Instar

Doing Things Right in Space Programs

This article is part of a series started in January, 2000. My intent is to share a philosophy and ideas for how to increase the chances of success in space missions while also reducing total cost. Once these articles are completed, I plan to assemble them into a book. Please send comments to me at Tom.Sarafin@instarengineering.com.

Ten Principles for Doing Things Right in Space Programs

1. **Adopt the right attitude**
2. **Invest in knowledge and understanding**
3. **Instill ownership and responsibility**
4. **Constantly seek ways to improve teamwork**
5. **Follow a sound engineering approach**
6. **Reduce total cost through good engineering**
7. **Keep everything as simple as possible**
8. **Establish an effective quality system that involves everyone**
9. **Be willing to accept risks, but only those you truly understand**
10. **Make sure everyone has enough time, resources, and freedom to do things right**

Article #13

Follow a Sound Engineering Approach

Part 2: Developing Requirements

March, 2001

Tom Sarafin

President, Instar Engineering and Consulting, Inc.

6901 S. Pierce St., Suite 384, Littleton, CO 80128 • (303) 973-2316 • instarengineering.com

Last month's article gave an overview of the engineering process for developing a system. Although I touched on the subject of requirements definition, I'd like to spend more time on it for two reasons: First, it's so important. If we don't do a good job identifying, deriving, and specifying requirements, the rest of what we do doesn't matter

much. Second, we tend not to do a good job with requirements in the space industry. In some cases, we can't even agree on the types of things that should be requirements.

In the eighth article in this series, I offered a definition of a product requirement: something the product must be able to do, or some characteristic the product must have. I went on to say that a requirement comes either from a function or a constraint. If you look through a specification on your program, chances are you'll find a great deal of "requirements" that do not meet this definition. They specify not something the product must be able to do, but something the product developer must do to ensure or demonstrate the product does what it's supposed to do (verification).

In Article #8, I argued against having such "requirements" in the specification because they took responsibility away from the developer. If you didn't buy that argument, let me pose another: Cluttering up a product specification with activities and criteria for verification and quality assurance often cause the true requirements to be lost. The people developing the product often lose sight of what they are developing because they're so busy responding to the checklist of required analyses, inspections, and tests.

Let me give you an example. As I've probably mentioned somewhere in this collection of articles, I entered the space industry fresh out of college with a bachelor's degree in civil engineering. Quite a few organizations recruit civil engineers to design and analyze structures for launch and space vehicles because the typical civil engineering curriculum (with a structures option) offers more structures education than any other four-year degree that I know of. As a stress analyst at Martin Marietta, I spent my first five or six years thinking the structural requirements were simply to have positive margins of safety under specified factors of safety. Although I had a surface-level understanding of the space mission, I did not understand how the structure actually related to it, and I had no interest in finding out. I was content dealing with two requirements: (1) no yielding (actually no permanent detrimental deformation) under design yield loads and (2) no rupture or collapse at design ultimate loads.[1]

We designed our spacecraft structure under these rules, and then we built it. I was assigned the responsibility for testing it. After many long months of planning, I felt ready to do the test. To make sure the test would be acceptable to our customer, I invited customer representatives out for a test design review. They brought their technical consultants.

I presented the test objectives, the concept and configuration, and the load cases. Then I got to the subject of success criteria, which I presented as follows: "No permanent deformation under test yield loads and no rupture or collapse at test ultimate loads."

I was about to move on, but I was stopped by one of the customer's consultants: "How will you detect 'permanent detrimental deformation?'"

I gave the standard, canned response: "Visual inspection." And I started to flip to the next viewgraph.

But he wasn't satisfied: "How do you know you'll be able to see it?"

---

[1] For those not familiar with the terminology: Structural failure can be catastrophic rupture or collapse *(ultimate failure*) or permanent deformation that is detrimental to the mission (*yield failure*). Structures are designed to ultimate and yield *factors of safety* (*FS*), with the former being the higher factor. The *design ultimate* and *design yield loads* are the *limit load*, *p*, which is the highest load expected at a target probability, multiplied by the ultimate and yield factors of safety, respectively. These design loads must not exceed the corresponding statistically based *allowable loads*, *P*, which are calculated from material properties and geometry. The ratio of the allowable load to the design load should be at least one; if you subtract one from it, you get the *margin of safety*, *MS*, which is a measure of strength in excess of that required by the design criteria: $MS = P / [FS(p)] - 1$.

Now, so ingrained was the standard way of doing things in my field that my first response was frustration. Who invited this guy anyway? But the more he poked, the more I began to realize how valid that question was. How did I know I could see it? The question began to grow uncontrollably: What was I looking for? Where was I looking for it? How much permanent deformation would be detrimental? Detrimental to what?

It began to dawn on me that the structure wasn't there just to hold things together; it had to hold things together well enough for the mission to succeed. And I realized I didn't know how well that was. I didn't know the structure's requirements.

I took an action item and then went immediately for guidance to the lead systems engineer. "How much permanent deformation can the structure withstand before it would be detrimental to the mission?"

"How would I know?" he replied.

I went to the lead structural design engineer and got the same response.

That sickening feeling of doom began to overwhelm me. We had designed and built a jillion-dollar structure for a gajillion-dollar spacecraft—and no one understood its requirements.

Fortunately, we recovered. The systems engineer, the designer, and I worked together to identify allowable permanent deformation in various parts of the structure to ensure the mission would be successful. I then found ways to measure deformation accurately enough to judge success, and our structure (again, fortunately!) passed the test.

Hopefully, you are beginning to see, as I have, that our typical process of defining and communicating requirements is flawed. We assemble a mind-boggling amount of specifications, standards, and criteria, and we then impose them on new programs regardless of the mission or product. In dealing with all these "requirements," engineers seldom have time to pursue a sound process of identifying the true requirements, which are related to functions and constraints.

All true requirements stem from functions and constraints. Environmental requirements typically stem from both: the need to function at some level despite the constraint of an environment. A well-written environmental requirement does not say “shall be exposed to …” or even “shall survive …” or “shall withstand …” A well-written environmental requirement specifies the product shall <u>be able to function as needed</u> during or after exposure to the environment.

To see how requirements should be defined, consider a system we are all familiar with: a hot-water system for a house. The objective, of course, is simply to provide hot water to convenient places in the house. Given that cold water is readily available, we know our system will have to store the water and heat it somehow. We can describe these and other functions in the form of a *functional flow block diagram* (Fig. 13-1).

The next step is to devise a concept for the system. Let's say we decide to tap into the water main, run a pipe into the house to a pressurized tank, heat the stored water with a gas flame, run another pipe system from this tank to several locations in the house, put in valves at these locations to dispense and shut off the water, and then install another pipe system to take the waste water to the sewage line.

Now that we have a concept in mind, there are a bunch of requirements we must identify. The system must be pressurized—how much pressure will we need? Will the pressure at the water main be enough, or do we need a pump? The system is constrained by its interfaces with the water main, the gas line, the sewage line, the clothes-washing machine, and the users; we must identify those constraints. We'll need some way to measure water temperature at the tank and have the system use that information to automatically turn the gas on and off. The users will want to control temperature, so

we'll need to run more pipes that deliver cold water, and the valves must allow the users to control the mix of hot and cold water.

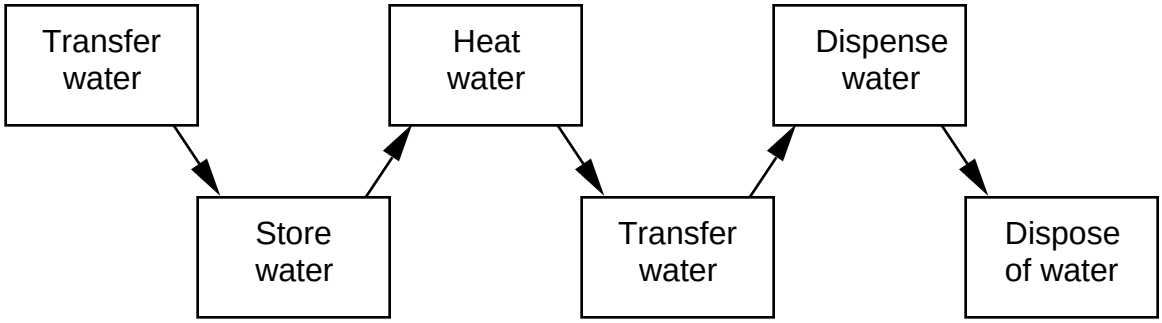

Fig. 13-1. Functional Flow Block Diagram for a Hot-Water System. Each block describes a function and is thus the source of one or more requirements.

One of the requirements for the pipe system is to contain water under the highest pressure it will see over the intended life of the system. But one of the requirements is not to sustain three times that pressure—we may decide to design and test it to three times the pressure as a way of becoming confident in the system, but that's not the true requirement. We may decide the pipe material must pass dye-penetrant inspection, but this is not a true requirement. It's something that may be worthwhile, given the way the pipe material is presently made.

Establishing standards and criteria is a smart thing to do, but mixing them with the system's requirements causes the engineers developing the system to lose sight of the true requirements. Over time, we may find a better system concept for providing hot water to homes. If we pass down our standards and criteria as requirements to this new system, not only do we potentially over-penalize the system, we may develop a system that doesn't work because our engineers don't identify its true requirements.

There are three types of requirements: functional requirements, performance requirements, and constraints. A *functional requirement* defines what must be done. For example,

- "The telescope structure shall maintain optical alignment throughout the mission."

This is an example of a requirement we might put in a system specification. Eventually, we'll need to quantify the requirement. A *performance* (or *operational*) *requirement* specifies how well something must be done:

- "The telescope structure shall be able to sustain launch (or a specified environment) without suffering a permanent change of greater than 10 microns in the relative spacing between the primary mirror and the secondary mirror."

If we are specifying the requirements for this telescope to a contractor, we certainly can't specify the above performance requirement. We can specify the needed image quality—which would certainly be degraded by relative motion of the primary and secondary mirrors—but it's up to people developing the telescope to define the needed positional stability because that will depend on their design.

A *constraint*, the third type of requirement, limits resources or physical characteristics:

- "The telescope structure shall weigh no more than 100 lb."

Let's pursue the telescope example. A *verification plan* describes how we will show our products meet requirements:

- "We will verify strength of the telescope structure by analysis, according to the criteria defined in document xxx."

Finally, *verification criteria* are the ground rules the defined methods of verification must satisfy:

- "For analysis alone to verify strength, the calculated stresses corresponding to design yield loads (limit loads multiplied by a 1.6 yield factor of safety) must not exceed the material's A-basis (99% probability) allowable yield stress. In addition, the calculated stresses corresponding to design ultimate loads (limit loads multiplied by a 2.0 ultimate factor of safety) must not exceed the material's A-basis allowable ultimate stress. Finally, analysis must show the structure can withstand design ultimate loads without buckling."

In the example I gave earlier, in which no one on our program understood the true requirement for structural alignment, we structural engineers worked according to verification criteria similar to those defined above, except our factors of safety were lower because we planned to test the structure. Criteria such as these were the only "requirements" we had. Most likely, on your program, the structural engineers have similar "requirements."

But let's return to the telescope example. The true requirement is for this hypothetical structure to maintain spacing of the mirrors to within ten microns. (We'll look later at how someone may have identified that requirement.) To verify the requirement, "the calculated stresses corresponding to design yield loads must not exceed the material's A-basis allowable yield stress."

As written, there is something wrong with this verification criterion. We need to ask ourselves whether satisfying the criteria would assure us that the structure will meet its requirement. The answer is no because there is no clear connection between the requirement and the criterion. If the requirement were for the structure to maintain five microns instead of ten, it's not apparent how the verification criteria would change.

You can't design a structure to have <u>zero</u> permanent deformation from applied loads. In material testing, you can't determine precisely the stress above which nonzero permanent deformation would first appear. A material's advertised yield stress is <u>not</u> that stress. The *yield stress* for most metal alloys is traditionally defined as the stress that causes permanent strain of 0.002 (0.2% permanent deformation), as shown in Fig. 13-2.

A telescope could not maintain alignment to within 10 microns if its structure suffered a permanent deformation of 0.2% of its length. The telescope would have to be no more than half a centimeter long! For designing an actual telescope, we probably would need to determine a lower allowable stress that is consistent with the 10-micron requirement, which we do by testing the material.[2]

[2] In addition, we should change the criteria from verification criteria to design criteria and plan to verify the alignment requirement by test. Stress analysis is seldom perceptive enough or reliable enough to be the sole method of verification for alignment-critical structures.

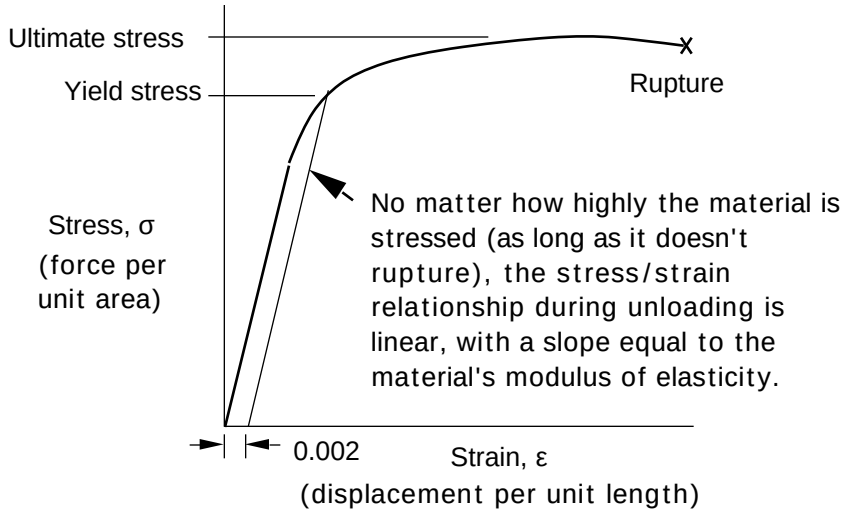

Fig. 13-2. Typical Stress/Strain Curve for a Ductile Material. The yield stress is the stress that would cause a permanent deformation of 0.2% of the specimen's length.

Remember: to design something that will work, we must quantify its performance and then design it to perform that well. The traditional criteria used to design space structures evolved from the design of aircraft, missiles, and launch vehicles, for which alignment is usually not so critical. How many other "requirements" are out there that don't apply to the system element we are designing and that distract us from ever identifying the true requirements? With a sound process, we'll find them.

Allocating Requirements

Let's look now at how we would go about identifying requirements such as the ten-micron one used in the above telescope example. In doing so, we'll continue with that example.

Let's say the objective of our space mission is to provide optical images of the Earth's surface, with the ability to detect objects as small as five meters in diameter. End-product images with five-meter resolution is thus the top-level system requirement. We now establish a concept: a satellite with a telescope, data recorded digitally, data transmitted to a ground station, data processed to produce images electronically and on paper. It's our job to design this system, given many variables. Among other things, we'll have to decide on orbital altitude, focal length, telescope length, number and arrangement of mirrors (photon path), aperture, sensing frequency, data-handling approach, and communications data rate. Many of these parameters will affect the end-product image resolution.

One of the things we must do to engineer this system is identify all the things in it that could affect image resolution. We then allocate pieces of the five-meter requirement to those elements as we would cut up a pie. The logical way to do this is to start at the system level and work down through the system elements.

In this example, our system is composed of a space system (spacecraft) and a ground system. If the way we will receive and process data on the ground could affect the image, then we must give the ground system a piece of the pie, and our spacecraft must be able to provide resolution that is better than five meters. The spacecraft consists of a payload (telescope and camera assembly) and a bus, which is the set of supporting subsystems. Obviously the payload will affect the image, but maybe the way the bus records,

encrypts, and transmits data can affect it also. If so, the bus must have a piece of the pie, which means the payload will have to better than the total requirement for the spacecraft.

We allocate requirements this way throughout the system. It is this process that tells the engineers how well they have to design their parts of the system to ensure the end product has 5-meter resolution. Figure 13-3 illustrates this allocation down to the level of the telescope structure.

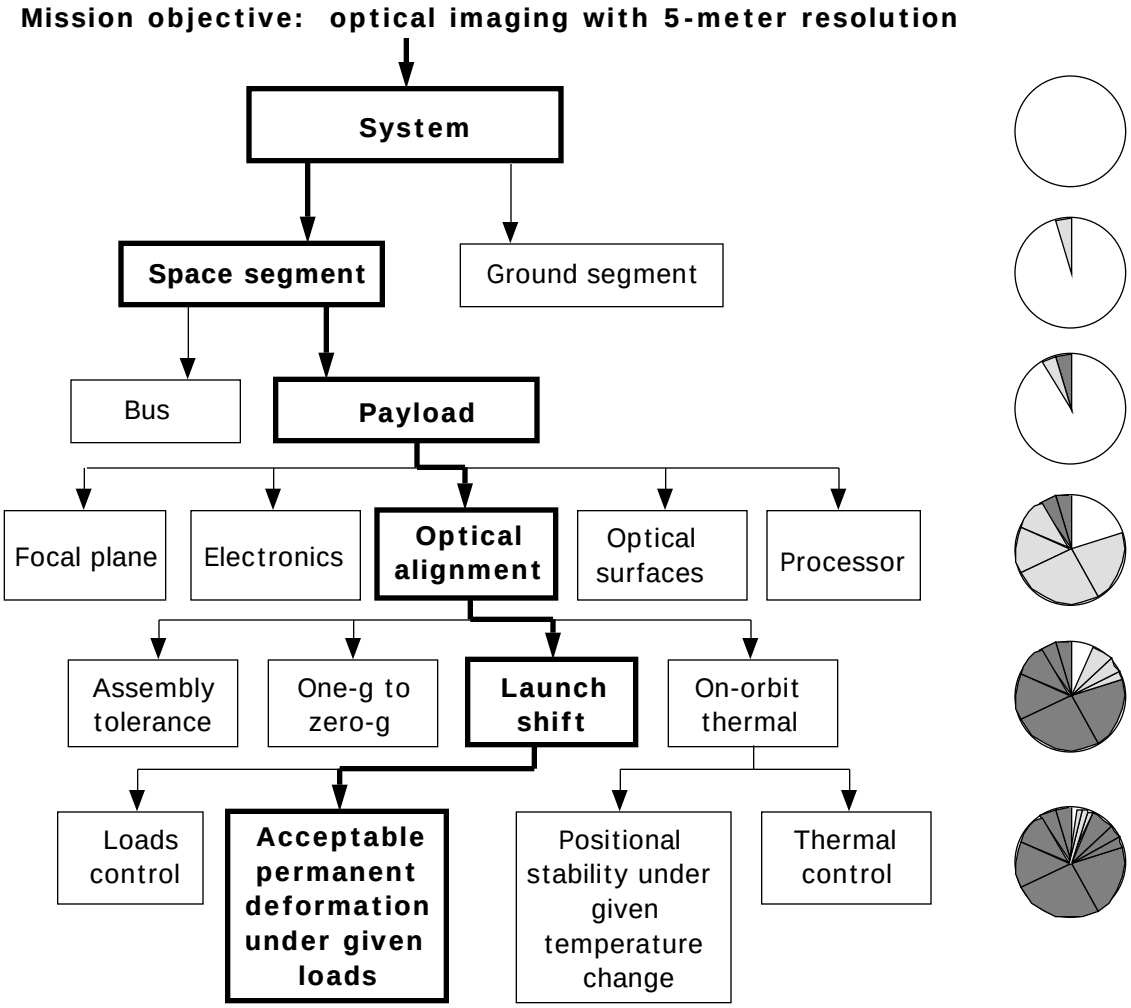

Fig. 13-3. Example of a Functional Requirement Flowed Down to the Structure.
At each lower level, we subdivide a higher-level allocation. Pieces of the pie get smaller and smaller. In this figure, only the allocations in bold are subdivided (*decomposed*), whereas for an actual system, most of the others must be decomposed as well. This figure is intended only to illustrate the process, not to identify all the things that would actually affect image resolution.

When we allocate requirements as indicated in Fig. 13-3, we assume it is reasonably likely for each element of the system to just meet its requirement without margin. In such a case, allocating with separate pieces of the pie ensure the system will work. Some allocations, though, may be for random error; for our hypothetical telescope, such allocations would be for assembly tolerance and tolerances on optical surfaces. Recognizing that it is extremely unlikely for each of several random tolerances to come in at their extremes, we may decide probability-based allocation is more appropriate. For

example, we may define an allocation, or piece of the pie, for the total effect of random tolerances, equal to the square root of the sum of the squares of those tolerances.

In our example, the allocation for the bold box in the lowest level is a 10-micron change in spacing between the primary and secondary mirrors. When working from the top down, as described in Fig. 13-3, this allocation comes from a study: Given the nominal dimensions of the telescope, how much spacing error would cause the adverse effect on image quality that we are willing to allocate to the structure for permanent deformation caused by launch? For a cost-effective system, however, we must also look at the allocation from another perspective: Given the materials and form of construction for the telescope structure, how well can we expect the structure to maintain alignment? Without addressing this question, the allocations are selected arbitrarily, based solely on subdividing the pie, without regard to cost.

Let's say the structures group assesses the effects of launch and concludes the structure they're designing can easily maintain optical alignment to within 5 microns. Meanwhile, suppose the allocation for thermal control is to maintain temperature throughout the telescope plus or minus 3°C from room temperature. (Changing temperatures, of course, causes the structure to expand or contract, thus changing the spacing of the mirrors.) To maintain such a tight temperature range while the spacecraft passes in its orbit from full sunlight to Earth shading, the thermal-control group designs a system of heaters and thermistors that, although effective, is quite expensive. Let's further assume the thermal-control system would be half as expensive if the requirement was relaxed from ±3° to ±5°. In such a case, it would make sense to take some of the pie allocated for launch shift (say, from 10 microns down to 5 microns) and give it to thermal control (from ±3° to ±5°).

As you can see, requirements allocation is iterative. In a typical large program, a systems group allocates requirements to specialized groups without the knowledge needed to allocate cost-effectively. If you are a specialized engineer working within such a process, you need to recognize that, especially early in the program, the allocations are not firm. Don't let an unreasonable allocation drive cost. Present your case to the systems group and lobby for more. On the other hand, don't overly protect your turf and refuse to give up allocations that are too generous. That's not good for the program.

A good question, here, is at what level do we stop allocating? The telescope structure may be an assembly of many parts; do we need to decompose the 10-micron requirement down to the part level? The telescope will have an allocated weight; should we subdivide it down to an allowable weight for each bolt?

It's important to recognize why we allocate in the first place: to ensure the individually designed elements will work together as a system. We allocate down to the level we believe necessary to ensure the system will work. Requirements allocation is part of a quality system, part of the verification process. There is always a hand-off from what we call requirements to what we call verification. Your organization may choose to allocate internally within an extensive hierarchy of specifications, whereas mine may choose to allocate within a pyramid of verification plans and reports. If the engineering is thorough, the results are the same. Just remember to keep allocations flexible, in whatever form they appear.

Throughout the engineering process, we want to maintain requirements traceability. For complete traceability, we must be able to follow each requirement upstream and downstream. *Upstream traceability* for a requirement means we are able to trace it back to the system specification to find its source. Any requirement we cannot trace back to the system spec is an *orphan requirement*. All requirements should derive from the system spec. If we find an orphan, we either delete it (because it will drive cost) or modify the

system spec so that the requirement is no longer an orphan. *Downstream traceability* means we can trace the requirement downstream, through lower-level specifications, if applicable, all the way to a plan (or report) to see how it will be (or was) verified. Periodically tracing requirements downstream ensures we haven't overlooked any when allocating to lower levels, helps us generate a thorough verification plan, and guides us in the final step of documentation: verification (compliance) reports.

The next article in this series will look more closely at verification.

About the Author

Tom Sarafin has been involved in the space industry full time since 1979, at which time he graduated from The Ohio State University with a BS in civil engineering and took a job as a stress analyst at Martin Marietta Astronautics in Denver, Colorado. While at Martin, he was involved with design, analysis, verification planning, and testing on several spacecraft and launch vehicle programs. After contributing to the book *Space Mission Analysis and Design* [Larson and Wertz, editors, first edition published in 1991], he obtained management's support and funding at Martin Marietta for the development of a book on the interdisciplinary development of structures for space missions, and served as principal author and editor for 23 other authors. He left Martin Marietta in 1993 to complete this book, under the guidance of Dr. Wiley Larson at the U.S. Air Force Academy. The result of nearly four years work—*Spacecraft Structures and Mechanisms: From Concept to Launch*—was published in 1995 jointly by Microcosm, Inc., and Kluwer Academic Publishers.

In 1993, Mr. Sarafin formed his own company, Instar Engineering and Consulting, Inc. Once he finished his book, he began providing review and advice as a consultant to space programs. He also developed a short course based on his book and began teaching it throughout the industry. The course has been quite popular, and the business has grown. Now Instar offers a curriculum of courses taught by experienced engineers and continues to add to that curriculum.

Instar's Core Courses

- **DTR—Doing Things Right in Space Programs: A course for managers**
- **SDV—Doing Things Right in System Development and Verification**
- **USS—Understanding Spacecraft Systems**
- **SMS—Space-Mission Structures: From Concept to Launch**

Additional Instar Courses

- DASS—Design and Analysis of Space-Mission Structures
- USRV—Understanding Structural Requirements and Verification
- SPAD—Space Propulsion Analysis and Design
- OSPS—Overview of Space Propulsion Systems
- DAFJ—Design and Analysis of Fastened Joints
- APSIT—Avoiding Problems in Spacecraft Integration and Test
- GDT—Geometric Dimensioning and Tolerancing

Additional courses in work; customized versions available

For information on these courses, visit our website at instarengineering.com