MARS SCIENCE LAB:

THE CHALLENGE OF COMPLEXITY

BY RICHARD COOK

One of NASA's great strengths over the past fifty years has been our ability to execute complex, one-of-a-kind projects. In some cases, we have literally written the book on how to carry out programs with difficult technological, scientific, or programmatic objectives. It is somewhat surprising, therefore, that we've had significant problems in the past few years with some highly visible, complex projects. I work on one of those projects, the Mars Science Laboratory (MSL).

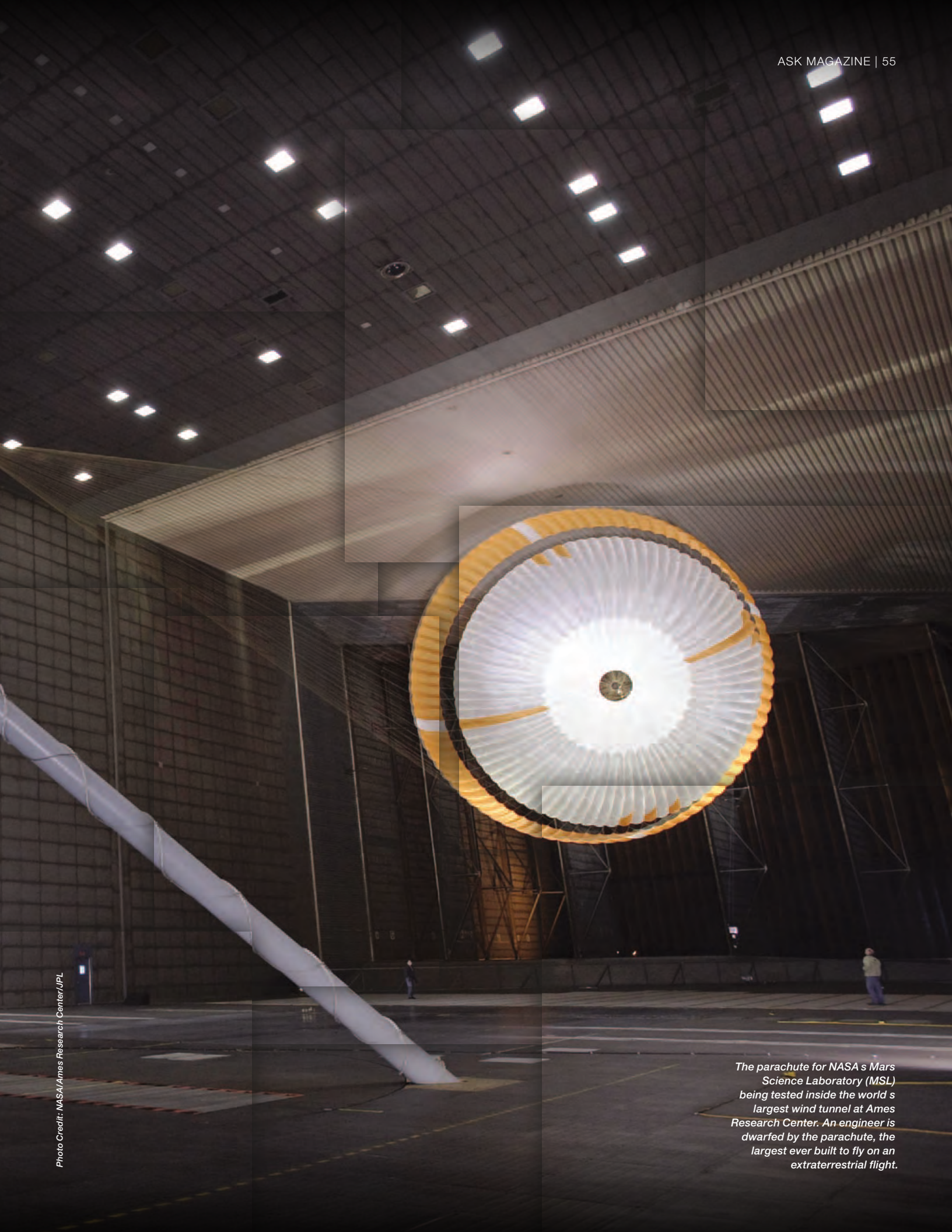


Photo Credit: NASA/Ames Research Center/JPL

The parachute for NASA s Mars Science Laboratory (MSL) being tested inside the world s largest wind tunnel at Ames Research Center. An engineer is dwarfed by the parachute, the largest ever built to fly on an extraterrestrial flight.

Image Credit: NASA/JPL-Caltech

Artist s concept of the Mars Science Laboratory in Martian terrain.

MSL is the next major step forward in NASA's Mars Exploration Program and will address key questions about the past and current habitability of Mars. The project is also developing critical new technology for landing on Mars, acquiring and processing surface samples, and conducting long-duration surface operations. This is probably the most complex planetary mission that NASA has ever attempted. As a result, it has stressed our implementation processes, our technology, our engineering capabilities, and our people. Although the project hasn't launched yet, it has been extraordinarily useful in one regard: demonstrating the challenges of managing complexity on large-scale programs.

So, what is complexity? The word is frequently thrown around as a sort of synonym for "difficult." But it is more than that. Paraphrasing Webster, "Complexity is the quality of being intricately combined." The characteristic that separates complex projects from merely difficult ones is the number of interconnected elements that are tied either technically or programmatically. Flagship efforts are becoming increasingly difficult *and* complex. Increased complexity is a primary cause for the challenges we've experienced. The MSL development experience is rich with examples where our ability (or inability) to effectively manage complexity has provided valuable lessons.

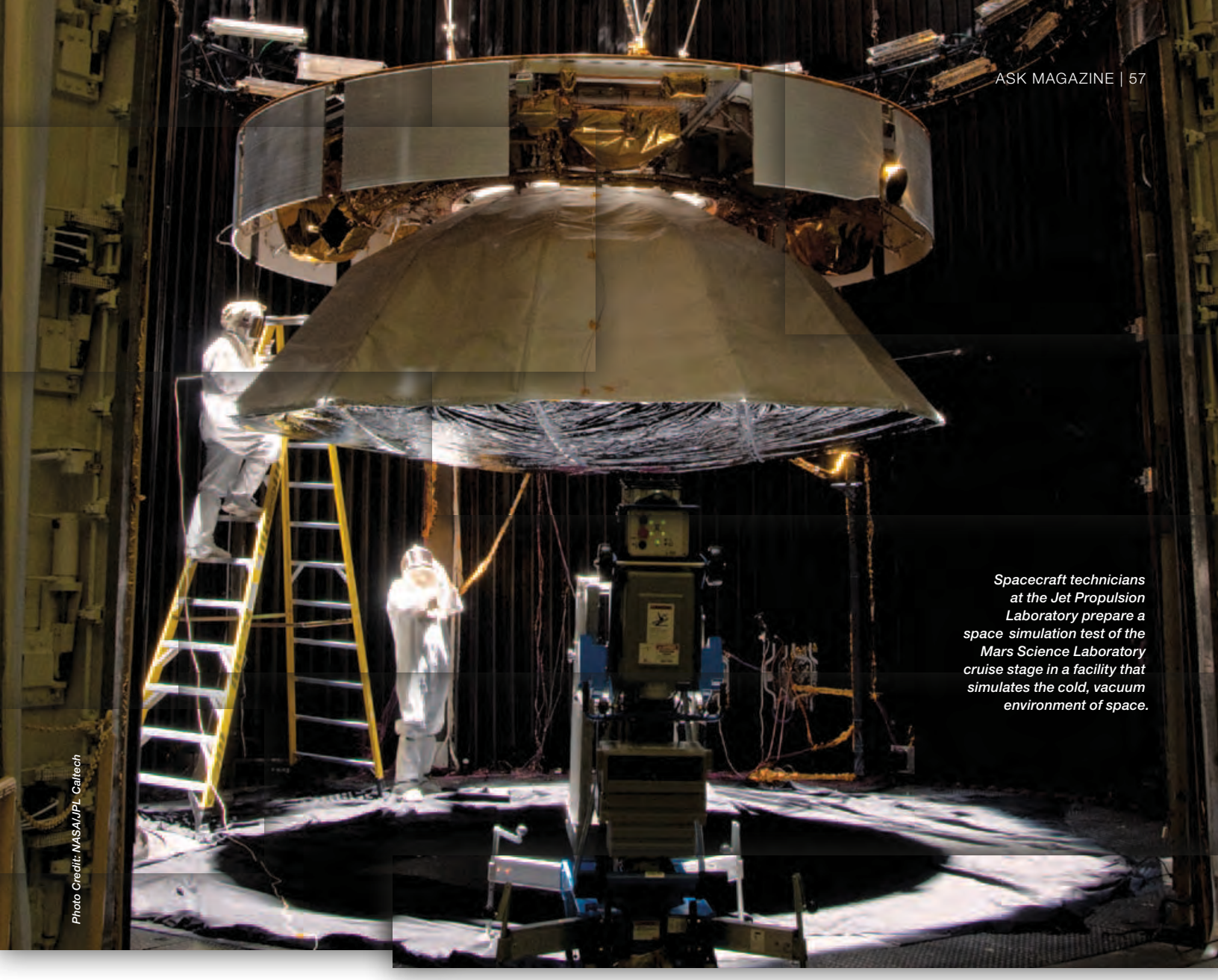
At the recent Project Management (PM) Challenge in Long Beach, California, I gave a presentation on those lessons across domains including technology infusion, margin management, schedule planning and oversight, and the role of external reviews. Given space limitations here, I will focus on the connections between system architecture and complexity.

Defining the right system architecture—the top-level structural and behavioral relationships between parts of a system—is critical to managing complexity. So what makes the "right" system architecture? The easiest answer is, the one that is as simple as possible but no simpler; the one with the most "separation" between elements; the one with the simplest interfaces, the most functional independence, the least reliance on those one-size-fits-all solutions that drive custom-interface accommodation. Greater complexity and interaction mean increased potential for problems and increased difficulty in testing to discover them.

Unfortunately, a number of factors frequently undermine system architecture simplicity. Examples include technology limitations and complexity, mass/volume constraints, cost, and the use of heritage hardware. I could mention several examples of MSL handling systems complexity well, but I'll start with one where we didn't.

THE CHARACTERISTIC THAT SEPARATES COMPLEX PROJECTS FROM MERELY DIFFICULT ONES IS THE NUMBER OF INTERCONNECTED ELEMENTS THAT ARE TIED EITHER TECHNICALLY OR PROGRAMMATICALLY.

We inherited several key aspects of the MSL architecture from the Mars Exploration Rover program. One example was having the rover's avionics control the entire mission from launch through landing. This architecture was adopted for MSL despite the fundamentally different functions for launch; cruise; entry, descent, and landing (EDL); and rover operations. The intent was to take advantage of the core elements of the rover avionics (the processor, the power converters) to perform cruise and EDL functions. Adding additional boxes outside the rover required accepting the associated cost, schedule, and mass impacts. The problem with this architecture is that it significantly increased the complexity of the design by functionally integrating the rover and cruise/EDL systems. The cruise/EDL system could not be designed and tested independently from the rover because it was an integrated system.



Spacecraft technicians at the Jet Propulsion Laboratory prepare a space simulation test of the Mars Science Laboratory cruise stage in a facility that simulates the cold, vacuum environment of space.

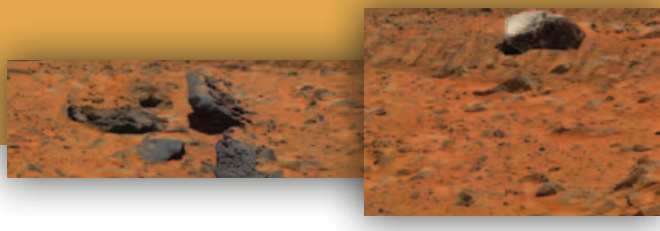
Photo Credit: NASA/JPL Caltech

So why was this choice made? We did an early concept study of a "smart" descent stage. The idea was to put enough avionics on the descent stage to control the vehicle during cruise and EDL (the rover would be along for the ride). The primary reason we didn't choose that approach is that we have a tendency in the early phases of a project to base system-design choices on box-level factors. Because the cost, schedule, and design of boxes can be coarsely quantified, it is simpler to factor them into design choices. Less apparent factors like the amount of input/output a box requires, the interface complexity, fault-protection implications, and verification challenges—all byproducts of system complexity—are difficult to quantify and factor into system decisions. These items typically don't manifest themselves until later in the development cycle and are frequently the source of significant cost growth. By not adequately factoring this cost-growth risk into the system trade, we ended up with a design with the fewest number of boxes rather than the least complex architecture.

Another driver toward functional over-integration is the pervasive impact electronics technology is having on our core systems. Unlike the world of thirty years ago, virtually all electronics we use today come from a commercial sector with different and diverse technology drivers, not just space applications. The increased functionality possible with high-density field-programmable gate arrays (FPGAs), low-voltage parts, and high-speed bus architectures are dramatic and enabling, but they increase complexity enormously. The pressure to have "less" hardware and depend more on software results in highly integrated and highly complex designs.

One associated pitfall is that we don't approach the incorporation of these new devices into our systems with the same degree of rigor we treat other types of technology. That may partly be due to the perceived maturity of the commercial components. We frequently have trouble with parts that have a commercial track record but haven't been through a full flight-qualification program. A good success story on MSL was our efforts to "mature" high-density, radiation-tolerant FPGAs.

THE PRESSURE TO HAVE "LESS" HARDWARE AND DEPEND MORE ON SOFTWARE RESULTS IN HIGHLY INTEGRATED AND HIGHLY COMPLEX DESIGNS.



The Mars Reconnaissance Orbiter and other programs had experienced a series of problems with less dense parts, so MSL adopted an aggressive program to establish acceptable design guidelines, packaging/rework approaches, and thermal control/qualification strategies. The result was that the project did not experience significant FPGA technology issues during the build/test campaign.[1]

The FPGA challenges we did have were associated with the design complexity caused by functional over-integration. The large number of logic gates available in modern FPGAs allows many functions to be combined into a single component. This does complicate the design effort, although some parts of the FPGA "code" can be developed by parallel teams. The verification and validation effort, however, grows dramatically because so much functionality is combined. Our test methods don't really support ways of performing rapid, parallel testing of a single, highly integrated element. A long serial-test program is difficult to manage, is brittle to changes and problems, and can be inappropriately curtailed if schedule pressure mounts. A design based on a larger number of simpler elements would permit parallel component testing and (with appropriate interface definition) simpler system testing as well.

Fault tolerance is another system-architecture driver that can significantly affect complexity. Inappropriate evaluation of local-versus-system fault tolerance can dramatically increase complexity without necessarily improving overall reliability. An example from MSL was the incorporation of partial redundancy in the core rover avionics. The mass and volume of the avionics are major drivers on both the rover configuration and the required capabilities of the entry, descent, and landing system. Heavier or larger avionics increase EDL system risk by reducing control-system performance margins or increasing landing velocity and loads.

Intrinsically, however, avionics fault tolerance is provided by adding redundant boxes with some degree of cross-strapping. (Cross-strapping permits redundant boxes to work with other redundant elements in the system architecture.) On MSL, the project took an intermediate position of incorporating some partial avionics redundancy to mitigate box-level failures while not driving EDL risk adversely. Unfortunately, the resulting system is neither fish nor fowl from a complexity perspective. By having a combination of single-string and redundant elements, the resulting fault-containment architecture is more complex and more difficult to design, analyze, and verify than either a single-string or fully redundant design. The marginal increase in reliability associated with the partial redundancy may not have been worth the increased complexity.

These are just a few examples of the drivers that can push a system architecture toward increased complexity. Potential institutional mitigations could include additional training to increase our systems engineering expertise on both the sources and consequences of architectural choices. Additional efforts can also be made to rigorously review system architecture choices to understand the long-term implications. Upgrading our cost and schedule estimation processes to capture the impact of complexity on cost and schedule risk would also be very useful.

From the perspective of an individual project manager, establishing simplicity as a programmatic goal is both a symbolic and a real step toward managing development risk. This is particularly imperative for projects with profound technical and engineering challenges. Intrinsically difficult missions like MSL are made much more challenging if managing complexity gets inadequate attention. Policy direction advocating simplicity is a useful first step to keeping complexity contained. ●

RICHARD COOK is the deputy project manager of the Mars Science Laboratory at the Jet Propulsion Laboratory. He is a veteran of NASA's Mars Exploration Program, having held key roles on Mars Pathfinder, Mars rovers Spirit and Opportunity, and Mars Surveyor '98.

1. We did have FPGA problems associated with design complexity (we tried to put too much functionality into a given part), which led to very long delivery delays and test-program challenges. The fundamental part technology worked, however.