Will the Hubble Space Telescope Compute?

Critical operations software is still a mess—the victim of primitive programming methods and chaotic project management

FIRST THE GOOD NEWS: two decades after it first went into development, the $1.4-billion Hubble Space Telescope is almost ready to fly. It is now undergoing its final ground tests and checkouts at the Lockheed Missiles and Space Company in Sunnyvale, California. This summer it will be shipped to the Kennedy Space Center in Florida to be prepared for lift-off. And in December, give or take a few more slips of the space shuttle schedule, it will finally be launched into orbit.

But now the bad news: the Space Telescope Science Institute in Baltimore still has dozens of programmers struggling to fix one of the most basic pieces of telescope software, the $70-million Science Operations Ground System (SOGS). SOGS is what the astronomers will use to plan and perform their observations. It was supposedly completed 3 years ago. Yet bugs are still turning up as fast as the programmers can fix them, and the system currently runs at only one-third optimum speed. For the record, officials at the institute and at the National Aeronautics and Space Administration (NASA) say that SOGS will be ready in time for launch. But they also freely admit that if Space Telescope had been launched in October 1986, as planned at the time of the Challenger accident, it would have been a major embarrassment: a superb scientific instrument crippled by nearly unworkable software.

SOGS' troubles can be traced back to the late 1970s, when the concept first took shape. Even then it was clear that writing the telescope's operations software would not be easy. The constraints on what the spacecraft can do are complex, to say the least (page 1439). When NASA appointed a committee of scientists and engineers to draw up the ground systems requirements in 1980, it took them more than a year and the resulting document was 2 inches thick.

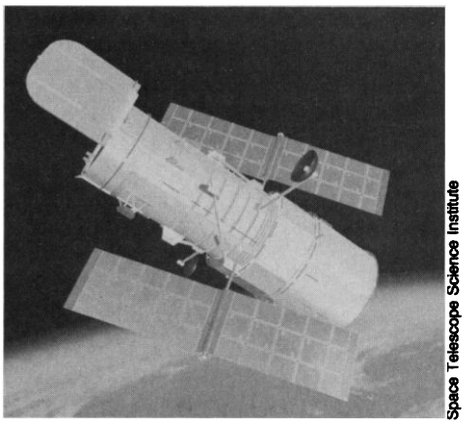
On the other hand, there seemed no reason to think that the software would be inordinately difficult, either. The idea was that the science institute would take in astronomers' proposals to use the telescope, review them, and then use SOGS to turn the list of accepted proposals into a detailed schedule: what objects to look at when, which instruments to use, how long to maintain the exposure, and so forth. That schedule would then be transmitted to the telescope via the Tracking and Data Relay Satellite System (TDRSS), a set of three communications satellites set up by NASA and the Department of Defense to provide near-continuous radio contact with the telescope, the space shuttle, and various other spacecraft in low Earth orbit. Finally, the telescope would use the same route in reverse to send its observational data back to the institute, where SOGS would be used to analyze and archive them. A contract containing NASA's detailed specifications for SOGS was put out to bid in 1981, and was duly awarded to the TRW Corporation of Redondo Beach, California.

At its peak the TRW team included 150 people. Their job was not made easier by the fact that the telescope itself was still under development, so that they were constantly having to update the software to take account of evolution in the hardware. Nor were they helped by having to work in the dark, without much input from the scientists who would actually be using the system: the science institute had only been organized in 1981, at about the same time that the contract was let, and was still in a highly embryonic state.

But by all accounts the TRW programmers made a yeoman effort. And as various pieces of SOGS were completed, the contract supervisors at NASA's Goddard Space Flight Center were able to certify that SOGS did, indeed, meet the agency's requirements. The software was accordingly forwarded to the science institute, where the first components began to arrive in 1983.

By this time, unfortunately, it was also becoming all too clear to the astronomers that "meeting the requirements" was not the same thing as "working as desired." The institute eventually filed several hundred problem reports, which together added up to a declaration that SOGS was utterly unsuitable.

By far the most glaring problems lay in SOGS' fundamental task of planning and scheduling: the system took about ten times longer to schedule a set of observations than the telescope would have taken to perform them. SOGS could not even keep up with normal operations, much less cope with glitches, or unexpected events such as supernovas. Indeed, the software ideally should have been at least three times *faster* than the telescope. If Space Telescope had actually been launched in 1986, say institute scientists, they would have had to operate their $1.4-billion instrument by hand, at a very low level of efficiency.



Space Telescope Science Institute

First light. *An artist's conception of the Hubble Space Telescope in operation.*

The problem was basically a conceptual one. NASA's specifications for SOGS had called for a scheduling algorithm that would handle telescope operations on a minute-by-minute basis. That is, SOGS explicitly dealt with all the gritty details of pointing the telescope, verifying its attitude, acquiring the proper guide stars, checking the availability of TDRSS transmissions, and so forth. The tacit assumption was that the system would schedule astronomers on a monthly and yearly basis by simply adding up thousands upon thousands of these minute-by-minute schedules.

In fact, that tacit assumption was a recipe for disaster. It was a bit like planning a cross-country auto trip in terms of which suitcase you are going to carry to the car first, whether to unlock the door before unlocking the trunk, whether to adjust the seat first or the rear-view mirror first, and so on ad infinitum. At this level of detail, the number of possible combinations to consider rises much faster than exponentially, and can quickly exceed the capacity of even a large computer. In the computer science community, where this phenomenon has been well known for about 40 years, it is called "the combinatoric explosion." Accepted techniques for defusing such explosions call for scheduling algorithms that plan their trips with a road map, so to speak. And SOGS simply did not have it.

In addition to performance issues, howev-

er, SOGS was also deficient in basic design terms. "SOGS used last-generation programming technology," says one senior programmer at the science institute. "It's as if a lot of the lessons of the 1970s didn't get learned."

For example, modern programming practice calls for writing software to be "machine independent," so that it can easily be transferred to new kinds of computers as the hardware evolves. Yet SOGS was configured to run only on a certain line of VAX computers from the Digital Equipment Corporation, as per the 1981 NASA contract. Those VAXes were state of the art in 1981. But they are fast becoming obsolete as researchers move more and more to high-powered workstations linked by data networks.

Furthermore, modern practice calls for programs to be put together from modular building blocks—self-contained pieces of software that can be pulled in and out of the system and upgraded independently. Yet SOGS was not. "SOGS was designed in such a way that you couldn't insert new releases without bringing down the entire system! For days!" says the science institute's associate director for operations, Ethan Schreier. "We spent a good year redesigning SOGS so that we could disconnect the pieces, upgrade them offline, and then reinstall them while operating."

Indeed, the fundamental structure of SOGS is so nonmodular that fixing a bug in one part of the program almost invariably generates new bugs somewhere else. Institute programmers say they are resigned to a debugging process that will essentially go on forever.

So, where did SOGS go wrong?

TRW officials declined to talk about SOGS with *Science*, saying that they were concerned about appearing to criticize their customer, NASA. But Schreier and many other observers close to the project emphasize that TRW fielded a crew of hardworking programmers who were, and are, concerned with producing a good product. The problems were not with the troops, so to speak, but with the generals.

One of the main villains seems to have been the old-line aerospace industry approach to software development—or for that matter, to hardware development as well. "There is a style common in the business that I call Black Box Engineering," says James A. Westphal of the California Institute of Technology, principal investigator on the telescope's Wide Field/Planetary Camera instrument, and a member of the committee that originally drew up the SOGS requirements. "It says, 'Give me the requirements and specifications, and then get out of my way while I build it for you.' That style is attractive to management because it allows them to divide the work up into convenient pieces. But there is only one problem: nobody, including me, is smart enough to anticipate everything."

Indeed, in the wider computer science community this Give-Me-The-Requirements approach is considered a dismal methodology at best, especially in the case of advanced development projects like Space Telescope. For one thing, any effort to pin down the details in advance ignores the fact that specifications inevitably evolve as the hardware evolves, which means that the software may well be obsolete before it is finished. For another, it tends to keep the programmers isolated from the people who know what is going on. "When SOGS finally arrived at TRW," says Westphal, "they did not have technical advice from users who really understood what the system was supposed to do. Nor was there anyone whose job it was to make sure TRW knew what it had to do."

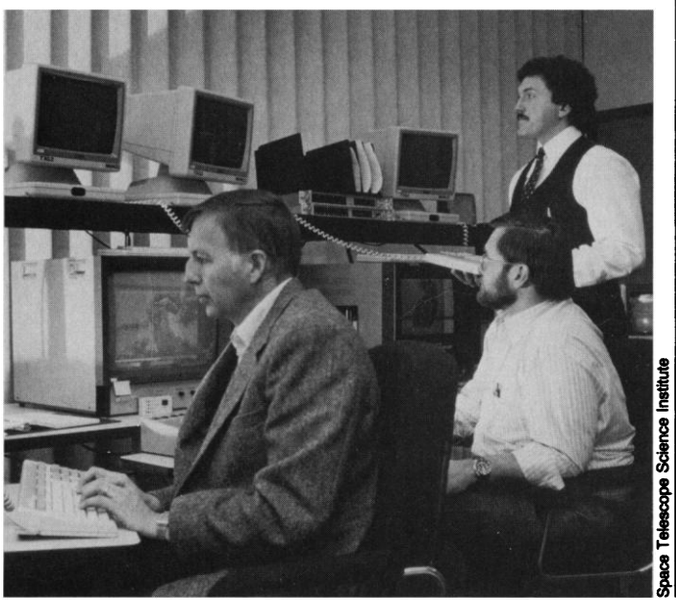
Modern programming practice calls for avoiding these traps with a style known as "rapid prototyping," which places much heavier emphasis on early input from the users of the software. "You're always looking at the issues with the users, trying to resolve them, trying to see if you can work things out," says James Weiss, data systems manager for Space Telescope at NASA headquarters. In this style the software itself evolves through a process of iteration. For example, a programming team might show the users a rough piece of software thrown together just to illustrate the basic ideas—"Is this what you want?"—and then refine the concepts based on the feedback. Then they would show it to the users again. And again.

Obviously, this process is labor-intensive, expensive, and time-consuming. And to project managers constantly working under the pressures of time and budget, it doubtless looks like a pain in the neck. But as the saga of SOGS makes all too clear, it is nowhere near as troublesome—or as expensive—as a textbook perfect piece of computer code that solves the wrong problem. Indeed, study after study has shown that getting the users involved early actually saves money in the long run.

If software methodology was a major source of SOGS' difficulties, however, it was by no means the only source. Even more fundamental was the fact that few people at NASA were even thinking about telescope operations in the early years.

To begin with, the Space Telescope project as a whole was saddled with a management structure that can only be described as Byzantine. Responsibility was divided between two contractors, Lockheed and Perkin-Elmer, and two NASA centers, Goddard and the Marshall Space Flight Center in Huntsville, Alabama. Communications were often abysmal and intercenter rivalries were rampant. "It was a system set up for trouble," says Schreier.

At the hardware level the chaos at the top was reflected in a raft of independently developed scientific instruments and onboard computers, none of which were well coordinated with the others. Indeed, the presumption was that any such problems would be taken care of later in the software. In the programming community this style of pasting the software onto the hardware as an afterthought is sometimes called the "appliqué" approach. And it, too, is widely con-



Space Telescope Science Institute

SOGS in action. *Space Telescope operators at the science institute rehearse with the Observer Support System, a component of SOGS that will allow them to issue commands and to monitor the telescope's performance. This room is where visiting astronomers will come to interact with the telescope in real time while it makes their observations.*

sidered a dismal methodology.

"To command a single instrument [on Space Telescope]," says Schreier, "you have to know how that instrument works, how its internal computer works, how the spacecraft computers work, and how two or three different pieces of the ground system work—each developed by different contractors, and each operated by a different group. And all that has to be in the software. So it's no wonder that planning and scheduling is difficult."

To take just one example, he says, Space Telescope is designed to have several instruments working at once, to maximize data return. But because no one had worried about the interactions, it turns out that different instruments can sometimes garble each other's data. So now the software has one more thing to consider in scheduling data readouts. "We can remedy the software to take care of this," says Schreier, "But problems always come up. And without a systems engineering overview from the beginning, you get into trouble.

So is SOGS fixed now?

Maybe. With TRW's help, the institute has spent the past several years beating the system into shape. Indeed, in 1987 NASA finally gave the science institute full responsibility for SOGS. Most institute researchers now seem to believe that the worst problems are under control: SOGS may not be perfect by December, but the system should at least be usable. "SOGS will support flight operations," says Schreier.

In the critical area of scheduling, for example, a combination of faster computers and better algorithms has gotten the planning time on SOGS down to where it is roughly equal to real time on the telescope. Additional fixes are under way that should get the system down close to one-third real time—the optimum—by the day of launch.

More generally, the institute has surrounded SOGS with various pieces of new software to fill in functions that the original specifications left out. A prime example is a system known as Spike, which will use a variety of artificial intelligence techniques to quickly rough in the telescope's schedule on a monthly and weekly basis. In effect, it will provide the road map for the trip as a whole, thus freeing up SOGS to do the minute-by-minute steering with its fine-grained scheduling algorithm. The result: a better division of labor and more efficient scheduling.

On the other hand, such progress has come at a price. SOGS now consists of about 1 million lines of programming code, roughly ten times larger than originally estimated. Its overall cost has more than doubled, from $30 million in the original contract to roughly $70 million. And the institute has had to take on some 50 programmers, including 9 from TRW, thus increasing its payroll by more than 20%.

Meanwhile, however, there remains a more important question: has NASA itself learned anything from SOGS—or from the Space Telescope experience as a whole?

Again, maybe. On the project management front, there are a lot of cynical observers who remain to be convinced that the agency has learned anything at all. Yet NASA officials are at least using the right words. "Don't do multiple center projects unless you absolutely have to," says deputy associate administrator for space science Samuel W. Keller, echoing a sentiment often heard around agency headquarters these days. In some cases the project is just too big for a single center, with the most obvious example being the $16-billion space station. But even there, Keller considers it significant that NASA gave overall control of the project to a new central management team located in the Washington area.

On the software front, meanwhile, Weiss and a few allies within the agency report some success in nudging people toward more modern methodologies. The space station program in particular has been quite responsive, he says. There are a lot of skeptics in this area, too. But there is also a considerable impetus for change: in both NASA and Pentagon contracting, the cost of the old-line approach is becoming all too apparent. Indeed, it has become a real sore point in the computer community.

"It's the methodology that got us to Apollo and Skylab," says Weiss. "But it's not getting us to the 1990s. The needs are more complex and the problems are more complex."

"SOGS," he says, "is probably the last example of the old system."

■ M. MITCHELL WALDROP

Scheduling the Stars

To see why it is so hard to schedule observations on the Hubble Space Telescope, consider just a few of the constraints that such a schedule has to satisfy.

For example, aiming Space Telescope at a new object is not just a matter of telling it to swing into position. The telescope's precision pointing system is capable of rotating the spacecraft only about as fast as the minute hand on a watch. So the schedule has to allocate the time for that. And even when the telescope does finally reach the target, the schedule still has to allow it an average of 15 minutes to find and lock onto its guide stars: the celestial reference points that the telescope will use to keep its pointing accurate to a fraction of an arc second. Only then can the observations begin.

Of course, in arranging all this, the schedule must also take into account certain forbidden zones where the telescope must not point at all. It obviously has to avoid the sun, since that would risk the instant destruction of instruments designed for very faint, very distant objects. It also needs to avoid the moon or the limb of the earth—not because of any danger in this case, but because scattered light could ruin the image. And it definitely should not point directly forward along its orbit, because the barrel of the telescope would function like a scoop for the rarefied atomic oxygen that exists at that altitude. Not only would the impact of those highly reactive atoms produce a glow that would fog the images, but they would quickly corrode the telescope's optical surfaces.

Meanwhile, the schedule must also warn the telescope to cease observations when it passes through the South Atlantic Anomaly, a region just east of Brazil where the earth's radiation belts dip low enough to spoil the data collection with random electronic noise. Space Telescope will spend about 15% of its time in this region.

And finally, the schedule must take into account the fact that Space Telescope orbits the earth once every 100 minutes. Among other things, this means that there is an excellent chance that the telescope will frequently find its target blocked by the earth as its orbit carries it around. And since the telescope cannot rotate fast enough to swing to another target and back again during the 40 minutes or so of blockage, all it can do is stay in position until the target is clear again—meanwhile pointing uselessly at the ground.

The best bet is for the scheduling software to try for targets in a direction perpendicular to the orbit, where blockage is not a problem. But given the other constraints, this is not always possible. Indeed, on the average, Space Telescope will spend no more than 35% of its time actually taking data—a fraction no better than a ground-based telescope that has to contend with daylight, moonlight, and clouds.

■ M.M.W.