NASA JOHNSON SPACE CENTER ORAL HISTORY PROJECT
ORAL HISTORY TRANSCRIPT

JOHN R. GARMAN
INTERVIEWED BY KEVIN M. RUSNAK
HOUSTON, TEXAS – 27 MARCH 2001

RUSNAK: Today is March 27, 2001. This interview with Jack Garman is being conducted in the offices of the Signal Corporation in Houston, Texas, for the Johnson Space Center Oral History Project. The interviewer is Kevin Rusnak, assisted by Carol Butler.

I'd like to thank you for taking the time out to speak with us this morning.

GARMAN: You're most welcome.

RUSNAK: And if we can start out, tell us about your interests and experiences growing up that led you on to the science and engineering path that eventually took you into the space program.

GARMAN: Well, I think I'm a Sputnik kid. You know, I was just entering high school when all that started, and finishing high school when the Mercury Program put a couple of very famous people into space. As I went into college, I guess my parents had always encouraged me to be an engineer. It's a wonderful thing to do, but it's kind of masochistic if you're a young person, to go through all that education. But in those days the pay was quite good, too. So I went into engineering school at the University of Michigan [Ann Arbor, Michigan].

It was somewhere during the college years when the Gemini Program was kicking off. I was in college from like '62 to '66, and somewhere in there a lot of news articles were popping out, of course, about the space program, and I was very much attracted to that. I remember that in that last year of school, when you start interviewing for jobs, about three out of four interviews I went to had something to do with the space program.

The fellow at the time, John [P.] Mayer, who headed the Mission Planning Analysis Division [MPAD], was a University of Michigan graduate, so every year he insisted on sending someone up to that school to do interviewing. Then and now, I think, NASA couldn't afford—the government couldn't afford or didn't pay for trips for interviews very often, but they didn't really need to. They didn't really need to. So I remember interviewing, and it was the lowest paying job, I remember, and I had no idea where Houston, Texas, was. When you're raised in Chicago and go to school in Michigan, you know, everything in Texas is somewhere south of Dallas. I got in the car and drove down.

Michigan, in those days, they didn't offer any undergraduate degrees in computers, none. So I went into an engineering curriculum at Michigan called physics, engineering physics. I used to laugh that I'm a physicist of some kind. I'm not. But that curriculum allowed a minor, a specialty, if you will. In those days, and I think it's still true, engineering degrees are usually rock solid. There's really little choice in what you take. They're usually five-year, and I wanted to get through in four, so I crammed my way through.

But the notion of being able to take a specialty in computers was neat, so I did that. So during my junior and senior year, when you finally take the courses that you can focus on in college, I ended up in grad school classes with all the double-E, you know, electrical engineering and those types of folks, who finally got a chance to take something in computers as grad students, and I was in there as an undergrad. I used to laugh at that. As I say, I'm no physicist, but that was the degree I got.

RUSNAK: What kind of computer technology are you learning on at this point?

GARMAN: With an engineering school, it's kind of from one end to the other. We were taught programming, of course, a Fortran style. They had a language called MAD, Michigan Algorithmic Decoder. It was an offshoot of Fortran that we all used, and that was wonderful.

Today, of course, folks sit in front of a keyboard with a CRT [cathode ray tube] screen and type their programs in if they're programming, and in those days we also sat in front of keyboards, but they were called keypunches, stacks of cards, and we'd type away and out they'd come. Walk up to that mysterious room where all the folks, in those days and maybe these days, too, with the very long hair and beady eyes sitting behind the counter running the mainframes to collect our decks of cards and spit us back our reams of paper the next day. That was great fun.

In engineering school, we also got into the bits and bytes. So I took a lot of lab courses where we were putting stuff together, you know, transistors, flip-flops, the whole nine yards. I knew I never wanted to build computers, but it's kind of like if you're a good driver of a car, you know what's under the hood even if you're never going to repair it yourself. If a mechanic is going to do something to it, you can be a smart customer, you know exactly what they're talking about. I think it's the same thing in computers.

That experience is clearly what got me into operating systems. I didn't know it was called that in those days. They didn't have a name for it. But the software that resides closest to the hardware and interfaces with the applications is called operating systems [OS], and it's kind of the glue that holds things together. If you want to do something, it's kind of fun to be in the middle, so that's where I was. That was what attracted me.

When I came down to NASA after that, they didn't have anybody that knew anything about computers. It was really funny. I was twenty-one when I walked in the door, for a few more months. There was an interesting side story. I can't remember the fellow's name. Maybe it'll come to me. In those days, Ellington Field still had the old white buildings with green roofs, wooden barracks kind of buildings from World War II, and the then Manned Spacecraft Center used a lot of those buildings as overflow. So all new employees—and there were a lot of new employees coming in 1966—all the new employees were sort of mustered in in that facility, in those facilities.

I remember, and you're kind of excited, you know. You're young. You don't know where the heck you are. It took me six months to figure out I was as close to Galveston as I was to Houston, by the way. That's kind of embarrassing, but it's true. You know, it was kind of "Alice Through the Looking Glass." You get in the car, you drive down, and you start going to work. Your eyes are wide, because I walked in, and they were finishing the Gemini Program. It was still flying. So everything was very awe-inspiring.

But I'm sitting at this long table with a bunch of other new recruits. I say that because it was very military-like in those days. I'm sitting there with a bunch of other new recruits, and there's this face across the table that I knew I recognized from somewhere. Well, you know, we introduced ourselves to each other, and "Where'd you go to school?"

"I went to Michigan."

"So did I."

"Oh. What degree program did you go after?"

I said, "Engineering physics."

He said, "So did I."

It turns out we'd been through school all together and never known each other. We'd passed back and forth in classes, and this young man was recruited the same way I was.

When we got there, the job we'd been hired for didn't exist. They'd interviewed us to work in MPAD, Mission Planning and Analysis Division, on a hybrid computer. Now, computers weren't real fast in those days. So the notion of analog computers still existed. They still used them. Capacitors and other electronic devices behave mathematically quite well and you can run currents through them to, in an analog way, do mathematical equations. So you'd sort of create a hybrid computer, part digital and part analog, to do a lot of the kind of work that NASA wanted to do. So they'd decided to buy a huge hybrid computer, and the money fell through. Bob [Robert C.] Arndt was his name. Bob and I were both hired for this job.

One of my mentors, a fellow named Lynn Dunseith, who has since passed away, Lynnwood C. Dunseith—Lynn was head of the branch. Directorate, division, branch, section was the structure in those days. He was head of the branch, and the branch had two sections. One was for onboard computers, Apollo, and one was for the big ground computers, mission control.

Bob and I went wandering in, and he sat us down and told us, "You don't really have a job. Welcome to Houston." We're wondering where the heck this was leading. He says, "Fear not. You're hired. In fact, what we have to do is figure out where to put you now."

So we had the great pleasure of interviewing throughout that division, on the fly, to decide where we wanted to work. Lynn told us both, he said, "Well, your decision is going to be real easy." He said, "You're either going to work on onboard computers or ground computers. From there on, who knows? You're new, you're young, we'll put you where you can dive in where you please."

Well, I chose onboard computers. That excited me, you know, the computers that fly. Bob chose the ground computers, and we hardly ever saw each other again. It was kind of funny. I have no idea where he is now, but we were kind of like ships passing in the night at school. We'd rendezvous when we'd come in, and then we go our separate ways and hardly ever see each other again.

Diving into onboard computers, the section was called the Apollo Guidance Program Section, and I can remember many of the names, not all of them, but a fellow named Tom [Thomas F.] Gibson [Jr.] was head of the section. There were a couple of relatively—to me at twenty-one years old—senior people floating around, generally folks that had just come out of the military, by the way, and there were like thirty of us in a section. Today there are directorates that are that size. There are certainly divisions that size. So this was a huge section. In fact, within a couple of years, there were several reorganizations that sort of put things back in perspective.

As I recall, what had happened was, it was before I was there, I came in May of ‘66, and I think back in February of that year the software program for the Apollo onboard computers had gotten in deep trouble. As you might expect, they had the entire computer operation in what was then the Engineering Directorate, still under Max [Maxime A.] Faget, I think. It wasn’t working. Most of the software expertise was with the folks that were doing mission control and the ground systems, and that was all in MPAD in those days. So they decided to lift the software portion out and move it over to the Mission Planning and Analysis Division in this—I think it was called Flight Operations then, but MPAD was part of a large group that included all the flight controllers and mission planning, in other words, all the operations as opposed to the engineering. They had the very military style organization of engineering in one group and operations in another.

So all these folks—well, many of these folks—were transplants. They’d just been reorganized. I didn’t know that. You know, you’re young, dumb, you don’t know. They all had new bosses and they were all figuring out how to fit, and we had this big influx of, I don’t know, maybe a dozen of us that came in in that ‘65, ‘66, ‘67 period. So there were too many people. There were lots of folks, and therefore lots of time. Go send us to school. Go send us to the contractor.

Well, they did two things for me which were really, in retrospect, wonderful. I’m not sure they knew what they were doing, but they did it. The onboard computers for Apollo were designed and programmed by what was then called the Instrumentation Lab of MIT [Massachusetts Institute of Technology, Cambridge, Massachusetts]. It’s now called the Charles Stark Draper Lab. But they didn’t do the building. A subsidiary of GM [General Motors], I think, Delco actually built it. They provided hardware training and some software training classes.

The notion of making sure everybody understood all the innards of everything being built in Apollo was very important in the space program, and so they poured a lot of money into training programs. So they sent me off to a school. I think it was here. I don't think I had to travel very far. It was in Houston, I mean. But very soon after I came on board, I was gone. I was in classes for four or five weeks. But you've got to picture, here's a young kid coming out of school who's just left a long series of hardware kind of classes and really into computers at this point. So while I was drinking from the fire hose, I had no trouble swallowing. I mean, I just picked up everything that they were trying to dump on me.

So when I got back to the office after that, I truly didn't realize it at the time, but I was an expert. I mean, I knew more than most of the other folks. I don't mean generically, but I meant if one of the people was an expert in one area, I knew just about as much as that person did in that area, very much a generalist as a result in those onboard computers. So for that reason, they immediately made me a group leader. Somebody divided the organization into groups and made me a group leader for something or other.

But I started traveling, then, up to Boston to visit the Instrumentation Lab, and, boy, they'd send us up, I don't know, every other week we'd go up. Looking back, it's really kind of funny, because these folks—you know, I'm twenty-one, twenty-two maybe by this time, clearly very wet behind the ears, and you walk into these offices where these folks that are in their thirties, which were quite old to me in those days, are working like crazy, and here comes this young thing with a NASA badge, on saying, "I'm in charge, and I want to know exactly what you're doing and why," because that's what we'd been told to do.

"Sure. What would you like me to tell you?"

I didn't have the slightest idea what to ask, so they'd tell us whatever they wanted. Some of these Instrumentation Lab folks were really good. They were more like—well, it being part of the school, they were more like professors. They wanted to teach and help you learn what they were doing and why. Others viewed it as interference, you know, "Get out of my way. I'm trying to do the job."

I think my interest in operating systems and that part of the business that they pointed me into made it easier, because this is the kind of work that these folks had created basically from scratch, so they loved to talk about it. You know, it's an invention almost. So that was training, too. I learned a lot about Apollo there.

They put me in charge of the AS-501 [core] ropes. We had a phrase that we called "rope mothers" back in those days, not meaning to be sexist at all. You know, parenting the development of the software. Why ropes? Well, the onboard computers in Apollo had hard-wired memory, not in today's chip sense at all. It was all core memory, but the bulk of the onboard computer memory was woven. If you take a magnetic core and you magnetize it one way, it keeps it, and if you magnetize it the other way, it keeps it. That's your one or zero and a bit. One core per bit, that's a lot of magnetic cores and a lot of power, and it's very expensive in a spacecraft.

So they came up with a way of creating one core that would have many bits. What they'd do is reverse the invert the way they did the ones and zeros. They'd run a stream of wires, and on a sixteen-bit word, they'd run sixteen wires through a core. Well, if it was a one, they ran it through the core; if it was a zero, they ran it around the core. But if you can picture a bundle of sixteen wires several miles long and series of cores, millions of them, with these wires going through or around and then one extra wire that would be used to flip the magnetic polarity of the core, what would happen is, if they wanted to know the value of a particular word, they'd find out what core, what entire core, represented that word, flip it, and all the wires that went through got a pulse and all the wires that went around got no pulse, and that's your ones and zeros all at once. So you get about sixteen times the compression.

The problem is—and literally, they looked like ropes. I mean, they'd bundle them up and put them in a box so they looked good, but as I recall, the myth, probably the fact, is that Raytheon did this, built these core memories, these ropes, and they did it in the New England area because that's where a lot of weaving took place, a lot of fabric industry, and they literally used—I mean, what they told us was that these were all little old women that would weave the ropes. I trust it was more mechanized than that. I actually remember going to see it once, and I don't remember—the myth got ahead of the fact, I'm sure.

But the striking part of it was that when it came time for a software development cutoff, it was real physical. I mean, once they started weaving, it was very, very difficult to change anything, as you can imagine. I remember once or twice they actually tried to do it, but if they did it wrong, they'd have to start over, basically, and this was very, very expensive.

Software development was not very mature in the mid-sixties. There was not a lot going on. The notion of modules, subroutines, and any form of abstraction, which is the way you think about software engineering today, either didn't exist or they were very far-out concepts. All programming was basically done at the ones and zeros level, assembly language programming. Assembly language was about as modern as you got, you know, the notion of having another computer that would translate semi-English words into the ones and zeros that another computer would understand.

The Instrumentation Lab folks had some mainframes that did this, and the computer programs for the onboard software were all in assembly language and all very well annotated. That is, they put comments into the listings. But that was about as far as it went on documentation.

There was very, very little in the form of requirements, and testing was a new concept. This was where I think my colleagues who had military experience in development sort of met the academics who had the push the state-of-the-art or practice experience in almost a collision, and folks like me were caught in the middle. Testing is the name of the game when you're playing with mission safety, critical kind of systems. So the idea of having to test and retest and every time you make a change test again was foreign, understood conceptually, but it was a lot of busy work, a lot of overhead from the Instrumentation Lab folks' standpoint to go through and actually document, write down all the tests which you're going to run, write the test results, review them, and if you made another change. "And do tests incrementally? What are you talking about? Let's build it and test it."

So the idea was to go ahead and test—the idea that was created was to do—I think we called them levels of testing. We did. My god, I've forgotten all that. Level 1 through Level 6 testing. You had to carry it all the way into the Shuttle Program, as a matter of fact. Very much, as I recall, on a military model, although I'm not sure the military was that far ahead of NASA at that point.

But I remember that as—and again, I'm very young, I don't know a lot of people, I'm just doing my job. When you're in your mid-fifties, as I am today, looking back there, you feel like you can talk with some authority, but I didn't know what was going on. I'm just a kid working back in those days. But I remember the conflict. I remember the arguments. And when you're in the middle of it, you're not quite sure which side to take, you know. It gets to be one of those "I can see the reasons for both sides. It's so slow to have to test. Let's get on with building it."

Anyway, that was clearly one of the contributions of my colleagues. Stan [Stanley P.] Mann was one of the key ones, I remember, M-A-N-N. He was a [US] Naval Academy [Annapolis, Maryland] graduate, as I recall. I think he's still in the area here. Anyway, Stan and the other folks really pushed hard on the levels of testing and won over, and it ended up, I think, being one of the instrumental things that made onboard computing work for the Apollo Program.

The other real mentor in my life was around in those days, a guy named Bill [Howard W.] Tindall [Jr.], who, unfortunately, also has passed away at this time. Bill was one of these never really wanted to be a manager, just wanted to get the job done, one of the real heroes of the Apollo Program, actually. He was a technical assistant or something under John Mayer in those days in the Mission Planning and Analysis Division.

Bill had an activity that was called data priority. I still don't remember why it was called data priority, but the notion was—I guess because Mission Planning and Analysis Division dealt with data: how do you figure out what the right way to handle and move data around is? He ended up getting in the middle because of the problems of the onboard software development.

It was somewhere in this time frame that we reorganized, and Lynn Dunseith's branch with two sections became a division with a couple of branches. I think it was called Flight Support Division, maybe. And come to think of it, he wasn't even division chief; he was like deputy. They had a military fellow named Clements, maybe, Henry [E. "Pete"] Clements, I think maybe was the division chief. Again, I was pretty low. I didn't pay much attention to those details. I knew I worked for Lynn Dunseith.

What was funny was they kept Bill Tindall over in MPAD, and he was basically the guy in charge of onboard software development, and we were in another division at this point. As long as—I'm losing it on the reorgs. Soon thereafter, there was another one, and Jim [James C.] Stokes [Jr.], who had been heading this other section for ground computers mission control, ended up heading the division, and we were the only folks in his division that weren't working for mission control. That is, his entire division was working on mission control software and operations and so on in support of all the flight controllers, and we were this one section. I think we were still a section, yes, somewhere in a branch in that division working on onboard software. So we had sort of two bosses and weren't really sure who was in charge sometimes.

But I remember clearly, in looking back, there was no question who was in charge of the onboard software. Certainly, as we started approaching the Apollo 11 time frame, it was Bill Tindall, and I remember long, long, long meetings up at Instrumentation Lab going through the software.

Digressing back just for a second, this "rope mother" business, what they did was that the flights were fairly close together. They were planned to be close together. So when that happens, you have the overlap effect. That is, you're working on several flights at once, because the time to prepare, particularly if you're going to weave ropes, is longer than the time between flights. And so to keep track of the parallelism—you're working in various stages on many flights at once—they would assign someone to be the "rope mother," the person that was accountable for making sure everything got done right on each flight.

What was interesting was that this AS-501, or SA-501, as they called it, if you were a Marshall [Space Flight Center, Huntsville, Alabama]—have you heard that story? Apollo-Saturn. You know, the Apollo Program, [Wernher] Von Braun was very, very strong, of course, in those days, and it was the Saturn that was the Apollo Program, of course. Well, no, it wasn't; it was the spacecraft that was the Apollo Program. Well, let's fight about it. Well, let's call it both; let's call it the Apollo-Saturn flights. So if you were in Houston, it was AS, followed by a number, was the flight number, and if you were in Huntsville, it was SA.

That kind of rivalry has carried on for years. It's good rivalry. I think NASA was put together purposefully that way. If you create some competition, generally progress goes faster. It's created some big silos that maybe we'll come back and talk about later that, they really should come down over time, but in those days it was great competition, and there wasn't a lot of overlap. I mean, if you were working on the Saturn Program, it was different than working on the spacecraft.

Anyway, I was the "rope mother" for this Apollo-Saturn 501, which was an unmanned flight. It was the first launching of that huge, thirty-six-story launch vehicle. That's what got me in the control center. If you think the folks in charge of the software didn't have a lot of computer and software experience, you can imagine what it was like for the flight controllers. They didn't. Moreover, the onboard computers in Apollo were guidance computers. Their function was not computing as you think of it today, even in an aircraft. I mean, there's guidance and navigation in those computers, but no flight planning, nothing like that. Basically what those computers were doing was flying the vehicle. That's what it was for.

So, people who'd grown up in flight systems of any kind were used to analog flight systems. So this was still a flight system to them, not a computer system, and there were computers in it, but the notion that the world was changing had not gotten through to any of us. I mean, you can't predict the future easily, but the notion that this was a computer surrounded by wire and plumbing had not really gotten through. It was wires and plumbing with a few computers in it, okay? The spin was the other way. So computers were a pain in the neck. They would not work. There would be software bugs. "What is all this stuff? I just want it to work." The real problems are in propulsion or in—you know what I mean by plumbing. I mean pressure vessels and stuff like that. That's where the real macho kind of space biz is, and, "All this computer stuff, get it out of my hair."

So in this flight control world, flight controllers are sort of a different breed in NASA. They almost all had some sort of piloting experience, meant to be that way. They had to pull in some folks that were much more academic in those days, people that knew something about trajectory management and guidance and so on. In fact, they buried all those types down in the front row in mission control. They called it "The Trench," and they buried them down there, almost literally, the flight dynamics officer, FIDO, the guidance officer, GUIDO.

Anyway, they didn't have a lot of folks who knew much about computers, so they said, "We want some support." In those days we were all in the same directorate, so I and a couple of other folks got loaned to go sit in what was called the staff support room, or SSR, and support them. You got a picture. Gene [Eugene F.] Kranz was very active in those days as one of the flight directors. Chris [Christopher C.] Kraft [Jr.] was head of it all. It was very much a do it by the numbers. Everybody knew it would be a lot of crisis management.

So as I look back, they wanted people that were fairly young in there because they were obedient. Experience didn't mean a lot, because there was no experience in that business. A lot of the day-to-day work was pretty boring because you were following the book, but these folks wrote their own books. That's what was fun. They'd write their own procedures and rules and then follow them, okay? It was kind of like writing a play and then having to go through rehearsals to act in it, but you wrote the play so it was fun, okay? More fun for them.

But we don't understand all this stuff, so we got these folks to come in, sit in the support room to support us. They don't salute as well. [Laughter] I and colleagues like me from MPAD, or ex-MPADers, as it were, were all, even then, we knew we weren't flight controllers, so we were kind of rogues in that environment. But we knew our stuff. We knew our stuff very well. So I ended up spending a lot of time sitting on a console and watching the onboard computers.

This is a good point to interject a technology story. In those days, the icon for a computer was a tape drive. You all, I'm sure, are maybe too young to remember that, but anytime somebody had a caricature of a computer, it always had a box with two eyes, you know, the two tapes that would spin. Today, an icon is a CRT, the screen, but in those days it was tape drive. You see the old movies, you first see the old movies, what did you see in a computer? You saw a row of tape drives, and then you'd see the console with all the lights blinking, which were usually meaningless.

So when you walked into mission control, that's what you saw down on the first floor, was all these big IBM mainframes with the spinning tape drives and the lights blinking and all that. But when you went upstairs into the control rooms, they had CRTs, televisions. You could actually see this stuff. That's absolutely—it doesn't mean anything to anybody today. That's how computers work today, right? But in those days, if you spent your life in front of a keyboard typing punch cards and when the computer ran, you got it back on paper, to be able to see things happening on the screen in real time was absolutely awesome, particularly if you knew anything about computers.

Well, for someone like me, it was doubly awesome, because I wasn't watching what was going on. I wasn't watching the computer screen to look at trajectory data or plumbing pressures or systems. I was looking at the computer to see inside another computer. It's kind of like having an X-ray machine. It's kind of like going through medical school, to be trained on the human body, but before the time of X-rays and then suddenly being able to see, live, you know, with an X-ray or something, what's going on and saying, "Oh, yeah. I learned all that. That's how it works." We could actually see inside the onboard computers for Apollo and we got to help tell them what kind of displays to write for us.

Well, looking back, the awesome wasn't very deep. Those computer displays that we saw, the mainframe that ran the whole of mission control was only one—they actually had two. The had a primary and a backup, and they ran in tandem so one field, they could swap over to the other. The MOC they called it, the Missions Operations Computer. That computer's pretty slow. Mainframes, even back in those days, could do a lot of things at once and were very, very capable of pumping a lot of data in and out, even as much as any desktop computer can do today, much more, but in raw compute power, they're very weak. A big mainframe was not much more powerful than today's desktop computers, certainly didn't have the memory of today's computers.

So if you can imagine the programming they had to do to drive hundreds of displays and do all the calculations at once, first of all, it ran pretty slow, and, second, computing resource was hard to come by. The way they got more throughput in those days was to offload things. I mean, the computer was to simply output numbers, not try to paint pictures, and they'd do that with other devices that the computers talked to. So like the world map that's in the front of the control center, in those days the world map was a painted slide of a world map, and the trajectories were basically an Etch-a-Sketch. Have you ever seen those?

RUSNAK: Yes.

GARMAN: Okay. That's how it worked. The computer would literally sort of etch, drive a pen. They'd etch on something that projected the trajectory. They looked like sine waves on a locator projection. So the computer didn't have to do very much, just plot the line and it was done. Same thing on the displays. The displays we looked at were just columns of numbers, and all of the names—you know, if you'd want to have a text that said what the number was, like time, it was on a slide, it was literally printed on a slide.

So when you called up a display, there'd be this Rube Goldberg affair they were very proud of. The computer would display its numbers on a CRT, okay, and here's some columns of numbers—just picture it, columns of numbers—and then a slide would come in over that and another camera would take a picture of the two together, and what we saw on the computer screens was the composite. Well, that's the best we did in those days.

I remember in the *Apollo 13* movie that Tom Hanks was in, I remember folks asking, as they saw these flight controllers sitting there with their slide rules, "With all the computing power, were you using slide rules?" Yes. Absolutely. The computers couldn't do anything. You couldn't interact with them. They just pumped data onto the CRTs. It was very much the paradigm of the mainframe operation in any batch environment where, instead of getting printouts, it came live on the screen, but it was continuous, and it was difficult to change except to select different things to look at that were pre-programmed. So, yes, we were all issued circular slide rules and did our things.

Anyway, that technology absolutely amazed me, enthralled me, and the ability to stare into a computer. Bear in mind, we sat through hundreds of simulations before any flight, and you know it's a simulation, so it's all a game. Well, when I sat in mission control for the first Apollo-Saturn flight and it kicked off and I suddenly realized I was looking at a computer that literally was out in space—and out in space is still new—it got to be awesome again. It was absolutely amazing.

Have you seen the movie *Matrix*? Anybody that stares at this years later is going to laugh at me for this, but there's a scene in there. That movie was the notion that the real world is a virtual world, everything's driven by computers, and there's one scene where there's a fellow staring at this sort of waterfall of numbers on a screen, and the main character says, "Well, what are you looking at?"

He says, "I'm looking at that virtual world."

"You can stare at those numbers and actually see it?"

"Oh, yeah. There's a young woman walking," or something like that, and he's just staring at numbers. That was deja vu for me on a much small scale, because we couldn't get them to put anything on the screen out of the computers except octal numbers. A lot of the progress of the computers was based on what were called flag bits, you know. Buried in a sixteen-bit word, every bit meant something. If it was a one, it meant the computer had progressed past a certain stage in process. I mean, it doesn't matter what. Hundreds of these things.

So if you knew the computer programs and you had sort of memorized all these flag bits, you could stare at these octal numbers, and a single octal digit is a combination of three binary bits, so you have to do the conversion in your head. You can stare at these octal numbers, and they're absolutely meaningless. I mean, the label on the screen would be flag word one, flag word two, and we'd stare at these columns of numbers and say, "Yeah, the computer's doing this, the computer's doing that." People would walk up and say, "You're weird." [Laughter] Well, we knew exactly what we were looking at, and it's completely a foreign language to anyone else walking up. Anyway, that deja vu was kind of funny for me in seeing that movie.

Apollo, yes. Well, sitting in the control center, we learned something quickly, and that was Murphy's Law. If things can go wrong, they will. Software was certainly a new ingredient in spacecraft in those days. It was wonderful because in some ways you could change it so you didn't have to rewire everything if there was a design problem. You could get in there and change the software, not stay the same because we had to weave the ropes or something if we had to make a change.

But we would figure out we could do some reprogramming in real time of the computers. While the bulk of the memory was this hard-wired, there had to be some of what was called erasable memory, the kind of memory you're used to in computers today. Otherwise, the computer wouldn't have any space to do calculations, right? Well, they were smart enough to put breakout points in the software code. Usually in these flag bits, the software would remember where it was, and it would jump out to look at the value in memory to decide where to go next. That was done mostly for reliability. That is, we didn't have redundant computers. There was only one computer in each spacecraft, the command module and the lunar module, just one computer, and it was designed to be internally redundant. That is, any component could fail in the computer and it would still work. Not any component, but it was meant to be ultra-high reliable.

I've totally lost my train of thought. Has that ever happened to you? You're in the middle of describing something and it goes south on you?

RUSNAK: Sure.

GARMAN: Oh, yes, the breakout points. So the software was designed so that it would memorize where it was so that if it was interrupted and didn't know what it was doing, it could jump back and restart. It was called restart. All calculations were done in a way that as the software progressed through—a lot of software is cyclical, of course—but as it progressed through any cycle, it would memorize by writing data in this erasable memory where it was in such a way that if it got lost, it could go back to the prior step and redo it. This was the way it could recover from transient failures, or "I got lost." In mainframes they call it AB-END, abnormal ending. In the mainframe, doing batch programming, if the software throws up its hands and doesn't know what to do, it just flushes the program, does a dump, and says "AB-END." You don't do that when you're flying a spacecraft; you just keep going. So the notion of restart was in there.

Well, by having to put all that in erasable memory, there was a degree of reprogramming that could be done. I remember very clearly that we had to argue to allow astronauts the ability to actually key in octal numbers into erasable memory. The computer interface in those days was a device called the DSKY, display and keyboard, and the paradigm used was verbs and nouns. There was actually a V key for verb and an N key for nouns. So maybe a verb would be "display" and then a noun would be what you were displaying. No letters, of course. It's all numbers. But with two-digit verbs and two-digit nouns and there was, I think it was three five-character displays, that's it, three because in navigation you were living in a three-axis world. So like velocity, there'd be X, Y, Z. There's always three of everything.

Anyway, computer inputs and displays were this series of verbs and nouns. Either the computer would put up a verb and a noun to tell you what it was displaying, or the astronaut would key in a verb and a noun to say what he or she—it was all "he" in Apollo—was trying to do.

We all worked very hard to get them to put in the old verb twenty-one, noun one—I still remember the verb and noun—which was the notion of keying into memory, into a specific memory location, in the erasable memory, a specific value. That was horrific, of course, because you could destroy the computer that way if you keyed in the wrong value. Let me point out that it's just computers. Remember, this is a different world. I mean, an astronaut could also grab the wrong switch and destroy the vehicle. Come on. If something goes wrong, you want to have the ability to go fix it. That's part of the notion of having human beings in space. So that happened.

But then over time, we would create all these little hip-pocket procedures that would solve problems that no one had ever thought of. As I look back on that, I'm horrified, because we'd have them written on the back of envelopes. The consoles had plastic like you have on a table today. We would stick these things under the plastic on the console, and during simulations we'd use them. You know, "Well, tell them to do verb twenty-one and noun this to go fix that or that." They'd go do it, and it would work, and it got to the point that they got mad at us. They literally got mad at us, because, "How many of those do you have? Have they been tested?"

"No. We just figured it would work."

Now, remember, we're young and we're stupid, right? I mean, we were just trying to do the job. So the notion of erasable memory programming, EMPs, got to be very, very popular. I hope I'm on the right program. With Shuttle and Apollo we ended up doing them both, but, yes, it was Apollo. It most certainly was Apollo, yes. I'm looking back. Because Ken [Thomas K.] Mattingly [II] was one of the champions of erasable memory programming.

So after a while, even that got formalized. That is, you'd have all these extra little things that you'd do that we'd document and test in all the simulations beforehand, and they'd actually get incorporated into the crew's checklist. But when you're thinking of workarounds and fixes, that's a never-ending thing, right? No matter what happened, we'd always have a few dozen more little fixes in our pocket. I mean, in the hardware world, people didn't think anything about that. You know, you try to document everything, but, yes, if you had this kind of failure, well, let's try throwing that circuit breaker before we do this to do that. They didn't think of that as a procedure that should have been tested before launch. You were just smart enough to understand the plumbing and to forget that's the thing to do. Well, it's the same thing in computers, but, remember, they're scary, and there's very few people that understand them. "You're looking at what on that screen?" You know, it's kind of mysterious.

This is probably a good point to jump into the Apollo 11 experience. It was amazing. We had lots of flights leading to Apollo 11. I wasn't in on the crews or anything like that. I was just watching, sitting in consoles between flights, doing simulations.

Remember, this notion of restart I've described, where the computer can go back, during simulations in mission control, because I sat in a back room, in a support room, I was never a "flight controller," I say with quotes. There was another group of people in the flight control game in those days. These were the folks that were the trainers. They would think up the problems, the failures to cause. You have to picture in those days that as you got close to one of these "going where no man has ever gone before" kind of flights, they didn't want to put in failures that you couldn't recover from. That would be both demoralizing and it would make the papers. I mean, even simulations made the papers. So they were pretty careful going at realistic things, and they would predict how the flight control team would react and what they should do to correct the problem, and generally they were right. Maybe the flight controllers would come up with something even more clever than the trainers though of, or maybe less, and they'd do a debriefing after every simulation and really walk through it carefully to see if they'd done their job right. This is another form of testing, isn't it? You're testing that the people part of the system is working.

Well, because I was a back-room guy, they didn't think it was cheating to come to ask us for what kind of failures they could put in to make the computers not work, because, again, there weren't a lot of people that knew about computers, much less the onboard computers. So we would help make up failures and then pretend we didn't know what they were. Okay? And it wasn't that bad. We'd suggest one kind of failure, and they'd, of course, doctor it up and make a different so we didn't recognize it. I mean, we were being tested, too.

But I clearly recall helping them come up with a couple of semi-fatal computer errors, errors that would cause the computers to start restarts. Well, it was one of those or a derivation of one of those, it was just a few months before Apollo 11, I'm quite sure it was May or June—and I'm sure you know by now exactly when—that a young fellow named Steve [Stephen G.] Bales, a couple years older than I was, was the Guidance Officer, and that was the front-room position that we most often supported because he kind of watched the computers.

One of these screwy computers alarms, "computer gone wrong" kind of things, happened, and he called an abort of the lunar landing and should not have, and it scared everybody to death. Those of us in the back room didn't think anything of it. Again, we weren't in touch with the seriousness of simulation to the real world. "Okay, well, do it again."

But Gene Kranz, who was the real hero of that whole episode, said, "No, no, no. I want you all to write down every single possible computer alarm that can possibly go wrong." Remember, I'm looking at this as, "Well, we should have thought up a better failure," and he's looking at it like, "This stuff can really happen," partially because he didn't understand the computers, but partially because he's absolutely right. He's looking at the forest and not the trees. So he made us go off and study every single computer alarm that existed in the code, ones that could not possibly happen, they were put in there just for testing purposes, right down to ones that were normal, and to figure out, even if we couldn't come up with a reason for why the alarm would happen, what were the manifestations if it did. Is it over? Is the computer dead? What would you do if it did happen, even if you don't know?

So we did. We did. I still have a copy of it. It's handwritten, under a piece of plastic, and we wrote it down for every single computer run and stuck it under the glass on the console. And sure enough, Murphy's law, the onboard computers ran in two-second cycles, which is horribly long in today's computer world.

The notion of navigation and flight control is such that kind of like if you were walking down the street, you open your eyes to see where you are once every two seconds, and you see the hallway around you and the ceiling and the road ahead, and then you shut your eyes and then decide where to put your feet. Okay? So there's no obstacles ahead of you and you haven't reached the end yet, so you figure you can probably take three steps before you get to open your eyes again. In other words, you kind of interpolate and figure out how many steps to take. So you take three steps. Then you stop and open your eyes, look around, shut your eyes, and go.

That's exactly how the navigation and flight control work, okay? Read all the parameters, do the calculations, and pump out the next two seconds' worth of commands for which way to point engine bells and throttles and so on. It didn't matter if you were in the command module with a service module behind you doing a burn. Or, more importantly in this case, in the lunar module, with that descent engine continually burning and having to continually adjust. Because, remember, they're like a helicopter, only they're riding on top of a plume, right?

Are you about to change that, because I'll pause if you are. Yes. I need a break first. Why don't we just take a break.

GARMAN: So Gene Kranz had asked us to write down every possible computer alarm and what could happen, what might cause it. The computers, as I was starting to describe, are running in these cycles, two-second cycles, for calculating how to drive the engines. When it got to I forget what distance from the lunar surface, the conclusion had early on been that more precision was needed so the computer programs would double their speed, they'd run once a second. Okay? You know, as you're getting closer to the ground, the lunar surface, you don't want to coast for two whole seconds. You want to get a little more precision.

Everybody knew the computers would be a little busier as a result of that. As I recall, there were some things it stopped doing to make up for that, but the fact is, what's called the duty cycle of the machine, that is, how much spare time it had, would drop when it went to the one-second cycle. I believe all the testing had shown that it was maybe running at 85 percent duty cycle when it went to the one-second cycling.

One of the test alarms that was in there was one that said if it was time to start the next cycle of calculations—open your eyes, look, calculate, and so on—if it was time to start the next cycle and you were still in the prior cycle, there's something wrong. This is like when you have too many things to do. It's called bow-waving all those tasks; you're not going to get them done. That's not good.

So the computer would restart. That makes perfect sense. Flush everything, clean it out, look at those restart tables, and go back to the last known position and proceed forward. Overload was not a situation that had ever been tested, ever, that I know of, but because that design of restarting in the case of unknown problems was done so well, an overload is a perfect example of an unknown problem, and it turns out it did recover.

Well, the reason for the overload wasn't known for another day or so, and a day is a long time when you're landing on the Moon. As they got down to the point where it switched to the one-second cycling, one of these computer alarms popped up. This is Neil Armstrong and Buzz Aldrin, and they're standing in this vehicle. They're the first people to land on the Moon, and one of these computer alarms come up, they get this four-digit code for what the alarm is, 1201, 1202 were the two alarms, as I recall. The only reason I remember that is a couple of my friends gave me a t-shirt that had those two alarms on it when I retired. Those were the numbers, all right.

In that system, as I recall, in the vehicle, when one of these alarms came up, it would ring what was called the master caution and warning system. Now, master caution and warning is like having a fire alarm go off in a closet. Okay? I mean, "I want to make sure you're awake," one of these in the earphones, lights, everything. I gather their heart rates went way up and everything. You know, you're not looking out the window anymore.

So this computer alarm happened, and Bales said, "What is it?" So we looked down at the list at that alarm, and, yes, right, and if it doesn't reoccur too often, we're fine, because it's doing the restarts and flushing. It can't happen, but because Kranz had told us to, we said, "All right. Theoretically, if it ever did get into overload, what's going to happen?" Well, the computer's going to have one of these alarms. It will clean the system of all the tasks that it needs to do out so any that are in there twice because of the overlap will be reduced to none, and then when it restarts, it'll only put one of them back. Right? So it's self-cleaning, self-healing, as long as it doesn't continually happen. Right?

Well, there's a two-second delay, just for starters, okay, you know, the speed of light. So obviously it's not recurring so often that the vehicle's going unstable. So I said, on this back-room voice loop that no one can hear, I'm saying to Steve, "As long as it doesn't reoccur, it's fine."

Bales is looking at the rest of the data. The vehicle's not turning over. You couldn't see anything else going wrong. The computer's recovering just fine. Instead of calculating once a second, every once in a while it's calculating every second and a half, because it flushes and has to do it again. So it's a little slower, but no problem. It's working fine. So it's not reoccurring too often, everything's stable, and he does his famous, "We're go, flight," or whatever it was.

When it occurred again a few minutes later, a different alarm but it was the same type—I forget which one came first—I remember distinctly yelling—by this time yelling, you know, in the loop here—"Same type!" and he yells, "Same type!" I could hear my voice echoing. Then the Cap Com says, "Same type!" [Laughter] Boom, boom, boom, going up. It was pretty funny.

What was really eerie was that afterwards, after they had landed—I have to say something about the landing before I go into that. For us, it was over at that point. That is, there's nothing anybody can do in the control center. I'm sure you've heard by now many, many stories, that they're running out of fuel, and [capcom] Charlie [Charles M.] Duke, "We're about to turn blue," and they landed. Everybody's holding their breath.

But the most phenomenal point to me, watching that, was we'd watched hundreds of landings in simulation, and they're very real, and on this particular one, the real one, the first one, Buzz [Edwin E.] Aldrin [Jr.] called out, "We've got dust now," and we'd never heard that before. You know, it's one of those, "Oh, this is the real thing, isn't it?" I mean, you know it's the real thing, but it's going like clockwork, even with problems. We always had a problem during descent. A problem happens, you solve the problem, you go on, no sweat. Then Buzz Aldrin says, "We've got dust now." My god, this is the real thing. And you can't do anything, of course. You're just sitting down there. You're a spectator now. Awesome. Awesome.

Right after the landing, of course, everybody that has anything to do with computers is all over, trying to figure out what the hell happened. I remember Dunseith came in. We had plugged in an extra tape recorder just to tape our voice loop, and I remember Lynn Dunseith came in and asked us to play the tape, because nobody in the front room could hear the back-room conversation. They were just doing the flight director's loop. He said something like, "Oh, my god," and he went walking back into the front room. It was pretty funny.

Evidently, in the heat of the moment, all the attention was focused on Steve Bales and not on me and my colleagues in the back room that were helping him, and I think that was what Lynn was doing, he was walking back out to point out that it's a team effort and all that stuff, which Steve Bales did, no problem, it was. The reason I raise that is we had no idea—you don't realize until years later, actually, how doing the wrong thing at the right time could have changed history. I mean, if Steve had called an abort, they might well have aborted. It's questionable. That is, those guys were so dedicated to landing that they might have disobeyed orders if they didn't see something wrong. But nonetheless, you know, paths not taken, you have no idea what might have happened. That was an extremely, in retrospect, one of those points where you were right at the—you were a witness in the middle of something that could have really changed how things went. So it was very good that there were people like Gene Kranz and Steve Bales and others that kept their heads on and thought about it.

It turned out the hardware guys figured out the problem up at the Instrumentation Lab. There was a grounding problem on the rendezvous radar. The rendezvous radar was used when you ascend to go back up to rendezvous, and they had, I think, left it on or something. There may still be an expert around that can tell you. But the way computers did their input output—at least these computers did it—was to read the analog world, like the position of a radar antenna in this case. They had analog and digital converters that would take a measurement based on voltage of a position and convert it to a number, a digital number. But today those kind of things are written directly into computer memory. In those days, no such thing. What happened is that every time the analog measure moved enough to change one bit, the analog and the digital conversation was simply the act of interrupting the computer and having it involuntarily add one or subtract one to a particular memory location. All right?

So the things of channels or I/O in that sense it wrote directly into memory, which today is called DMA, a direct memory access, but it's not done that way. It's done via hardware. If the radar or something moved one bit down, it would subtract one; one bit up, it would add one.

Well, because they had the radar turned on in the wrong time, there was a grounding problem, and the analog and the digital converter wandered. It's called white noise. It averaged zero, which is what it was, but it would wander up and wander down, wander up and wander down. So the computer was continually being interrupted to add one, add one, subtract one, subtract one, subtract one, add one, add one, subtract one, add one, subtract one, continually, at a high rate. In fact, it consumed 14 or 15 percent of the computer's time.

So when they dropped from a two-second cycle to a one-second cycle, suddenly there was this extra involuntary load on the computer meaninglessly adding ones and zeros and minus ones to some of the cycles that caused it to run out of time. It was running at 101 or 102 percent duty cycle, meaning there wasn't enough time to do everything. So sooner or later, it caught itself wanting to start another cycle before the prior cycle had finished. It flushed, that cleaned everything out, and because it was just marginally over 100 percent, it would run along for several seconds before it finally caught up and flushed again. Hence, the problem.

Well, you can imagine—you can imagine—that made everything fine for the Apollo 11 landing. I mean, they got the problem solved, we'll switch this right, this won't happen again. But what else could be in the vehicle that would cause the computers to run over 100 percent duty cycle to the point where they couldn't survive? It would restart so often that it couldn't do it. This starts a search for problems that went on for years.

In fact, jumping ahead, in the Shuttle onboard computers, we actually ran tests where we kept stealing cycles from the machine until it failed, to find out what that point was. We'd steal cycles. And the Shuttle onboard computers, because of that Apollo experience, were designed to be fault-tolerant on that kind of cycle. We'd steal upwards of 60 or 70 percent of the computer cycle, and the displays would freeze—they have real CRTs on the Shuttle—the displays would freeze because that wasn't high enough priority. The highest priority thing is driving the vehicle, and the displays would freeze, things would start falling off, but it would keep driving the vehicle, and finally, at an incredible 60 or 70 percent stealing, it would finally fail. That's graceful degradation, okay? That was the notion. But that's unheard of in onboard digital computers. Onboard digital computers reviewed is gears, okay? Cycle, cycle, cycle, if things turn, they repeat like a clockwork that continues to go, and you're supposed to pre-program this stuff so that it can't run out of time, just make sure there's always margins.

Well, that's where I learned how to give presentations, of course. After that, this young fellow who was one of the few people that could actually explain what had happened, didn't have to wear a NASA badge. The Draper Lab folks could explain it, but they didn't talk English very well. And I mean that in the kindest sense. There's this notion of you learn how to translate when you talk to a group of people that speak a slightly different language and don't necessarily understand. You learn how to listen to the experts who do know what they're talking about and translate it into the language of the other folks that need to understand it, and I found myself in that position quite often in this computer jazz. So I ended up giving lots of presentations to very senior people. It scared the living bejeebers out of me, but it's where you learn to give presentations and talk.

We spent a lot of time, of course, making sure that kind of problem would never happen again. It didn't. As I say, when we went into Shuttle, it became a great debate, which I'll talk about in a minute, on how to make sure those kinds of problems don't happen. But they etch deeply. Those kind of near misses etch very, very deeply into your mind when you see how close you can come to really doing some bad stuff.

The other adventures in Apollo were, well, the lightning strike on Apollo 12, you know, during launch. No problem. The computers restarted, right? The onboard computers, the ones in the spacecraft, I mean. Apollo 13, that was everyone's nightmare, you know, the long nights, but there were no computer problems per se. The notion there was to get the lunar module computer, which was designed to guide the lunar module to land on the surface of the Moon, to instead become the push vehicle to push this stack. Never even thought of. Actually, it had been thought of. It had been thought out a bit, thank god, but the notion of using the lunar module to push the whole stack, rather than the service module, was a profound change that they worked through in real time.

But again, that was mainly a testing issue. It was not changing what were called the e-loads, erasable loads, the parameters that we put in that erasable memory that you couldn't hard-code. There were enough of those that you could virtually re-program the onboard computer by doing that, and that's what was done, and that was largely the navigation people, the flight control people, and all the Instrumentation Lab folks checking all that out. So I was pretty much an observer in the Apollo 13 experience, as nervous as anybody else.

Bill Tindall was the hero there. He ended up calling endless meetings, almost twenty-four hours a day, not the kind of meeting that you think of in a bureaucratic sense. I mean brainstorming meetings. "Have we really thought of everything? Is there anything more we can do?" That movie was good, but it took many, many people and they compressed them into one, because you can't capture in a movie all the folks. So it was inaccurate in that sense, that you have lots and lots of people, each experts or semi-experts in their area, sitting around these tables arguing and trying to figure out if everything was being done correctly, if they'd thought of everything possible, as that movie portrayed, not knowing what the answer was, in many cases, until almost too late.

Apollo 14 was the solder ball or whatever it was, switch, abort switch. That's one of those—you wake up and have a—what do they call it in war? These long days of boredom and tedium punctuated by moments of pure, stark terror. That's how I remember Apollo 14, because Apollo 11 just happened and it was over. Apollo 14, we're sitting there getting ready to—they turned on the lunar module for the first time, still attached to the command and service module, and the crew's over there, starting to get ready to separate to go do the landing. Right?

By this time—you remember my notion of staring at all these numbers on the screens? By this time, they'd figured out how to put what we call event lights onto the consoles. They don't do it anymore, but they did then. These are small lights that we put printed labels on, had colors, that were driven by these bits, some of them hardware, a lot of them software. So rather than trying to read all the actual digits, you just had these banks of lights all over your console—I had hundreds of them—of every significant event or item or error, and if it was a problem, we'd color it red; if it was a caution, it was yellow; if it was something good happening, we'd color the light green. But all these lights.

So instead of staring at bits on the screen, there's this mass array of lights, and sure enough, the labels are all there, but after a while you don't care what the labels are, you remember them positionally, right? Just like on the typewriter. Most folks, if you erase all the letters on the main keys, you can still type just fine as long as you can figure out where to place your hands. You learn positionally where things are.

So now people would walk up to a console and say, "What do all those lights mean?," and we'd say, "Well, this is what's going on." We could just tell by the patterns of lights. Well, it's hard to miss a red light, and this particular event light was on many consoles because it was the abort switch, you know, on or off. It had a lot of meaning in software, it had a lot of meaning in hardware. And they would throw the abort switch purposefully during a test to make sure it would work, so it wasn't terribly odd to have the light come on and go off. I wasn't the guy tracking the testing of switches. And I forget who saw it. It doesn't matter. I mean, everybody saw it and said, "The abort switch is on. Is it supposed to be on?"

"No."

"Why is it on?"

You start talking on all these back-room loops, these voice loops where you could talk to each other, and the call up to the crew, "Would you verify abort switch off, please." Of course, the crew—that's code. They know that something's not right, because by this time we've gone through the checklist and know damned good and well it's not supposed to be on at this point, and the crew knows it, too.

"It's off."

"Would you toggle it, please."

They toggle it, and when they turn it off, it goes off. They turn it on and turn it off, and it went off. Well, you know, that doesn't help a bit. [Laughter] Oh, I don't remember whether they toggled it or they hammered it or they did something, but it went off. It went off. I think they actually hit the console. I think that's what happened. I guess toggling didn't work, but hitting it did, and that's where they started concluding it was a hardware problem.

So if the conjecture was that it was a solder ball or something loose in there, which I guess turned out to be the case, I don't think they ever knew for sure, because by the time the vehicle gets back, the solder ball's—the vehicle doesn't come back, you know. It's gone. But assuming that was the case, they concluded quickly that they're at zero G right now, and the act of lighting the engine suddenly puts gravity, artificial gravity as it were, and whatever is in there loosest is going to start flying around again and could absolutely do it again inside the switch.

A fellow named John Norton, he'd be a good one to get hold of if you ever can, TRW in those days. I don't know where he is now. He was a genius. Like many geniuses, he had trouble communicating with management, okay, but in the computer game, he was a genius. TRW had many roles in those days, but part of it was continuous independent assessment of what was going on. John's task was to look at all the onboard software code and do what is today called code inspections. It's a normal part of testing. It wasn't done in those days, except that John Norton did it.

The way he'd do it is, he'd take this awful assembly language and translate it back into his own version of a readable higher order language. The Norton Document, as we called it, that he put out for every version of every program, all typed by hand—no word processing in those days—was our Bible. We actually used it the same way somebody might use a Fortran listing or higher order language listing of a program to analyze their program.

Now, bear in mind, that isn't necessarily reflecting what the program is. I mean, if you write in a higher order language, C++, ADA, whatever, you're looking at an untranslated version of the actual ones and zeros. It's a computer program that translated—or in this case, a fellow's head—that did the translation. So it's risky to be dependent on that. But, you know, we're dumb. We figured that was close enough, and that's what we used. That's what we used to come up with these erasable memory procedures I talked about earlier, too, which made it a little risky. They always worked because John was always right. He never made a mistake. Well, he made a couple. That's a story he can tell you someday, maybe.

As soon as this happened, we opened up our Norton Documents and started looking for flag bits, remember, hard-coded stuff. The first thing we determined was that the minute the engine lit, the minute it lit, it would be shut down and it would abort, because that's the way the computer was programmed and that's hard code. It would assume that the crew just—first it cycles and reads it environment every two seconds, including all the switches, and it would read the switch and say, "Oh, time to abort. I'll do exactly what I'm told," and separate the descent part of the vehicle and ascent and fire right back into the command module. No, we don't want to do that.

On the other hand, if there was a way to disable it, then how do you abort if it's time to abort? I mean, this is a Catch-22. We immediately figured out a way to disable it. Remember verb twenty-one, noun one? The way to abort would be to key in verb twenty-one, noun one, to put the bit back. Okay? And a couple of us, we had that on the table within ten minutes, but that's very dangerous. You know, if you're aborting, you may be spinning. It may be impossible for the crew to hit that many keys correctly, and they're right, that's not the way to do it. I suspect that would have, they would have taken the chance, but not if there's any time.

So we kept digging, we kept digging right there. Well, we had a direct line, voice line, to the Instrumentation Lab, right to the console in those days. Those were very expensive in those days. I can't remember his name, young fellow out there who got right into it, right with us, and started digging really deep. My memory, and he figured out—about the fifth iteration, we came up with something that worked. It was pretty foolproof, and that's what was used.

I think more entertaining for this was my own sense of what was going on, because I'm sitting there in the back room, there's a team of us, and we're staring at this, and we've got all these—remember, computers were just things that display stuff in those days. We've got these documents just all over the place now, papers flying, the Norton books open. We're animated and talking, "What about this? What about that?" talking to the Instrumentation Lab, talking to other people in the control center.

I looked around, and standing behind me were about ten people. Every icon of the space program was standing behind me. I mean all of them: Gilruth, Kraft, all of them. Tindall, Dunseith, everybody. It's like a private turning around and seeing all the four-stars standing behind them or something. It scared the you know what out of me, because I woke up that we were in serious trouble at this point.

See, they're in lunar orbit, and they have only so long before they have to abort and come back. Not abort in the flash sense, but you can only sit there for so long. So we only had like two hours. That's the worst nightmare of all. Right? I mean, like Apollo 13, give me a day or two. But this is like two hours. That's all we had. I may be wrong. Maybe it was four hours. I forget. It doesn't matter. But it was a short time that we had to come up with a solution.

And worst of all, we had too many solutions. It was a risk-gain thing. We had, do this one, but they have to key Verb 21, Noun 1 to abort. Do this one, but maybe it'll happen anyway. Maybe this one isn't as good and there's a probability it'll abort by accident, by, by god, the abort switch will da, da, da, da.

Eyles. Don [Donald.] Eyles at the Instrumentation Lab figured it out, and as soon as he identified it, everybody went, "Yep. That's it." You know, when you're all searching for the same answer and somebody has the "Aha!" They tested it, read the procedure up, stuck it in, and the landing went on and everything was fine. But, boy, you talk about those moments of terror. They were kind enough, as I recall, to get out of the way, because when you know you're being watched, you don't always work better, you sometimes work more nervous. But that really was, that really was one of those—worse for me than Apollo 11 was, because there was too much time and too many people watching us.

I don't remember offhand any other major problems, all sorts of minor ones in onboard software in Apollo, but nothing that made the papers, so to speak.

I will share one other event, if I can flash back first, things that you remember forever, you know, when you're in this. Aside from that Buzz Aldrin "We've got dust now," call, which woke me up, the other one that hit me was before Apollo 11, it was Apollo 8. It's not a big deal, but for several of us it was one of those "Oh, my god, we're really doing this" kind of moments again.

The onboard computers worked on a coordinate system since the Earth is the center of an X-Y-Z coordinate system for doing navigation, calculating gravity, all those kind of things. Apollo 8 was the first time we'd sent a vehicle all the way to the Moon. It had to have people in it, too. Right? But it was the first time. Just to go around the Moon and come back, that was the flight that was done over Christmas and all that. We had one of these event lights on the console. We had a few in those days. The event light was set that when the vehicle got closer to the Moon, it would switch coordinate systems. Instead of coasting away, you're suddenly falling to the Moon. At the point where the Moon's gravitational field is stronger than the Earth's, you've gone over the top of the hill and you're starting to fall.

For navigation purposes, the computers would do a—I mean, you're coasting, no big deal, you're just coasting on the way to the Moon, but the onboard computer for the command module would do this big switch, would recalculate everything and switch to a lunar-centered coordinate system, and that was a light.

At two o'clock in the morning or something on the way to the Moon, you have nothing to do. Long moments of boredom waiting for something horrible to go wrong. So we were guessing the exact point at which the light would come on. Now, navigationally, we knew exactly the point that the vehicle was actually going to cross this threshold, as accurately as they did navigation in those days, but we're down to a gnat's hair here. We're trying to figure two-second cycles, how long it will take, the transport time. We're trying to take bets on exactly when that light's going to come on on the console, you know, the lag time for the telemetry to come down and all this. Yes, we're a little nuts, but what do you do? It doesn't matter who won the bet.

The light came on and we all stared at it and said, "My god. Do you know what we just saw? We saw a human being for the first time that we know of, ever, being outside of Earth's gravitational field." Right? Because what that light meant was that they were falling towards another planet, body, up there. So that another one of those—you know exactly what you're doing, you know exactly what's going on, but when something actually happens, you get that sort of gut, "My heavens, it's real. They're falling towards the Moon." And that was pretty awesome, pretty awesome at that time.

RUSNAK: Well, as you've mentioned, for these missions, you're, of course, in the staff support room doing these. Can you describe that environment, the typical kind of mood? You've mentioned that there were these long moments of tedium followed by spurts of excitement, but just the sort of general atmosphere, a kind of description of what it looked like, felt like, that kind of thing.

GARMAN: Sure. There's both the humorous end and the serious end. The humorous end would be that in those days we didn't have computer graphics like we have today. You didn't have Excel or Lotus to draw diagrams for you. What computers spit out were columns of numbers. So if you wanted to visualize something, a plot or a graph, it was usually done by hand. Okay? So you'd have somebody sit there, take long rows of numbers, and painstakingly plot them on plot paper. So all of the analyses—in fact, in the Mission Planning and Analysis Division, for years, there was a group of, oh, I don't know, maybe twenty or thirty young women—they were all women; this was still a very sexist time, by the way, as you'll see in a moment—young women who did this. That's what they did. They plotted. They were called math aides. Okay?

And this was very serious stuff, by the way. I mean, you can imagine how easy it is to misplot something. Okay? If that's used to make a decision on how we're going to do something, it can be very serious. So accuracy in manually plotting and creating plots was extremely important. It really was. Some folks were very good at it. Some of these young women were very good at it.

But during the missions, they'd select some of the best of these young women to be in the control center. They actually badged them and they'd be there, because they'd have these big tables with an overhead television camera looking down on them so that if we needed something done in real time during the mission, they could sit there and plot something for us, and we'd call it up on the screen, using the screen as a standard television, and talk about it.

But of course the other reason that they were in there was that the average age in mission control was about twenty-nine, maybe twenty-six, and they were all quite pretty, and they loved to get us coffee and do things like that. During Apollo 8, I was attracted to one of them who later became my wife. So that's the humorous part of that. You wonder how many times that kind of thing went on, all those little side stories about the space program, that we met in mission control during Apollo 8, got married right after Apollo 11, which was pretty funny. Raised two daughters. She went back to work and ended up joining NASA again as a NASA person, rose very quickly, which was sort of entertaining in the later years. We'll get back to that. Stick to the chronology here maybe.

The staff support rooms weren't carpeted like the main control room was. They didn't have the big screens up front. Although in the one I was in, which was called the flight dynamics staff support room, the trench staff support room, if you will, we had plotters, which were very expensive, with a camera so we could point it at a plotter, an X-Y plot, you know, a vertical device with a pen that could plot on a sheet of paper. We didn't use them a lot, but that was very expensive gear for those days. So that was about the only unique outfitting in the staff support room.

The control center had very, very tall ceilings, and everything was raised flooring, and I mean real raised flooring. It was like two or three feet down below the floor. So if you look at the old control center today, it's like a five-story building but it's only three stories because each was so big. So you'd get into these staff support rooms, and you'd get this feeling like you're in a big cathedral or something, because the ceiling is way up there. You have these rows of consoles. They're all standard, whether they were in the main control room or a staff support room. There were two rows of consoles in there. One of these plot tables was over on the side with the camera, with the young lady sitting there and these plot boards way up in front laid out kind of like the main control room is with its big screens out in front.

A human-interest thing on that was that we'd be in there all sorts of hours, pad tests, simulations, what have you, and we had these communications panels. They were analog, not digital like they are today, but the same idea, where you'd have, in those days, a white button that you would punch and it would flash on the voice loop that you were talking on. Then there were amber-colored ones for ones that you could listen to. So for many loops, you had two buttons, both the amber one for listening and the white one to talk. So, for example, the air-ground loop, very few people had a white button for air-ground, the Cap Com, maybe the flight director, and a couple of others, but we all had the amber button. We listened. The flight director's loop the same way.

In fact, if you were in the staff support room—I think later on we got the flight director's loop. We kept getting these strange problems that we'd explain, and it got very clumsy for the flight director to have to either go off his loop or listen to. We finally got the flight director's button, I think, in later years. In fact, they moved the position into the front room. It's called DPS [Data Processing System] today. They moved it. It's an interesting transition story from Apollo to Shuttle, which I'll jump into in just a second.

Anyway, in the staff support room, there was a volume control on these loops. The volume control, the amber. Okay? So the notion was that the loop you were talking on, even if you had just the talk button, what you heard was a constant volume on that, relatively loud. All the amber ones that you could listen to—you could listen to many loops at once—you could turn that volume down. The idea being that if someone was talking to you on your loop, it would come in loud so you wouldn't miss it. The other thing the human mind is good at besides pattern recognition and my notion of all the numbers or lights, is listening to many conversations at once. You do it in a room all the time. You can be talking to someone and hear a side conversation and jump over to that and then jump back to your conversation. Right? You can do that.

Well, you can imagine that, with nothing else to do, we'd want to hear everything that was going on. So we'd have half a dozen different loops running in the back of our head and listen to all of them at once while we're carrying on a conversation, while we're watching lights and displays. It was mesmerizing. People could sit there for hours and feel like, when they left, they'd just awakened. It's kind of like driving. You know how you can get deep involved with your driving and sort of forget, "Did I really come here? What path did I take?"

"How many stop lights did you go through?"

"I don't know. They were green, weren't they?" You know, you just lose it. You're conscious and you're working, but you're kind of mesmerized.

Anytime we were in the middle of significant operations, it was kind of like that. You'd sort of lose it. You were totally immersed in all these voices in your head, all these lights and displays in front of you. In our case, and anybody that did a system that was far, far away, you also continually have this mental image of what's going on that you're trying to put together in your head, because it's not real, what you're staring at, it's just numbers.

Some of us, because we were in the staff support room, we had a speaker on the console, too, so we could have a third thing. We'd plug into one voice set and then with the speaker on the console connected to another set of buttons, we could turn on more voice loops. So that you had the plug in your ear with the lower volume listening, the speaker with another set in your other ear, and the loud one, which is you talking, back in the same ear. Oh, it was weird.

Particularly funny was that we'd talk to people to get things done. "My CRT's broken. Could you get someone to fix it?"

"Are you sure? Things are running slow here."

There was a bunch of console positions that had to do with just running the control center. These were staffed by people from my own division. Remember, I was in the division where we worked onboard computers, a small group, but the rest of them were all mission control, and I didn't know half of them. Over time, you'd get so you'd recognize voices and call people by their first name. "Hey, Joe. How are you doing today?"

"Yeah, doing fine."

"Are you on console next Saturday for the big test?"

"Yeah. See you then."

You know, you talk back and forth and have no idea what they look like. You work in the same building, pass each other in the halls every day, and had no idea who they were until at some meeting, "Oh, you're Joe. How are you doing? I'm glad to meet you. I've been talking to you for years." It's one of those really eerie experiences that happens when they set up a system that's supposed to totally immerse you in what you're doing, and you don't really get to see all the other people around. Every once in a while you do.

What else can I tell you about staff support rooms? There was a lot of concern about too many cooks will spoil the broth. Right? They were very careful about letting people into the MOCRs, the Mission Operations Control Room. I remember clearly, if you had a green badge, you could get into the control room. If you had a pink badge, you could only get into the staff support room. They guarded it fairly well, particularly during critical phases like landing. They'd actually have guards there. Otherwise, it was just honor system. I mean, pink was pretty visible if you were walking into a room. You just didn't go in.

It didn't really bother anybody, except that every once in a while it was pretty important to just say, "Look, let's sit down and talk," and in the main room, they'd just stand up and turn around and talk to each other. I'm sure you've seen this in pictures and so on. They'd stand up and talk. But you couldn't do it if you were in a room down the hall.

So early on, all of us with pink badges didn't worry about it particularly, but after a while they started issuing these green temporary badges. In other words, they'd issue me and the team of five or six of us on this—AGC support was what we called ourselves, Apollo guidance computer support position, and the AGC support position was the lead and had to get two or three of these green temporary badges, and that made life easy. We'd just stick one on and walk into the room and we could talk.

I remember for a lot of people later on, it was one of those things that, "Do you get a green badge or a red badge?" For people that didn't sit at the console, that they would come in and do things, like the young ladies, the math aides, they all had green badges. I mean, how can you take coffee into the—that's not nice. How can you take a plot into the main room if you don't have a green badge? So it was one of those kind of games. I guess that's a way of saying there was a sort of ranking or a class system. But it didn't really work, because some of the best experts were the ones that weren't in the main room. It was sort of an odd kind of a hierarchy. Control was in the control room. The decision, the power to make decisions, was in the main room.

RUSNAK: How did that relationship work between the guys in the front room and those of you in the staff support room or directly supporting them?

GARMAN: Wonderfully. Remember, we're all about the same age. In Apollo, nobody's ever done it before. There was no question that everybody had a different job. There was very little overlap. And we're all very goal-oriented. We're fighting this battle, is what it boiled down to, how to get people to the Moon and back several times. So I recall the relations as being great, wonderful. People would debate and argue and yell at each other, but it was always in the debate and argue and yell.

Now, looking back, I now visualize, I can remember, there was politics all over the place, people trying to get promoted and all that, but when you're twenty-one to twenty-five, you don't think about that, not when you're a witness. So I think a lot of experiences are colored by the old you are and who you know. So at the bottom level, where I was, it was wonderful. There was no competition per se. Everybody's on a team. Everybody's afraid we'll get in trouble from the bosses somewhere, so they always try to do the right thing.

I remember once—there was no fence around the center in those days, and I remember once I decided to take a hike and just walk as far as I could go, and I ended up at the lake past the West Mansion, you know, east of the center. I ended up walking all the way to "Mud Lake" [Clear Lake] because there were no fences on that side. When I got back, I told somebody in the office about it and how it was really neat just to go hiking out in the woods and the prairie and so on.

"Oh, boy, are you lucky you didn't get caught. You'd have gotten fired."

I said, "Really?"

"Yeah. Yeah. You're not supposed to go walking out like that."

You know, you're twenty-one or twenty-two years old, and I didn't know all the rules. I didn't care what all the rules were, and nobody told us. Anecdotally, just sharing that you have a completely different perspective on the way things work. "Fired? Give me a break." I didn't care. They didn't want you out there getting hurt, getting lost, doing things, of course, "You shouldn't do that." What the heck.

Shuttle?

RUSNAK: Well, just before we move on, I wanted to ask, as you're immersing yourself in Apollo, your specific missions and the job that's going on, did you ever take time to kind of stop and look around at what's going on in the rest of the world? Are you paying any attention to larger events that are going on outside the space program?

GARMAN: Well, the Vietnam War, for sure. I mean, most of us working there were either—there were three kinds. There were folks who were already through the military. There were folks in the military, either in the National Guard on reserve or we had some military folks actually stationed at the center. Or there were folks like myself who had gotten a deferment. You know, NASA said, "We want them to work here, not in the military."

They'd write a deferment letter, and the deferment boards—what do they call them? The draft boards, they didn't care who you worked for, government or otherwise, but they'd have a certain quota of young men to call up, and they'd rank them in order of need, and if you were working for NASA in those days, you were generally pretty low on the list.

Growing up in Chicago, I was in a huge draft board, so I was lucky. It was by the numbers. If the quotas were normally sized, as they generally were, then I ended up escaping. But we had several folks that didn't. They'd be in a small draft board. I remember one fellow in our group, a fellow named Sam Ankney, Walter Sam Ankney. It was ‘67 or ‘68, I guess, and he got called up. He got drafted. He was from a small draft board in Oklahoma or something, and they just ran out of names. So we had a party to toast him goodbye, all this kind of business, and off he went. He was back two weeks later. He failed the physical. [Laughter] So we had another party. Welcome back. He stayed with onboard computers all the way through Shuttle, all the way on into about—in fact, he just retired a couple of years ago, still working the same, but Shuttle onboard computers by that time, I remember.

But, you know, when you’re totally immersed in this thing, that old notion of you live to work as opposed to working to live. No, I don’t remember too much about other events. I mean, we’d read the newspaper and watch television like anybody, but most of the party conversation was either what young people talk about or work, not current affairs or anything like that. But again, I was a computer geek, so maybe it’s always like that for folks that get into something they’re pretty passionate about.

BUTLER: If we could pause.

GARMAN: Now maybe is the time to talk moving from Apollo into Shuttle. Skylab happened in there, too, of course. My involvement wasn’t very deep because I’d been moved on onto Shuttle.

Let me talk about the times first. You’d asked me earlier about watching current affairs or what was going on. No. So it was a rude awakening after Apollo when the nation went through the “Is that all there is?” kind of problem with the space program, and we were faced with something called a RIF, a reduction in force. For those of us that had only been with NASA for five years, four years, three years, the government in those days, and still largely today, if it has to do reduction in force, it’s based purely on seniority, how long you’ve been with the government, whether you’re a veteran or not. Most of us weren’t.

So that was traumatic. That was very traumatic for young people, wondering if we were going to lose a job. In retrospect, it wouldn't have mattered. We'd have gone on to something else. As I've watched contracts get re-competed and I see contractors lose a job because their company didn't win, the ones that are any good, it was the best thing that could have happened to them. They move on to something better. But it doesn't help the trauma. It doesn't help the trauma at the time.

I remember one of my good friends, still around, still works for NASA, a fellow named John [W.] Jurgensen. Jurgensen was one of my colleagues on the consoles throughout the period. He ended up being RIF'd. They hired him back soon, as I recall, but in a different location at a lower pay, and he worked his way back into the system over time, but it was pretty nerve-wracking for folks. I think it was that sense of "Look at what all we did for the country. Why is the country doing this to us?" kind of thing. Maybe some of that. Not in a real negative sense. I think everybody understood what was going on, but it was what happened.

Well, I'm sitting there going through these Apollo flights. Let's see, the last one was in '72. The agency had decided, through it downsizing and everything, they decided to move on to Space Station at that point. Of course, they couldn't get funding for Space Station, but one of the ingredients of Space Station would be a vehicle that would be a taxicab to go back and forth. Like what do you call a taxi? A shuttle, something to shuttle back and forth. So they called it the Space Shuttle. The notion would be to get to low Earth orbit cheaply, reusable components, rather than all this throwaway, you know. Apollo's thirty-six-story building leaves and the only thing that goes back is this little command module at the top. Now, let's see if we can't do it, using [current NASA Administrator Daniel S.] Dan Goldin's phrase, "faster, cheaper, and better."

So they did the usual, let a contract. Rockwell won it, what was later called the Space Division of North American Rockwell, but it was North American Aviation when they first won. Many of us were moved immediately into that program, myself included. I continued doing some support in the consoles, as I recall, but the award was the summer of ‘72, was when the first Rockwell had won, North America had won. We went out for our first big meeting to talk with them. Talked about organizing at NASA to do Shuttle onboard software in my case.

But what was going on in the mission control was that, remember I said we were support, we were never flight controllers, and after those Apollo missions, Gene Kranz concluded correctly that it was pretty important that he get some flight controllers that were computer experts so they could do that. So during the last—oh, I don’t remember exactly when, but they started lending me people. I remember clearly the people at Dunseith asking me, “Are you sure you don’t mind?” It was like they were taking my job away.

I’m saying, “Why should I mind?”

“Well, you’re really good at this stuff.” He didn’t want me to go over to be a flight controller, see.

“No,” I said. “I don’t want to be here.” After you do enough of this, what you end up doing is following somebody else’s procedures and spending your life punching buttons and staring at screens. By the way, that is a derogatory way of—you can say the same thing about people that do computer programming. All they do is stare at numbers and write code and do stuff and make it work. I mean, there’s a derogatory way of talking about any job, but it wasn’t my turn-on. I was happy to start migrating out of that.

So, yes, they took some folks from the Flight Control Division and had them sit with us during the latter Apollo missions at this AGC support position, and then when they reconfigured for Shuttle, they created a data processing system, or DPS, position in the front room.

One of the fellows that went through that, I don’t think he worked with me in Apollo, but he ended up going through the DPS positions, a fellow named Randy [Brock R.] Stone, who’s now the head of MOD [Mission Operations Directorate]. Several of them. Ron Harbrow [phonetic] was another one, a guy that just retired, that was the EVA manager, astronaut. He went through the DPS position, too.

But bear in mind, except for the Skylab Program, which was basically from a computer standpoint the same thing, it was just using the Apollo spacecraft in lower Earth orbit, and I had nothing to do with the Skylab vehicle itself, except for that, what was happening was flying a, sort of in the background, Skylab while the main work force was trying to build the Shuttle.

Skylab happened, I think, within a year or so after the last Apollo flight. Then they had one more last gasp, the Apollo-Soyuz mission, remember. '74 maybe. Then that was it. No flights. No flights until the first Shuttle flight in '81. So the agency went through this period of spending money like crazy but having, quote, "nothing to show for it." Okay. That's good in some regards. They spent a lot of time redoing mission control and stuff like that, fixing things up, getting ready, but I do recall it was real problem years for the agency from a money standpoint because the customer's the public, and like movies, they want to see action, and they weren't seeing a lot.

Okay. Well, the interesting thing about Shuttle coming in was that after all the Apollo experience with software, Chris Kraft, who by this time was center director or deputy center director, still under Gilruth, had concluded that if you wanted visibility into the development of any program, what you want to do is hold onto the software.

Well, if there's hardware problems, the first thing they'll try to do to fix the hardware—I mean in design—the easiest thing to change is the software. Once you start cutting metal, it's hard to change hardware. Software is easy to change. So if there's a problem, you'll see it because it'll come in as a change to software. Probably the most difficult part of the Shuttle would be the software because it was designed to be a purely fly-by-wire vehicle, all digital.

So when North American wanted in the Shuttle Program top to bottom, he forced innovation that is a movement of the software contract, which IBM was subcontractor, out and directly linked to the government. So the software was the one component of the vehicle, of the Space Shuttle, that North American was not responsible for. It was GFE, government-furnished equipment.

I love controversy, love being thrown in the middle, and there we were, right in the middle, because you can imagine, here's a bunch of government engineers that came off of Apollo, and IBM's now working for us and formed this division called Spacecraft Software Division, SSD for short. A guy who had—I don't think he'd done a lot of software work before, but a really brilliant fellow named Dick [Richard P.] Parten was plucked out to be the head of that. He had been, at that point, I think, heading the institutional computing operation in Building 12. This was '73, I think.

So they pulled the contract out, formed the Spacecraft Software Division, and plucked a bunch of us in—that was already in it, but to really be in it—and off we went to go work with Rockwell—I'm going to call them Rockwell; they change names sooner or later—to work with Rockwell on building the onboard software in the computer they were building. Well, IBM was building the computer, too. The contract for the hardware, the IBM computer hardware, was a different branch of IBM, and it was through Rockwell. The software was another branch of IBM through the government.

Now, that created some interesting times. There's always a debate on some subject, you know, "Do it this way. Do it that way." The Shuttle onboard software, the debate was whether to use synchronous or asynchronous operating system. I'll explain it shortly and then give you some of the adventures that went with it.

Asynchronous is what Apollo was. Asynchronism means that a computer responds to interrupts or activities on a random basis and goes and does what it's supposed to in order of priority. It's how you think; it's how you work. If I feel a tickle in my throat, I can keep talking and still grab a cup of coffee. I don't have to think one step at a time. It's sort of like thinking in parallel.

Synchronous operating systems are clockwork. You allocate time slots to processes, and when it's time to do a process you let it go, and priority doesn't mean a lot because there's a serial order to things, as in gears and clocks.

The only digital fly-by-wire system that had been built, other than what the Apollo Program had done, it was largely a digital, too, but truly fly-by-wire, no hydraulics, if you move a stick, you're not doing anything except moving switches, the only prior one had been the F-18, as I recall, fighter aircraft, that had been done by North American Aviation. So the folks they had for Shuttle were all refugees—I'm kidding—were all veterans of that program, as we were refugees from Apollo.

Well, this is kind of like two different religions meeting, because we'd been through this experience of what happens when the unknown happens and a computer doesn't fail gracefully if it's a synchronous system, it just crashes. Yet behavior in a computer system that's constructed synchronously is totally predictable. You can test it down to a gnat's hair. When you work with asynchronism, the exact path that a computer takes, what instruction executes in what order, is never the same. It's random. It's not predictable. This drives people who want to test to a gnat's hair berserk. Literally it drives them crazy. There's the joke about how to drive an engineer crazy, tie him down in a chair and have someone open up a map and fold it the wrong way. That's exactly that sense of driving people—you've got to be nuts. You cannot test software that's built asynchronously. You've got to be nuts. You cannot build software that will degrade gracefully under unknown failures. And this debate went on for years.

Well, the IBM folks came from a mainframe environment that always ran asynchronously. That's the way it was done. We came from an environment, with the Instrumentation Lab, that had built an asynchronous operating system, that worked under some pretty wild conditions, not just the Apollo 11 one, which was spectacular, but lots of other cases. It had survived failures and went right chugging along, can't AB-END. And the Rockwell folks came from a very successful fighter aircraft that worked. Okay?

Well, when you're GFE'ing the software, the government wins, except that another real name in this game is a fellow named Sy Rubenstein. I don't know if you've found him, had a chance to talk with him. I think he's still around. I do. But I'm not sure. I remember his having some medical problems. But Sy was the grand architect of all the avionics in the Shuttle, all of it, Rockwell, and he was smart, willing to work through all sorts of political hurdles to get the job done, including this damned GFE software from the government. I remember lots of long debates with him and his folks, but in the end, the government compromised.

There were five computers in the Shuttle. They called them general purpose computers, or GPCs. They were not general purpose, but that's what they were called. In the beginning, the five GPCs were set up such that four of them would be redundant and do the flight control and navigation, the thesis in Shuttle being "We're not going to build ultra-reliable hardware. We're going to use cheaper hardware this time and just be redundant, and we'll be FOFS." Have you ever heard that term? "Fail op, fail safe," FOFS. So it takes four.

If you have four—I'm holding four fingers up—if you have four and one device fails, the other three can vote it out. You know who's right. Well, you're still operational with three, because if you have a second failure, you can still vote it out. But on the second failure you only have two. You're safe, but you can't take another failure because you don't know who's wrong. So the whole notion on Shuttle was to try to have quad redundancy to get fail op, fail safe. It's kind of hard to do that on wings. There's some pretty fundamental things you can't do it on, but in general, on systems that could fail, they had quad redundancy.

For example, there were accelerometers, things that measure acceleration. They didn't go to quad redundancy on accelerometers. What they did was have software that would take the inertial measurement unit that measured attitude, and by calculating as the vehicle moved, by calculating how fast the vehicle was moving, rotating, you could calculate acceleration. So that was used as the fourth accelerometer in terms of being quad redundant there.

Why am I telling you this story? Well, because it turns out that you can have quad redundancy on computers so that any hardware failure you're going to be fine on, but you don't have quad redundancy on the software. You have identical software in four machines, and if there's a bug in that software, as we proved many, many times in the Shuttle Program, all four machines will obediently do exactly the same thing, vote each other doing exactly the same thing, right into the tubes, because there's a bug.

Well, Shuttle flight software was extremely carefully and expensively done. There were not many bugs. Didn't happen. In an asynchronous environment, okay, here it comes. In an asynchronous environment, the order of processing is unpredictable, and, moreover, you end up having parallel processing. It's not really parallel, but the computer stops doing one process to go do one of higher priority and then jumps back to continue the other, so it's as if it's parallel.

The bugs that would happen in the end were not the kind of bugs that you think about, a plus sign instead of a minus sign or an "IF-THEN-ELSE" that isn't right. They were purely asynchronous kind of issues. If, in fact, this event happens before that event and simultaneously the crew hits a button at this point, then because of this interrupt-driven environment, the subparameter is not passed between the two parallel processes correctly and you've got a bug. They're endless, very, very difficult to find.

So Sy Rubenstein talked the program into building a backup flight software system their way. Their way. So the fifth computer, which had been designed to run payload and systems management kind of stuff during ascent and entry, was instead allocated the notion of running backup flight software.

Now, a backup system in the Apollo or any normal vehicle sense is a completely alternate system. That wasn't the case here. It's the same computer. In fact, any one of the five computers could be designated the backup computer. Okay? Same requirements, same algorithms and equations, driving the same hardware. What was different? Same programming language. They used the HAL [High Order Assembly Language] language to do it. They used a different compiler on a different host computer, but it was still the same. What was different? The operating system. They built it synchronously. They built it absolutely synchronously.

So the probability then of one of these timing kind of bugs happening, while vanishingly small, to be really bad, in the premise it was vanishingly small, but if it did happen, it became diminimus that it would happen in the backup too, because it was a totally different operating system. It just ain't going to happen. And it doesn't. It never has. It never has.

So we went through the expense of building a completely different version of the flight software, and Rockwell got their wish, all right, they got to build the software, too. They got to build it, too. In retrospect, it was the right thing to do, and I've had Sy Rubenstein years later tell me, in retrospect, asynchronism was the right thing to do in the primary. You know, things tend to evolve to the right answer or a good answer over time.

The interesting thing in the Shuttle onboard software development was that it ran out of resource very early on. That is, I remember clearly in the '78, '79—the first orbital flight was in '81 as I recall, April of '81—I remember clearly that when we added up all the code and all the processing time required to do all the software, we were at 200 percent or something. When you're running this in simulation, you can artificially make the memory of the computer bigger and all that, but it wouldn't fit. So we went through scrubs, continually scrubbing or reducing requirements. In fact, Fred [W.] Haise [Jr.], who was one of the Apollo 13 astronauts, he led one of the more infamous scrub teams in the Shuttle Program as we approached the first orbital flight. This is a point to mention something.

By the way, that may be a good ending to that theme of discussion, that the Shuttle onboard software was built three or four times in the primary because of scrubbing and built yet another time in the backup to have this alternate form of processing to make sure there were no bugs.

But a segue here into software engineering for Shuttle was that we did it in higher order language. It was the first time we'd ever done it. What I'm going to tell you, the punch line, is that we would have never succeeded had we not done it in higher order language, because when you scrub software, it's real easy to say, "Let's take out that function," but when you try to find that function and all of its entrails buried in assembly language, it is almost impossible. You end up creating more bugs and more problems. When you're working in a higher order language, it's much easier to change software. So the notion of using a higher order language, which today is, you know, "You've got to be kidding. Nobody does assembly language." Well, some people do. It goes without saying today, but in those days software still weighed something. It was expensive. That is, you know, weight and memory and electricity and all that flying a vehicle, it was a big deal.

So part of our Apollo heritage was a company. John Miller was the founder. He's Air Force Academy grad in the forties that grew up in the Instrumentation Lab, I think on the hardware side, formed a company called Intermetrics. Today it's called Averstar. They have an office here. There were two key guys, a fellow named Fred Martin and another one named Dan Likely [phonetic]. I think they're both still around. Their notion was to take the Apollo experience that they'd had in software engineering and so on and make a company out of it.

Our notion after Apollo, and NASA had a program for research called RTOPs, Research Technology Operating Plan, it was the name of the form we had to fill out to do research, RTOPs. And I was asked to do RTOP on software engineering. So I wrote one to go do a higher language for the next program. This was during the Apollo-to-Shuttle transition. Intermetrics won that contract and proposed this HAL language, and we went out and did it, went out and gave them a contract, and it was another one of the GFE components.

What was amazing was that we had in '72, '73, at the beginning of the Shuttle Program, what we had now was Rockwell building the—with IBM building the computer hardware, building it, literally, they chose an off-the-shelf AP-101 computer that was used in fighter aircraft, but they had and they built a whole new input-output processor around it so it was a different computer. We had IBM Federal Systems Division doing the flight software, and that was through the government side, and then we had Intermetrics building this compiler that they were going to program the software in, that was also a government contract being GFE'd through IBM. Okay? I mean, talk about getting us in the middle, getting the government in the middle.

What was even more amazing was that, for whatever reason, this computer that was selected had no operating system. The AP-1 did not come with an OS. I mean, this is like buying a new computer chip and having no operating system. So there was nothing. It was an absolutely naked computer when we bought it. IBM said, "We'll design one. We'll build on." Ended up, we called it the Flight Computer Operating System, or FCOS. I became lead on that, of course, after the Apollo game.

In the game of software, compilers' languages speak to operating systems. That's the most intimate relation. Operating systems are software that actually run in a computer when the computer's running, whereas compilers generate the code that runs in the computers. So when compilers compile, they're not running in the computer. Although nowadays, a lot of compilers run in the computers they actually generate code for. But in the host target sense, it certainly was in the case in the Shuttle; the host for the compiler was the IBM mainframe and the target was the GPCs.

So the idea of having these higher order languages became very compute-intensive, as you can imagine, because you're talking about trying to translate something that more or less English into target code for a GPC, talking to an operating system. You don't replicate operating system software. A lot of it's simply calling the routines that exist in the operating system. What operating system? It didn't exist. So we ended up, from '72 through '74 or so, building in parallel the compiler, the operating system, and the applications. This had never been done before. It was stupid to try to do it that way, but we were already late by the time the contracts were awarded. So that's the way it was done. And on top of that, trying to build it in such a way that we could have computers running redundantly, running exactly the same code, voting each other, hundreds of thousands of times a second in what we called sync points, synchronization points.

In late ‘74, ‘75, it was Christmas one of those two years, we were finally required to deliver some software to the vehicle at Palmdale for testing. Why? Remember, we weren’t going to fly until the Approach and Landing Test [ALT] in ‘78, first vertical flight later.

RUSNAK: If I could stop you for a second, we’ve got to change out the audio tape.

GARMAN: I was talking about the notion of building all these things in parallel, the compiler and the operating system, building the computer hardware, by the way, at the same time, to the applications. The other piece I neglected to mention is that we’re building the test facility at the same time. We called it the Software Development Lab. We ended up using the leftover mainframes from mission control. They were actually in the Mission Control Center. I forget the numbers. I guess they moved—these were IBM-360s. They moved to IBM-370 mainframes by this time in mission control. All this stuff’s going on in parallel.

My point at that break, that earlier break, was that we were called on to deliver some software to [Rockwell in] Palmdale [California], where they were building the vehicle in 1974 or ‘75, I can’t remember. Why? Well, because this vehicle, the Shuttle vehicle, is basically all driven by computer, it turns out they couldn’t test the vehicle without computer software. In retrospect, it’s just patently obvious. I mean, you know, just think of a PC. How do you check out a hard disk without putting software in to make the hard disk work, right? It’s obvious in retrospect, but remember, the mind-set here is aircraft of yore, spacecraft of prior days, where these systems were designed with test points that you could test them. There’s a lot of testing they could do without the computer, but in the end, they had to have some test software.

Well, you can’t even build test software unless it runs on an operating system. We’re building things in-house. In other words, you wanted to take a drop of where we were in developing the software, add some simple algorithms that they wanted just to do testing with, and deliver it to them. And we did, and it didn’t work. It just flat didn’t work.

And all of a sudden, all of a sudden, based on a requirement that had not been identified in the first place, software became the driving factor in the development of Shuttle. If you can't test the vehicle, you can't finish building it. If you don't have the software to test the vehicle, you can't test it, so software became the long pole. Of course it was the long pole. It wasn't Rockwell's fault. The software was GFE, which added to the political game. "We told you we should have done the software in the first place."

On top of that, we couldn't get this game of synchronization to work. Not only could we not get software to work, we couldn't get multi-computers voting, couldn't get it to work. I can remember when we counted the number of successful sync points in the hundreds. Bear in mind, the software that runs today on the orbiter synchronizes thousands of times a second. So this was pretty rudimentary kind of problems.

I was in the middle of it, as were a lot of people. We ended up getting some serious help from senior management. And when you're in trouble, you end up getting serious money, too. You know, it's a shame to think of it that way, but there's two ways to get more money: be successful or fail. If it's not working and you've got to make it work, you pour more money into it. If it's very successful and you want to do more, you pour more money into it. So don't ever be in the middle of mediocrity. You might get paid. No, don't go there.

Anyway, it all came through, of course, but that was a couple of very painful years, as I recall, particularly for people like Dick Parten. I'm still—what am I? I'm thirty years old in '74, so I'm still relatively young in the game and doing my thing and having fun at it, but I do clearly remember the pressure. I remember Dick Parten and I having an argument once, which we did often, friendly arguments. He was absolutely one of these "I want a schedule, I want a plan, I want all the steps laid out." And I would look at him and say, "You know, we have no idea how to do some of this stuff." The problems had not been solved in building an operating system that could do this synchronizing between machines. "You're asking me to schedule creativity."

And he'd look at me and he'd say, "Yep. That exactly what I'm asking." [Laughter] You know, "Set yourself a damn time limit for when you're going to solve these problems and get them solved."

How do you do that? You know, because you sit around and you're trying to figure out how to do it. But that's exactly the position everyone was in, whether you're sitting at a console on Apollo 14 with two hours to solve a problem before you can land or you've got a year or so to solve a problem and you don't know the answer. That's the way life works. If you don't know the answer, you've got to figure one out. I guess it's what makes engineering and science fun, part of what makes it really neat stuff.

Anyway, we did, we ended up figuring out. A lot of very smart people at IBM and among our colleagues at NASA figured out answers to all that stuff and got it working.

The sidelight in here that I want to talk about is this Software Development Laboratory. There really wasn't such a thing in Apollo. They had the computers that did simulation mainframes, they did their assembly language assembling on big machines. The crew trainer for Apollo, you know, the simulators that the astronauts train in, actually ran an interpretively simulated Apollo guidance computer. ISAGC it was called. A fellow named Jim [James L.] Raney, who was around for a long time, I think he's still around somewhere, created that, ended up working all the way through Station, in fact, on onboard software.

But in the Apollo days, the onboard computer was so slow and the mainframes were so much faster, they're slow today, but compared to the onboard computer there's a big gap between the speed of a mainframe—these were Univacs they used in their simulator—a huge gap between the speed in mainframes and the slowness of the onboard computer for Apollo, that they actually loaded the listing, the assembly language listing, of the onboard computer, and it would interpret that listing and run it.

Why? Because software changed so often that they couldn't afford to keep changing the simulation. As a result, it actually helped debug the software. As often as not, the simulation was wrong. The simulated Apollo guidance computer running this interpretive thing of the listing of the computer program would do perfectly, and the simulator couldn't handle it, and it was a simulator problem. But as often as not, it was also a software problem.

Well, in Shuttle, they went one better. They actually put GPCs, real GPCs, into the simulator and surrounded it with a box, whose name I can't remember, that fooled the onboard computers into thinking they were actually flying when, in fact, they were sitting inside a simulator and could be stopped, reloaded quickly. You know, simulations, you run to a point, stop, reset, go back, go on, redo, and stuff like that.

Now, they couldn't do a lot of diagnostic work, but here we weren't actually loading the listings, we're loading the code, and because the GPCs are running in real time, the fact that the GPCs were much faster computers, and the difference between the mainframes used to simulate in a simulated environment and the onboard computer was much narrower, but we distributed the load. You have the real GPCs just running their real software. So the simulator just had to keep up with real time, which is what simulators have to do anyway. So that worked out okay. Very complicated.

The same thing was done in the Software Development Lab. The Software Development Lab, the device there was called a Flight Equipment Interface Device, FEID, which we pronounced, of course—any acronym longer than three letters NASA pronounces. You heard me say NASA. I didn't say "N-A-S-A." I said "NASA." It pronounces. Three letters, we say it. Software Production Facility, "S-P-F." And we pronounced that; we called it "spiff." But the FEID, F-E-I-D, we pronounced it.

The FEID was the same idea. It was a box that you plugged in the actual GPCs, and on the other side of this box was the mainframe, where we ran the compiler's developer code, but, more importantly, ran all the simulation to simulate actually flying the vehicle to test the flight software. Okay? Now, this is all digital. In a crew trainer, there's a lot of real gear. This is all digital. There's no cockpit. It's all simulated.

Moreover, we didn’t have to go on real time. There’s no people there to keep up with. So simulations could run much slower than real time and often did. Why? Because the idea of testing software would be, maybe I want to track a variable in an equation, and I want to see it every time it changes, on a printout or somewhere. Well, the FEID was constructed so you could do that.

Now, bear in mind, I’m looking at a HAL listing, and here’s the name of a variable maybe called altitude. And they actually spell it out: altitude. “Hey, that’s English. I can read it.” Where the heck that variable was in the computer memory is lost in the notion of compiling, first to binary code and then in this world of asynchronism—well, actually, we did static memory allocations so you knew where it was, but when it changed value, it was not predictable.

So the idea of the FEID was that you could put triggers in and it would actually freeze the computer, put it in suspended animation, reach into its memory, and pull variables out, pull parameters out. Well, this causes it to run slower than real time. So can you see the distinction? In a simulator, in a crew trainer, same idea, real GPCs, but you’re not stopping it except maybe to restart a send. You’re trying to run in real time. Whereas in this Software Development Lab, we’re stopping it all the time.

Well, not only did we have to deliver software to Palmdale and had to check out the vehicle, we had to deliver software for crew trainers as they were building the crew trainer, to help check out the crew trainer, much less trying to make it work at all using these FEIDs. The time spent solving those problems was endless. Now, bear in mind, that was my first experience in development. When I came into Apollo in ‘66, it was all designed, right or wrong, and everything was trying to make it operate. I got into Shuttle kind of on the ground floor, and so I got in the middle of all these fun things of trying to solve problems.

So we have a Software Development Laboratory, we called it. The host machines are old IBM-360s that ran the control center, the real-time computing complex, or RTCC, as they called in the control center for Apollo, we inherited those computers and built this thing called a FEID to plug these GPCs into, and we're trying to deliver software out to all these other places.

There was another facility called the SAIL, Software Avionics Integration Laboratory, had the same idea. It also used a device like a FEID with real GPCs, but it had real gear. Its notion was not necessarily that it had to run real time, but it could partial tests. It wasn't trying to do everything. They'd try to pull in real hardware and make sure—you know, hardware built by that subcontractor or that subcontractor or this one, put it all together to make sure it runs together. That was the purpose of the SAIL.

So we're sitting there from '75 through '78 or so, having to send software, not real software, test software, to three or four different locations, first to help develop the facility and then to help develop the hardware or the vehicle itself. That was another of the detractors from focusing on actually building the real flight software.

I'm a little out of order here, but if you can add it up at this point, you can see the challenges that were faced here. Remember, 200 percent, the code didn't fit, had to continually scrub it? All right? Trying to develop whole sets of software to send out to laboratories that needed it to exist themselves. Plus, underneath all this, trying to develop the actual software that's going to fly. All right? It was really entertaining times, and it's amazing it was pulled off. Now, I recall people saying the same thing after Apollo. I wasn't deep into the development enough to realize why they say, "I can't believe we did all at," but the same thing on the Shuttle, I can't believe we did all that.

In the approach and landing test flights, yes, that's Fred Haise flew that, and it was after that flight he got into the scrub for the first vertical flight, all the software scrubbing. Fred Haise, I was in the control center watching that, not at a console, just plugged in on that one, and the instant separation happened, off the back of the 747, you know, fired the explosive bolts, one of the computers failed.

Boy, you talk about heart rates rising. See, we were still—there was no issue on our part that one of the other three computers would take over. I mean, that was fine. None of us really knew for sure that we didn't have generic problems, that whatever failure happened in the first computer would immediately happen in the second, third, and fourth, which would have been a disaster for the Shuttle Program. I don't think we had a BFS then. Yes, we did. They had a backup flight system, but it was—oh, you didn't want to go there unless you had to. Anyway, that was shaky, but it worked. Amazing, you know. You plan for the unknown and a failure, and it was, it was a hardware failure, and things work like they're supposed to, and you go, "Wow. Okay. That's good. That's good."

The adventure of Shuttle, the next place I found myself sitting in front of audiences I didn't want to sit in front of, was on the first orbital flight, STS-1. You remember I said that the backup flight software was designed completely different than the primary, synchronous versus asynchronous. The backup flight software system was never in control unless they threw a switch to put it in control. It had always to be ready to take over control. So it had to continually listen to what was going on in the vehicle to keep up. How do you listen to another set of computers? You listen to the bus traffic, to the LAIO. You can't use those computers. They may be wrong. You can't hand the primary computer—hand any parameters to the backup because they might be wrong. They might be failing. But you can listen to the raw data.

So when the primary computer asks for a value from an inertial measurement unit and the data comes floating across the bus, the backup computer could listen to that and pull the data in. To do that, it has to know when the primary's going to ask. Said another way, the backup and the primary have to be synchronized. The primary operates in an asynchronous world. The backup operates in a synchronous world. This was like trying to mix oil and water. Okay? It was done by brute force. It worked, except that. Except that. There was one minor little big, and I'm not going to bore you with the details of the bug. I don't even think I could remember them all. But there was one bug that was one of these weird probabilities, one in sixty chances, that if you started up, first turned on the primary computers and within a certain time period then turn on the backup computers in the orbiter, they'd never synchronize. The backup would not be able to hear the primary correctly.

Never found the bug. Why? Because it was like a one-in-sixty chance, and in all of the SAIL runs, computer simulations, FEID runs, and all of that, you don't turn on the computers and start from there. You start from a reset point. A reset point may be hours before launch, but building these simulation resetter initialization points is like a tree. You start with the trunk, and you run a while, and you build a branch for one for this, and another one for that and another one for that, and you build off that one, but they're all based, just like parentage, on the same trunk. And the particular trunk, or two or three trunks, that were used were not the one in sixty that caused the problems.

Murphy's Law. The first orbital flight of the Shuttle, they turn on the computers for the first time to start the countdown and hit the one in sixty chance, hit it cold. As we start coming down, the countdown things are not working quite right. They're not synced up. There's problems. So it's another one of those "Are we going to fly or not going to fly?" Of course, this is good, we're sitting on the ground. Still, this is very embarrassing. This is not good. This is the first orbital flight and software is the problem.

What was the problem? They held up the launch for two or three days, as I recall, and I think it is the only time ever that software has caused an impact to a NASA flight like that, that it's caused a shift or caused a major change. Yes, I found myself talking to newsmen and all that kind of stuff after being up all night. We finally figured out what the problem was, and then of course the fix was real easy, right? Turn the computers off, turn them back on. One in sixty chance, and, sure enough, didn't hit it. It's not a problem that could ever reoccur. See, once you're synced, it's not a problem ever again.

So, absolutely Murphy's Law. If things can go wrong, they will. One of those wild adventures. I clearly remember staying up all night in a conference room. I don't think we had to post guards, but it was close to it, to keep people out so that we could talk it out. You know, you kind of know what the problem is, and you start trying to focus in. It's an absolute classic brainstorming session with people drawing on the blackboards and trying to figure out how to do this or that, what the issue was. It took all night.

Once it was explained, it was too late, of course. The two-day slip was simply the turnaround. Once you decide not to launch, you detank the vehicle and start the countdown again, and everything went fine. STS-1.

The adventure on the Shuttle, of course, was the launch and landing. That's the hard part. Computers can't stop, they can't fail, you've got to keep going.

After Challenger, we all know the dangers of launch, but unlike the Apollo Program, where the computers in the spacecraft where the people sort of monitored it, Saturn had its own flight computer, the instrumentation unit. For Shuttle, the same onboard computers actually control everything, launch all the way up. The Marshall folks didn't like that very much. They wanted their own computer system, but—well, there's some reason for that, the same reason that Rockwell wanted backup flight software. If there's more components, there's a higher probability of failure because any one of the components can fail. There's a smaller probability of failure because you can focus separately, you don't get things mixed up with each other. So it's a real trade on systems reliability.

Anyway, while the main engines of the orbiter, of the Shuttle, do have their own computers, all they do is—I shouldn't say "all they do." It's very complicated what they do, but they control the engines and that's it. They manage the flow of fuel and when to turn them off, when to turn them on, stuff like that. But all of the guidance, which way to point the engines, what throttle level and all that, comes purely through the onboard computers of the Shuttle, which is another reason it had to be built four times, because all the logic that people wanted to put in to cover all the instances just didn't fit.

Okay. One other adventure in Shuttle that was interesting. It's how I started moving into mainframes. At this point I'm all onboard computers, what you'd call mini or micro computers today. They're avionics kind of things. Well, you can tell. I mean, we used mainframe computers to do the compiling and the simulating, and we'd inherited all these old Apollo RTCC computers, and we realized as we approached STS-1, the first orbital Shuttle flight, that if the agency was into anything like what is said—by the way, they were talking about flying fifty-two times a year back in those days, a flight a week.

It didn't matter whether it was fifty-two times a year or eight times a year. The fact is, the parallelism that I talked about earlier was enormous. Determining what payloads to carry in the Shuttle and all the work that goes in and all the flight software parametric testing that has to be done, we're talking a long lead time. I remember sitting down one day, again with Lynn Dunseith, and saying, "We've got a problem," because the flight-to-flight, I mean, we're building software and the parameters and the testing, and everything's driven towards this first orbital flight. Get it done. All right?

The problem with Shuttle was—and in Apollo it was one flight after another, but it was going to end. The problem with Shuttle is, it never ends. It's going to go on forever. It's still going on. There's not just five or six developmental flights and then we're done. We're supposed to get up to many, many flights a year. So I remember drawing diagrams for them that said, "All right. All right. Let's lay out launch dates," and I'd put a point on a calendar, you know, a big picture, a big piece of paper with calendar across the top, a three- or four-year calendar, and you put, "Let's not even bother with fifty-two flights a year. Let's just put one every other month." So six points in a year scattered evenly across a chart. Now, let's draw a line back from that launch date. When is the software cutoff? When is the this cutoff? When is the that cutoff? And we came up with like an eighteen-month lead time.

Well, if you're flying every month, that's eighteen parallels. If you're flying every other month, it's nine parallels. Right? If it's an eighteen-month lead time. Wow! The load on simulators to run that many tests simultaneously, the load on crew trainers and everything, unheard of. Hadn't even thought about it. I'm not accountable for having thought of it. I mean, I was one of the people that thought of. A lot of people woke up to the volume problem we were facing.

So we said, "We need a bigger computer." Well, in 1978 or '77 or whatever, we realized—every once in a while you think ahead. So we're still in the middle of the trauma of doing the software, but we're looking ahead. When a bunch of engineers say they need a bigger computer, the people that have money say, "Yeah, right. You always need a bigger computer. Sure. Prove it." So we proved it.

"Nah. We don't have enough money."

Also there's a local representative named Brooks, Jack [B.] Brooks, in this area, who in the mid-sixties had gotten mad at IBM and got a bill passed called the Brooks Act, that caused the government to go through all sorts of competitive operations before it can invest any money in computers. So the act itself was fine, but the overhead for getting any major expense on a computer was humongous, all sorts of planning and cost analysis and business cases.

So all of a sudden, I was out of the avionics SPF, Software Development Lab business, and we were charged with building the IT plan, information technology plan, for buying a new mainframe and construction of facilities. Where are we going to put it? We decided we'd stick it on the third floor of Building 30 in the admin wing, and we got the third floor, build raised flooring in there, and stick it in as a new computer facility, which became called the Software Production Facility. It still exists in the same location.

Complicating things worse was that DOD was getting interested at this point in a flying secure missions, so we had to build part of the facilities secure so it could do classified operations, you know, radiation-proof so you can't sit outside and see what the computers are doing inside.

So all of a sudden, a bunch of us were in the middle of acquiring and building a data center. That's what it boils down to, a big mainframe data center. See, heretofore we'd been using leftover Apollo stuff, right? The raised flooring was there, the operators were there, the computers were there. We just used them. All of a sudden we're building a new data center, and that was a migration for a number of us, but if you can picture my own experience, I've been working these onboard computers for years and all of a sudden I'm a mainframe guy. And as we'll talk about later, then I became a PC guy, desktop computing. NASA's been really great that way. I mean, you talk about getting—I like to say to people, "I've been in computers my entire career with NASA."

"Oh, and always on computers?"

"Yes, but what a variety." One end to the other, all over the place. Great fun.

Anyway, as most things happen, perseverance wins out, and we got the computers and built the facility. Of course, it wasn't big enough, but it turned out that it didn't need to be because there was no way to fly Shuttles that often. The SPF was sized about right, and I think it survived any major upgrades for a long time. More automation, I think, was the thing that drove it to get bigger over the years, not a capacity thing, because if you only fly eight flights a year and you thought you might fly as many as fifty-two, you come out pretty well sized, even if you were guessing conservatively or being forced to go conservative.

What else on Shuttle? Shuttle was pretty much over for me soon after the first flight. I remember I'd known a lot of people that they get in charge of something and get very closely attached to it, and they lose sight. It became an empire to them. I remember in the Software Production Facility, one day I was giving a tour or something to people and was very proud of it, it's the new data center, lots of money, lots of people, operations big time. I remember I was patting one of the mainframes, and I looked at myself and I said, "You have got to get out of here." [Laughter] So I said, "I think it's time for me to go do something else. I'm getting this relationship with the gear that I don't want. I'm beginning to feel like I own it."

So I got out of the Spacecraft Software Division. I think I'd become deputy division chief, and John Aaron, whose name I'm sure you've known, he was division chief by then. I got out of that and went up to the directorate staff to start working future projects, in particular the Station program. Now, this was in '82, long before Station was a program.

There's one other point I'll raise on Shuttle, and then I think we can say we can put the Shuttle discussion to a close. One of the interesting things that happened when Spacecraft Software Division was first formed was, what kind of people do you gather in a division like that? There were a bunch of us geeks, and I say that in the best way. When you are one, you are one. But we knew the systems, we knew operating systems, we knew compilers and all that, but we weren't experts in navigation or guidance or electrical systems or what have you.

So Dick Parten was given permission to go pick whoever he wanted to start this new division, and one of the characters he picked was John [W.] Aaron. John had been the EECOM, electrical environmental communications officer, in Apollo 13 when all the electrical—the blowup, but it was basically an electrical problem, like no electricity, hit him, and he was pulled over.

Boy, you talk about an adventure for a young man. He knew nothing about software. Okay? Nothing. And ten years later he was head of the division that was still building the stuff, which was very good. The only reason I bring that up is there were several people—Bill [William A.] Sullivan is another one. He was a guidance, navigation guy. Jim [James E.] Broadfoot was another one. I think he ended up being brought in for systems management. I was the operating system guy. Bringing these people in from different disciplines to form a new program like that had that real spirit of camaraderie and "You know things I don't, and I know things you don't, and let's see if we can't share and figure out how to do this." That worked very well.

In closing the discussion, I have to be real clear on one thing. It's real easy to talk in an egocentricity standpoint. I'm talking about the NASA people. I know I've mentioned IBM and

Draper Lab. The view when you're sitting where they were sitting is completely different. I know it was. I've talked to them. They were very happy to work with some of us because we were good customers and helped. We didn't hinder. If something went wrong, we shared the accountability, we failed as well as them. But in many cases it's not that way at all. The government people are just in the way. They make us cross T's and dot I's, hard to get the job done, it's very difficult.

It would be good to talk—I'm sure you have, but it would be good to talk to some of them in this, too. IBM no longer exists. IBM exists, but the Federal Systems Division, which had the core of the people that built the onboard software for Shuttle, was sold off. IBM sold it off to first Loral, and so Loral picked up the onboard software continuing contract. Then when the space flight operations contract—USA [United Space Alliance], when USA was formed, USA picked that up, and those people all moved over to USA, what was left of them, what was left of them. I don't have any names offhand for you, but some of those folks are still around and they'd be good to talk to.

So I think that's a good closing point for now.

RUSNAK: Thanks again for your time today.

GARMAN: You bet.

[End of interview]