additionally, they support payload data processing, personnel training, and MCC testing and checkout. Generally, one computer provides flight control as the Mission Operational Computer (MOC). During critical periods of a mission, however, two of the computers process data simultaneously while the third remains in a standby mode or serves as the Payload Operations Control Computer (POCC) [10]. The POCC provides capabilities to monitor and control experiments (payloads) carried into orbit by the Space Shuttle.

The Shuttle data processing application programs form one of the largest and most complex real-time programming systems ever produced. Fundamentally, these programs process telemetry and radar data into information for flight controllers to make decisions and take subsequent actions. The real-time portion consists of five subsystems (with over 600 000 lines of programming): trajectory, telemetry, command, network communications, and control.

The trajectory programs comprise the largest of the applications subsystems (some 220 000 lines of code). Monitoring and evaluation functions are performed during the launch and landing phases, where radar tracking data are converted to vehicle position and velocity and utilized to determine and evaluate spacecraft maneuvers. Prediction and planning routines are used during orbit to compute information such as sunset and sunrise times and spacecraft maneuvers for orbit, reentry, and landing.

The telemetry programs, with more than 138 000 lines of code, are designed to receive more than 3600 parame-

ters of discrete measurements per second from onboard systems and an additional 3600 parameters per second of recorded data from a tracking site. The telemetry subsystem converts these raw data into engineering units and checks certain parameters of data against predetermined limits to assist in monitoring the status of the onboard systems.

The command subsystem formats and provides data for transmission to the Shuttle onboard computers. Among the tasks accomplished are the real-time transmission of commands to back up or relieve crew members of tasks that can be performed in mission control. The command subsystem also uplinks navigational data, computations for maneuvers, and changes in crew procedures, as well as updating software for the onboard computers.

The network communications subsystem configures and monitors the status of communications into the Mission Control Center (MCC). It also configures the MCC to receive and route data from the tracking and communications network to the appropriate MCC destinations for processing.

The control subsystem ties the other four applications subsystems together by initializing systems, managing work and data, interfacing terminals, managing and formatting displays, and recovering from errors. The control subsystem consists of over 135 000 lines of programming and frequently interfaces with the operating systems services by requesting data access routines, storage control, and restart processing.

In addition to these real-time programs, support programs for checkout and configuration processing are provided along with special programs for the POCC. The checkout programs, consisting of over 115 000 lines of code, generate and analyze data for testing the command and control software. The configuration processing program maintains a file of information to produce configuration tables for the command and control subsystem programs and consists of over 200 000 lines of programming, using IBM's IMS/VS data base management program. The POCC programs provide both developmental support and real-time payload command and control to Shuttle payload users.

Development of RTCC systems

- *Systems architecture*

The evolution of RTCC systems architecture has been driven by ever-increasing requirements, advances in hardware and software technology, the need to improve productivity, and the maturing experience of designers and managers. This section summarizes some of the more significant advances in software systems architecture.

Control program evolution

An essential element of any real-time system is the executive or control program, which provides the interface to hardware components and controls the sequence of operations. The architecture of control programs has evolved throughout the era of manned spaceflight. Some of these changes have been driven by expanded mission requirements, but many have resulted from taking advantage of improving technology in both hardware and software products. The trend in RTCC systems has been to utilize more and more off-the-shelf products, thus reducing initial development costs and ongoing maintenance costs while improving transportability to other systems.

During the Mercury era, the control program was a special-purpose monitor providing limited multi-programming capabilities. The Mercury monitor was tailored specifically to support the real-time mission operational environment with separate facilities provided to support the job-shop environment used for software development.

A new executive program developed for the 7094 computers for Gemini was again specifically designed to support the Gemini mission environment. Some advancements were made in providing more generalized services which were applicable to both the simulation and mission support; features such as limited device independence were introduced. While the separate executive was required for mission support, the trend toward the use of a standard operating system began with the use of the IBM 7094 system (IBSYS) to support standard job shop.

The standard operating system provided for the Apollo RTCC 360/75 computers represented a significant advancement. Features such as multitasking and dynamic storage allocation were a part of the standard operating system. The Apollo control program took full advantage of these features, but because of the stringent response time and reliability requirements to support manned spaceflight, additional features were still required. The Apollo Real-Time Operating System (RTOS) was developed during this period. RTOS was a customized version of the standard IBM 360 Operating System, and achieved a significant advancement with the ability to support all RTCC activities, from job shop to real-time operational support, with a single control program.

As the spaceflight activity moved into the Space Shuttle era, the use of standard operating systems continued to increase. All phases of the development process and mission operations are supported with the standard

IBM 370 Operating System (MVS). Standard features such as virtual memory are used extensively. In order to achieve this, while still supporting the extreme response time and reliability requirements of a manned spaceflight mission, the concept of providing centralized system services to interface with the standard operating system was introduced. The system services execute as normal application programs, but provide services which are required by a number of the operational programs. Features normally associated with the control program, such as data routing and error recovery, are accommodated without modification to the standard operating system.

The maximum use of standard operating systems supplemented by common systems services routines has been an unqualified success during the development of the Shuttle RTCC. The cost of maintenance has been reduced and the transportability of the system has been demonstrated.

Use of process control tables

A fundamental architectural concept of the Shuttle real-time command and control system is its use of process control tables. These tables are tabular specifications of the actions to be performed by the executable real-time routines. Their elements generally consist of processing codes, control codes, pointers to related data, and data constants.

Usage of process control tables began during Gemini and increased for Apollo. The prime motivator was the goal of divorcing the basic execution logic from the specific processing descriptions to facilitate testing, since tables can usually be revised or replaced with little or no real-time code change. To meet the Shuttle program's unique environment of rapid flight turn-around and concurrent support for multiple, differently configured vehicles, the process control table concept has been broadened to maximize both the flight-to-flight and in-flight reconfigurability of the real-time system.

Numerous new table applications were designed for Shuttle. Previous telemetry subsystems, for example, used tables to unpack, calibrate, limit-sense, and display vehicle data. The Shuttle software design extends to extracting the subsets of telemetry parameters required by the trajectory and command subsystems via table-driven routines. Also controlled via tables are the computing of additional quantities and event indicators from the converted input data. Even the specifications for processing sequences, which can vary depending on the vehicle and data class, are controlled via tables; and multiple table versions afford dynamic control over the number of parameters processed each second. A major advance engineered in the trajectory and command subsystems was implementation of a table-driven data retrieval process. It collects data from an assortment of tables of varying structures for input to the spacecraft command load generation and trajectory/command display functions.

Software generalization is the key feature of the Shuttle table-driven design. Because orbiter flight turn-around will approach fourteen days in the mature operations era, flight-specific software development and management of the attendant multiple software versions would be impractical. Hence, the real-time software is designed to support a wide range of functional capabilities; flight- and vehicle-specific definitions are introduced into the system solely through off-line-generated process control tables. Additionally, much of the real-time software is reentrant code; that is, a single program copy can be executed concurrently to support several different vehicles, data classes, or other processes. This feature combines with the process control table organization and the extensive user configuration controls to optimize efficient computer memory utilization. The process control tables are stored on disks, by flight, in a hierarchical structure. At the top level of the hierarchy are tables containing specifications applicable to all vehicles within a flight; the next level contains vehicle-specific specifications; and at the bottom level are those specifications that vary by data class (real-time, playback/dump) for a given vehicle. Based on a sequence of user configuration requests, access paths are dynamically established, and only those tables required at a particular mission point are loaded into computer memory. Even a totally new flight definition may be copied to the on-line disk storage and subsequently accessed for real-time support without impact to the ongoing flight activities.

Much of the processing specification is under further user control at the individual entry level, to enable rapid changes in flight. Generally, these in-flight reconfigurations take the form of changing data constants (*e.g.*, calibration coefficients, limit-sensing values, mathematic-formulation values) or process controls (*e.g.*, disabling a parameter's limit sensing, redefining the parameter set on a configurable display). Changes generally can be made either to memory- or disk-resident table copies, with memory-resident changes usually preserved by copying the tables back to secondary storage when the software configuration they are supporting is finished.

As the real-time system has undergone significant evolution, so have the off-line programs which produce the process control tables. For previous projects, the reconfiguration task was accomplished by a collection of independent preprocessors, each generally consisting of spe-

cially developed software to create intermediate data bases which were subsequently accessed for in-storage construction of tables. Other tables were coded and assembled using Basic Assembler Language or special data-table-formatting macros.

For Shuttle, these preprocessors have been consolidated into a single Configuration Requirements Processing (CRP) system, which provides a much faster, more efficient, and more reliable system to configure a specific Shuttle flight. CRP runs on the 370/168 and uses IMS/VS for data base management. The system is capable of processing multiple flights and multiple updates to a single flight concurrently, and incorporates significantly more automated input and product validation processing than its predecessors.

Managing complex systems memory and CPU resources

Computer memory and CPU resource management have continually been a challenge in RTCC development, since requirements have tended to drive the systems to the limits of their resources. The method used to manage RTCC CPU and memory can be characterized as controlled, prioritized competition among cyclic and demand/response application functions. As a part of system design, application functions are characterized in terms of resource utilization, frequency of execution, and allowable-elapsed-time-to-complete (response time).

Demand/response functions are user-requested and execute at irregular intervals. The definition of an explicit required response time for these functions is critical, since they generally execute at the lower end of the priority structure. The most important of these functions, from a system performance point of view, are those which require significant CPU time and memory to complete. These typical functions perform complex trajectory-related computations such as trajectory determination, ephemeris generation, and maneuver planning, and are serialized to prevent unreasonable competition for resources among themselves.

Cyclic functions are driven by timers or by the regular arrival of data. Typical functions are those which process telemetry data from the network and those which output display information on a time cycle. Their frequency of execution is defined explicitly, and execution time is extremely important since they generally execute at the top of the system priority structure. Each of these functions must complete sufficiently ahead of its next execution to allow lower-priority functions to meet their explicitly stated response times.

At the beginning of the Shuttle RTCC design phase, specific memory and CPU limits were established as design constraints. Systems designers and analysts then put together a profile of application functions which allowed them to predict the percentage of CPU and memory needed for cyclic functions. Each of the demand/response functions was then analyzed to determine whether it could meet its required response time within the remaining CPU and fit within the remaining memory. Design alternatives and requirements tradeoffs were then exercised until the projected system performance was within the design constraints. Memory and CPU budgets were established for the overall system and each major system component. System performance was managed against established budgets throughout the development period with the same rigor used in managing cost and schedule.

Reliability through redundancy

The reliability requirements for RTCC systems have remained similar from Gemini through the Shuttle program: during critical periods of flight operations, the system must achieve a 0.9995 reliability and, during noncritical flight operations, if system support is lost it must be recovered with a relatively current data state. The basic approach to satisfying these requirements has also remained the same.

For critical flight operations, the approach to satisfying the 0.9995 reliability requirement is through redundant systems with a prime Mission Operational Computer (MOC) system backed up by a fully functioning Dynamic Standby Computer (DSC) system. Locally developed system software provides a High-Speed Restart function, which creates a DSC as a mirror image of a functioning MOC in preparation for critical operations. In addition, a Selectover function provides DSC-to-MOC role switching, in case MOC support is lost.

During noncritical operations, a single command and control system (the MOC) is used. Locally developed system software provides a System Tape Checkpoint function which, on user request, transfers to tape the total system state for subsequent use in case of a loss of system support. Also provided is a System Tape Restart function, which uses the data recorded by System Tape Checkpoint to start up a new MOC in the same state as existed at checkpoint time.

Although this basic approach has remained similar across programs, system software has been adapted to increasingly complex system environments. In the Gemini environment, for example, all system control programs were locally developed and customized. This made it relatively simple to suspend all MOC processing re-

quired for High-Speed Restart and System Tape Checkpoint. This task has become much more complex in the Shuttle environment, which features standard system software such as MVS, VTAM, and JES3.

In addition, the performance of High-Speed Restart and System Tape Checkpoint has been challenged to minimize the time required for data transfer despite the tremendous growth in the volume of data. This growth is illustrated in Table 1.

Although the software is optimized to transfer data in the shortest time possible, total MOC support interruption at the required frequency for System Tape Checkpoint is still undesirable for the Shuttle operation era. To address this problem, the basic system software functions are being augmented with an application-data checkpoint/restore. The application Flight Checkpoint function will be used frequently at user request, with minimal impact to MOC support, to capture and record all important MOC data related to the user-selected flight. The corresponding application Flight Restore function will be used in the recovery process, to restore, in a functioning MOC, the data previously recorded by Flight Checkpoint.

Unique features of the payload operations control computer (POCC) architecture

The POCC simultaneously supports multiple independent real-time users in monitoring and controlling experiments onboard the Shuttle Orbiter. It embodies a number of unique architectural characteristics which distinguish it from its RTCC predecessors.

The basic architecture provides the user independence, error protection, and guaranteed CPU and memory resources afforded by a distributed system while retaining the resource-sharing flexibility of a centralized system.

The fundamental elements of this architecture are the Executive Address Space (Exec); Inter-Address Space Communications through a Common Service Area (CSA); and User Address Spaces (UASs). The Exec receives multiple telemetry streams from a "front-end" computer system which interfaces with an external communications system, isolates parameters according to the user group which is authorized to process them, and distributes these parameters to one of the several UASs—each of which supports a given user group. A storage area (CSA) addressable by all UASs is used as the transfer medium and is protected from unauthorized or inadvertent "reads" and "writes" by an authorized SVC.

UASs for different user groups are dynamically created and terminated by the Exec according to the mission-support timeline. UASs operate independently and asynchronously, execute computations unique to a given user, and are protected from each other's potential malfunction by standard MVS error-recovery software. Computation results are placed in CSA where they will be available to all UASs for viewing on displays. UASs which send experiment-control information to onboard equipment do so through the Exec, which provides the interface to the external communications system. Most functions which are common across UASs and the Exec are supported by a single, reentrant copy of the required software which is loaded into write-protected CSA and is therefore executable by all address spaces as required. MVS-SE time-slice and rotate-priority functions are employed to ensure equitable CPU resource distribution across address spaces.

Table 1 Data transfer growth. (Note: M = 1 048 576.)

Project	*Transfer data volume* (M bytes)	*Data residence*
Gemini	0.3	Main memory
Apollo	5.0	Main memory, LCS
Shuttle	32.0	Real memory, disk

Display support software (DSS) executing in each address space uses MVS VTAM to communicate with a distributed display control system (DCS). The DCS consists of a minicomputer and several attached display terminals with minicomputer capability. DSS operates as a data-retrieval service only, leaving the functions of display composition (off-line definition), display formatting (arrangement of data on viewing surface), and units conversion to terminal-resident software. Given standard DSS/DCS communication conventions, DCS hardware or software changes remain transparent to the POCC 370/168.

An off-line POCC 370/168 program interfaces with each user group via DCS to accept a group's unique FORTRAN-defined computations and to integrate them into individual UASs for execution in real time. Each group's computation definitions are retained in separate libraries which are updated both before the mission and during flight support.

- *Computer hardware technology*

From Mercury through Apollo, some significant elements of hardware technology were driven by the unique requirements for real-time support of manned spaceflight. For Shuttle, the emphasis has shifted to take advantage of existing leading-edge hardware technology to satisfy program requirements. The required processor speed and memory capacity have increased by a factor of approximately 65 from Mercury to Shuttle. Additionally, aggre-

gate channel rates have increased by a factor of eight over this same period.

Early in the Gemini program, the need for fast auxiliary memory led to the development of the four-megabyte Large Core Storage that was used throughout Project Apollo. Beginning with a 96K-byte core file (where K = 1024), the auxiliary memory progressed through 256K and 512K versions before reaching 4096K bytes. Much of this activity provides a base knowledge for the design and development of today's virtual systems.

A significant contribution to evolution of today's byte- and block-multiplexer channels was the need to accommodate the continual growth of both the number and speed of digital data streams. The 7281 Data Communications Channel (DCC) used on Project Mercury was, for its time, highly innovative in its approach to the hardware/software design tradeoffs needed to handle I/O traffic and interrupt rates. The DCC function was provided by the 2902 Multiplexer Line Adapter through Project Apollo and by the 2909 Asynchronous Data Channel in today's Shuttle RTCC.

Another significant requirement was the need to configure the wide array of peripherals among as many as eight different processors. This led to the development of the 2914 control unit string switch, the 2844 DASD switch, and designs for channel-to-channel interconnection. The resulting speed and reliability have greatly influenced today's data processing system designs.

- *Software technology*

The emphasis on quality, performance, and reliability necessitated by manned spaceflight has provided the incentives to use the most current software engineering techniques and to develop sophisticated software tools. These incentives provided the basis for several standard software products, including tools for automated output spooling and products utilized in electric utility and oil refinery applications. Many currently accepted software management concepts also originated in the RTCC projects, and significantly influenced the evolution of software engineering techniques.

It was discovered in 1963, during the early days of Project Gemini, that one of the fundamental prerequisites to successful development of large software systems is early visibility and control of the developing systems. From a program code perspective, these were provided by creating a master system at the beginning of code development and putting it under strict configuration control. From a work planning and tracking perspective, this was accomplished by regularly publishing a development plan containing all the significant development and test milestones and current status relative to these milestones.

In the late 1960s, on the Apollo project, top-down system development was used by several subsystems. Beginning in 1970, the simulation systems required for flight controller training used top-down system development and structured programming across the total system. Since 1975, the Shuttle RTCC has used systematic estimating techniques, top-down development, structured programming, design inspections, and code inspections. The Shuttle Payload Operations Control Center is using all the latest FSD Software Engineering Practices [11].

A more systematic approach to software quality assurance has evolved with the development of the Shuttle RTCC. An independent software quality assurance organization was established to participate in all major project reviews and to provide periodic audits and spot checks of the development process and products. Working in conjunction with other software development mechanisms, such as project development plans, the software quality assurance activity provides increased visibility, control, and independent checks and balances throughout the development process.

In the early years, it became evident that the development of tools which are essential to implementation, testing, and even design itself must begin with the design of the operational system. For example, programs to generate test data, to measure performance, to log and delog data in real time, to take snapshots of code and data areas, and to execute programs without access to real-time interfaces are all typically designed along with the design of each new or modified system.

The size, complexity, and performance requirements of RTCC systems led to early development of sophisticated design and system-performance-monitoring software tools. Computer system models were used, beginning with Apollo, to evaluate systems performance for candidate designs. For Shuttle, parametric analysis techniques have been developed which are less costly and time-consuming than the earlier models and have proved to be very effective. System-performance-monitoring software tools have been used, beginning with Apollo, and have continuously been improved, culminating with the comprehensive Advanced Statistics Collector software used for Shuttle.

The criticality of the RTCC systems has resulted in development of testing approaches and techniques which have significantly influenced the industry. This includes

test tools, test planning and specifications, role of independent verification, and the relationship between development and independent verification.

- *Software development process*

The RTCC software development methodology has made significant advancements over the years. The catalysts for this continuing improvement have been the size and complexity of the systems, ever-increasing requirements, and the continuing need for improving the predictability, manageability, and cost-effectiveness of the software development. This progress has occurred over a long time period and has been evolutionary, not revolutionary, in nature.

A significant advancement over the years has been a progressively clearer view of the steps necessary for effective software development: define and control requirements; select/document implementation approach; develop/control system design; estimate cost, schedule, and development resources; establish detailed development and test plans; and continually assess progress *vs* plans. A brief discussion of these factors with illustrations of how they have evolved follows.

Define and control requirements

The evolution and growth of the space program from project to project has been paralleled by a corresponding change in the software requirements and the requirements generation process. The trend has been characterized by a growing level of detail in the requirements, an earlier documentation schedule, more uniform and consistent control procedures, and more effective change control.

During Mercury and Gemini, there were essentially two levels of requirements which were provided by NASA to IBM: an initial set which provided an identification of the functions the system was to perform, and a final set which was used primarily to describe the outputs required by the flight controllers. The details of the requirements were negotiated between the NASA technical interface and IBM lead programmers. The initial requirements were provided by NASA fairly early in the development process along with a transmittal form directing IBM to implement them. The final requirements were agreed upon but not completely documented until the program implementation was near completion. This provided a great deal of latitude in making program changes, since the requirements and change control were effectively in the hands of the NASA technical interface and the lead programmer.

The increased size and complexity of the Apollo project demanded additional formalization of the requirements definition and control process. To facilitate this formalization, another NASA requirements delivery termed "Basic Requirements" was included between initial and final. A formal change procedure was also established. "Basic Requirements" was, in fact, a NASA-published draft of the final requirements for use in establishing the detailed requirements. Final requirements still were not transmitted by NASA until late in the implementation cycle. This earlier documentation of requirements and procedures for approval of changes put more of the monitoring and control of requirements in the hands of NASA and IBM management. During Apollo, the major emphasis was the technical aspect of a lunar landing, and very few requirements were rejected on other than technical grounds. Skylab's less flexible budget made the decision to make a requirement mandatory dependent on its cost as well as technical considerations. A new procedure was developed to allow an evaluation of potential requirements changes from a cost and schedule impact standpoint prior to final approval.

The Shuttle program re-evaluated the requirements process and established a system with four levels of requirements intended to parallel software development. Level A requirements are generic and identify the major system functions. Level B requirements provide a complete description of the capabilities of each function and are sufficient to develop subsystem design. Level C requirements provide the final detail and are sufficient for completing program implementation. Level D requirements describe certain reconfigurable items such as data formats which are updated beyond Level C cutoff. These requirement levels are scheduled and formally delivered by NASA to IBM during the appropriate program development phase. IBM has a documented, formal change assessment and control process which provides increased requirements visibility and change control to the programmers, line management, and project management.

Select and document the development approach

The success of any system development depends to a large extent on how effective the planning is in the initial stages. Not only must plans be established, but they must be communicated and understood by the people who have performance responsibility. For a project to run efficiently, the management approach as well as the technical aspects of the work must be understood and agreed upon.

As a result of these considerations, IBM published a comprehensive management plan during the system definition phase of the Shuttle RTCC project. This plan describes the cohesive management and technical approaches which are used in system design, development, and test; it serves as a project standard to be followed by all elements of the organization. The management plan

serves not only to educate people on how the project is run, but also as a guide for development of more specific plans and project procedures. The management plan addresses the following major topics: organization responsibility and authority, requirements management plan, project phases and milestones, configuration management, schedule and resource management, computer systems analysis plan, documentation plan, and standard techniques and tools.

Develop/control system design

Design for both large and small systems is a process requiring creativity, knowledge, and discipline. Similarities in the design process end rapidly, however, as the design becomes the framework in which hundreds of programmers will produce hundreds of thousands or even millions of lines of code. For a large system, design quickly exceeds the span of control of individual designers, and the need to support a cohesive design process becomes of great importance. This process must bring out the best creativity in individuals while maintaining discipline.

The IBM RTCC approach has been to use an organizational structure in which there are no separate "design" or "architecture" departments. Lead technical people are expected to be designers and implementors. This in no way de-emphasizes the design function, but rather emphasizes the design process and not the designer as an individual. A major influence on the RTCC systems achieving good design, with structural integrity, efficiency, and other desirable attributes, has been moving the design process out into the open. For design of the Shuttle RTCC, a systems architecture group was established. This group is not an organizational entity, but consists of senior technical representatives from each major RTCC subsystem.

Planned customer reviews and internal audits review and validate the design at key points in the development process. Customer reviews are formal, scheduled, and rigorous. Two major reviews are conducted during the design of each subsystem; these are the Preliminary Design Review at completion of the functional design, and the Critical Design Review, which serves as a final checkpoint before the majority of the design is committed to code. These reviews usually last for several days and are preceded by IBM internal reviews which in themselves are effective architectural controls. Design reviews are attended by NASA customers, consultants, and other contractors. Questions raised in these reviews lead to documented action items which require formal responses to the questioner and to a smaller design control board established for this purpose. Once this formal review process is completed, the design is placed under strict change control.

Overall system architecture, logic and data flow, and interfaces between programs are documented in plain English in the functional design. Module design is expressed in a Program Design Language.

Estimate/budget cost, schedule, and development resources

Accurate resource estimating has been a continuing challenge for the software developer. Many factors contribute to this, not the least of which has been the lack of a systematic approach to estimating and the absence of valid reference points. For many years, resource estimating depended almost solely on the experience of the individual programmer. Estimates were developed from the bottom up in a rather *ad hoc* fashion.

Although resource estimation is still not an exact science, significant improvements have been made. In the development of the RTCC system for the early Shuttle orbital flights, resource predictions have been very close to target. This was accomplished while a system of one million lines of code was being developed over a three-year period.

The first step in any estimating process is to understand the job. This involves two major elements: understanding the environment, and understanding the requirements. Key elements of the environment include stability of computer hardware and operating system, newness of design, level of experience, and status of development tools. Understanding the requirements involves identifying each software product and the associated cost components, as well as laying out the key milestones. The second step involves the establishment of a clear set of ground rules for obtaining customer agreement. The ground rules define the assumptions and criteria to be used in the estimating process as well as in the establishment of an estimating model. The model is developed using available history data and experience. Throughout the development process, historical data (*e.g.*, lines of code produced, computer hours used) are maintained. Periodically, the historical data are analyzed and the estimation model is recalibrated. By obtaining customer concurrence on the ground rules and applying them consistently across the project, many of the problems associated with estimating and negotiating resources are eliminated.

Establish detailed development and test plans

A critical factor in the RTCC software development process has been the use of formal Project Development Plans (PDPs) and Test Plans. These plans document details of the project's top-down development process, describing the development philosophy and specific activities of the development and test processes.

Prior to Space Shuttle, the RTCC Project Development Plan depicted schedules in terms of a prose, tabular format; the Shuttle Development Plan uses both tables and Program Evaluation Review Technique (PERT) charts to describe development and test schedules. The PDP is used to track all of the RTCC project activities from initial design studies through final system delivery. Key milestones described in the plan include requirements dates; system design reviews; preliminary and critical design reviews for each subsystem; initial, basic, and final system releases for each subsystem; and final system delivery. These milestones and accompanying subsystem/system-level PERTs are carried in the summary volume of the PDP and are used for reviews with upper-level NASA management. Targets for detailed development activities—design inspections, code inspections, development test specifications, development test completion dates, periodic system builds, Independent Verification (IV) test completion dates—are described in additional volumes of the PDP and are used by first-line IBM and NASA management to track development of each module of the subsystems.

A primary activity in establishing the schedules in the PDP is the resolution of interdependencies. These include the dependency of the software developer on NASA requirements, the dependency of software testing on hardware availability, and the dependency of software development on other software elements. This has proven to be the most time-consuming part of the scheduling process, as it usually requires several iterations and involves a number of parties including IBM, NASA, and other contractors. The results of these negotiations are clearly documented as dependencies on the summary PERTs and in a special detailed interdependency volume of the PDP.

Another element which has proven to be a primary factor in development planning is projecting the performance of the real-time mission software. For Gemini and Apollo, these projections were made using detailed digital computer simulation models. These models have been replaced for Space Shuttle by a parametric analysis technique [12]. This concept reduces requirements and design to a set of key parameters (*e.g.*, average number of displays updated per second) to be monitored throughout the development process. As design is finalized, initial projections are updated and used by IBM and NASA management to make key requirement and performance tradeoffs. The key parameters are listed in the PDP, and regular updates and presentations to NASA are scheduled at significant milestones in the development process.

Software developed for the RTCC prior to the Shuttle followed the traditional test hierarchy of unit, subsystem, and system testing. This hierarchy was followed from the earliest days of RTCC software development through the ASTP era. Beginning with the Space Shuttle, RTCC software is being developed using structured programming with two levels of testing—Development Testing and Independent Verification (IV) Testing. The Shuttle Test Plan was originally generated during the system design phase and has been updated periodically at key project and requirements milestones. The Test Plan defines guidelines for Development Testing and IV Testing.

Development Testing encompasses all testing performed during the development phase. Beginning with the testing of the application control programs, the development testing follows an orderly process of requirements-oriented testing of each function both before and after it is incorporated into the master system. This testing continues until all elements of the software are tested together, at which time it is delivered to the IV group as the Final System Release (FSR).

IV testing is done by an organization independent of the development organization, using test specifications and system test environments unique to IV. This test activity begins following receipt of the FSR system from the development organization. Initial testing verifies that the software is capable of supporting all required mission configurations. This is followed by interface tests to verify internal and external interfaces, error recovery, and restart and selectover. As interface testing concludes, emphasis is shifted to performance measurements which will measure memory, CPU, and I/O utilization and be used to validate the performance projections in the PDP. Following the successful completion of IV testing, the system is delivered to NASA for flight controller training and mission support.

Continually assess progress vs plans

The Project Development Plan is a "working" document used to continually assess the progress of software development. All parts of the PDP, from the summary sections through the detailed sections, are monitored on a daily or weekly basis by IBM managers and programmers and their NASA counterparts. Formal meetings are held weekly with NASA to review the status and problems noted in the PDP. This continual monitoring of status yields the significant benefit of identifying problems early, thereby allowing adjustment of dependent activities or schedules while the greatest number of options are still available. The PDP is republished every two weeks to reflect updated status and actions taken.

Formal development plans have been used for all RTCC projects. The content of the plans has been up-

graded to reflect advances in programming technology and sophistication of both the IBM team and NASA personnel. In most respects, the current PDP includes more detail than those of the earlier RTCC era. This is primarily a result of a greater understanding of the criticality of timely status monitoring and the need to clearly communicate the detailed interfaces and interdependencies of today's more complex systems.

Summary

Throughout the U.S. manned spaceflight programs, IBM-developed RTCC systems have been at the forefront of technology. The fundamental driver in the evolution of these RTCCs has been the requirement to perform real-time command and control for increasingly complex missions. RTCC improvements have resulted from these changing requirements, advances in hardware and software technology, the need for improving productivity, and the maturing experience of the technical and management team.

One notable aspect of the RTCC evolution has been the continuing movement from highly specialized systems, containing much unique hardware and system software, to a more generalized computing complex with emphasis on maximum use of standard hardware and system software.

Through the years, major emphasis has been placed on improving software development by establishing a process that ensures software visibility and control throughout the development cycle. The objective is to ensure consistent, predictable achievement of cost, schedule, and quality commitments. IBM has maintained an excellent record of achievement in this regard.

Acknowledgments

This nation's space program would not have been possible without the combined energies of the many thousands of members of the NASA/Industry team over the last two decades. On a much smaller scale, this paper would not have been possible without the combined support and encouragement of a number of people at IBM Houston. In particular, the author appreciates the contributions from E. L. Campbell, G. H. Evans, W. S. Harner, H. Hulen, W. J. Kloster, G. E. Morris, H. L. Norman, R. G. Olin, F. M. Riddle, W. D. Sigler, C. K. Waund, and L. S. Wright.

References

1. J. M. Grimwood, *Project Mercury, a Chronology,* National Aeronautics and Space Administration, Washington, DC, 1963, p. 38.
2. B. C. Hacker and J. M. Grimwood, *On the Shoulders of Titan, a History of Project Gemini*, National Aeronautics and Space Administration, Washington, DC, 1977, p. v.
3. W. von Braun and F. I. Ordway, *History of Rocketry and Space Travel*, 3rd Ed., Thomas Y. Crowell Co., New York, 1975, p. 237.
4. W. G. Holder and W. D. Siuru, Jr., *Skylab, Pioneer Space Station*, Rand McNally and Co., 1974, p. 84.
5. E. C. Ezell and L. N. Ezell, *The Partnership, a History of the Apollo-Soyuz Test Project*, National Aeronautics and Space Administration, Washington, DC, 1978, p. vii.
6. C. M. Green and M. Lomask, *Vanguard, a History*, National Aeronautics and Space Administration, Washington, DC, 1970, p. 160.
7. "Final Report, Project Mercury," prepared for National Aeronautics and Space Administration, March 1, 1962.
8. "Project Gemini Final Report," submitted to National Aeronautics and Space Administration, Manned Spacecraft Center, Houston, TX 77058, March 5, 1968, IBM Federal Systems Division, Houston, TX.
9. "Apollo Lunar Landing Report, Interim," submitted to National Aeronautics and Space Administration, Manned Spacecraft Center, Houston, TX 77058, November 25, 1970, IBM Federal Systems Division, Houston, TX, p. 5.3.
10. "Space Transportation System Mission Control Center System Specifications for the Shuttle Ops Time Frame," National Aeronautics and Space Administration, Houston, TX, April 1979.
11. H. D. Mills, D. O'Neill, R. C. Linger, M. Dyer, and R. E. Quinnan, "The Management of Software Engineering," Parts I-V, *IBM Syst. J.* **19,** 414–477 (1980).
12. J. M. Mohon, "Performance Projections During Design Using a Parametric Analysis Tool," *Proceedings of the IBM Design '79 Symposium*, Santa Teresa, CA. April 1979, p. 379.

Received July 7, 1980; revised October 21, 1980

The author is located at the IBM Federal Systems Division facility at 1322 Space Park Drive, Houston, Texas 77058.