CHAPTER EIGHTEEN

Peter Gluck on Building Spaceships

Is there any geeky kid in the United States who didn't grow up wanting to be a part of NASA? We sure did. That's why we were excited when we got in touch with Peter Gluck, who's been working on and running software teams for NASA's Jet Propulsion Laboratory (JPL) for more than 20 years. We wanted to know what it was like to work on a NASA team, and even more, what it was like to send his software into space, where it has to work and you can't just walk over and patch it.

Andrew: Tell us a little bit about your background, how you came to be at NASA, and the kinds of projects you've worked on.

Peter: I have a master's degree in aerospace engineering and a bachelor's in mathematics. I actually went on a field trip in fourth grade to the Jet Propulsion Laboratory here in Pasadena, and came home that day and said, "Mom, when I grow up, I want to work at JPL."

Andrew: So you've wanted to work with NASA since you were a little kid.

Peter: Yeah, so I was kind of focused as a youngster. Actually, at one point, I wanted to join the Air Force and fly jets, but my vision precluded that.

I went to Cal State Northridge, and while I was there I had a class with a girl whose father worked here. She got me an interview, and I've been here for 22 years. That's how I got started at NASA.

Andrew: Tell us a little bit about the projects you've done, especially in terms of the kind of software. We want to talk about teams in general, and the kinds of teams that build software in particular, and teams that work well together.

Peter: Teams can be a challenge. Getting people together that have the right mix of skills and the right mix of personalities can sometimes be a challenge.

I worked on many flight missions here at JPL for my entire career, from Magellan back in the 1980s—which was a Venus radar mapper—to a Mars observer, which made it to within three days of Mars and then disappeared. I worked on the TOPEX/Poseidon mission which was an Earth-orbiting oceanographic satellite, and then on the ground systems for the Cassini mission.

Deep Space 1 was my first leadership role where I led the flight systems control software team doing the Deep Space 1 mission for three years.

I was on Phoenix for the last four and a half years. I was the project software systems engineer, was responsible for the overall development of both the flight and ground software for an entire mission. I have since moved on to the advanced optical systems program at JPL.

Andrew: It sounds like you've had your hands full with a lot of really big projects over the years.

Peter: Yeah, a lot of different stuff, including a couple I didn't mention.

Andrew: So you said that getting teams together can be tough. What's tough about it? Specifically, what challenges have you seen in the past?

Peter: Well, there's two things. One is finding the right skill mix. You need people of different capabilities and different tastes, frankly. You can't have a team that's just a bunch of code hackers, because you need people who can also do the documentation and the systems engineering and can run the test program. You need people that can interface with the customer.

In any successful team, you're going to find that there's a variety of people; so one aspect is the technical skills of the team members.

The second aspect is the personalities. Some people are more withdrawn, some people are more outgoing, and to some extent you can't have it all skewed one way or the other. You need to find the right balance in personalities so that people can get along. You have too many people with strong personalities and they're going to lock horns.

On the other hand, if you have too many people who are meek, then you might not get the best ideas coming forward. You need to have a balance. And you need to have an environment where people feel safe in communicating their concerns and their ideas.

Jenny: Can you think of a time that stands out where in a team you worked on everything just meshed well, where you had an ideal mix of talent and personalities?

Peter: I guess I've been pretty fortunate that most of the teams that I've been a part of have been pretty good here; some better than others.

But in general, I guess I'd say the Deep Space 1 team was a pretty tight-knit team. Our software team, especially, was a very good team. I think we had a lot of good relationships. There were some members of the integration and test team that used to lock horns a bit, but overall I think that was a really good team.

The Phoenix team was also really a great team to be a part of. The software developers were all very sharp, mostly very willing to accommodate whatever the project needed.

Andrew: Do you think that locking horns is something that always tends to be good for a project, or is it something that can go either way?

Peter: I don't think it's always a terribly good thing, actually. I think that there needs to be a forum for discourse, but at the end of the day, you've got to have people that are willing to accept whatever the direction of the project is.

We have some very, very bright people here at JPL, much brighter than I am in many respects. I have seen those people just deadlock the entire team to the point where you can't get a solid direction one way or the other, because you've got these two or more individuals who simply won't budge from their righteousness, and that's counterproductive, I think.

To some extent, you need some healthy skepticism, some healthy exchange of differing viewpoints, but at the end of the day, there needs to be one decision to go down a certain path and everybody needs to be able to accept that and contribute for the good of the project.

Andrew: It sounds like if there has to be one decision, that's a role for a team leader, somebody who has not just the authority, but maybe the respect of the team to get everybody to move past the disagreement and accept the decision.

Jenny: Do think that has an effect on the ability for the team to gel? When you're leading a team, is there anything in particular that you do?

Peter: That is an interesting question.

The job of a team leader is an awful lot of diplomacy, at least on Phoenix. The last four years I've had to deal with three different large institutions: the University of Arizona, the Lockheed Martin Space Systems Company in Denver, and the Jet Propulsion Laboratory.

I had developers each independently developing their own software at all three institutions. And not to mention all of the contractual agreements and the management chains of command and all that. Diplomacy is a skill that you have to have in that kind of environment.

In terms of getting the teams to gel together, one of the things that I like to do is provide some team-building activities. Now, when you're scattered across time zones like this, it's a little difficult. And it may sound silly, but one of the things that I used with some success is fantasy football.

I invited everybody to join a fantasy football league, and I got about a dozen people to join it, not so much in the software team, but across the different teams that we worked with. There were members of the operations team, members of the integration and test team, members of the software team, and members of the systems engineering team all involved in this activity, and it's just a little something that you can talk about, and kind of humanizes everybody.

Otherwise, you end up where you're always dealing on a professional level. I think it's easy to kind of lose sight of the fact that there is a person back there who has feelings and does the same things that everybody does.

That's definitely one technique that I try to use.

The other thing that I try to do is I try to make myself available to anybody who has any questions or concerns. I have an open-door policy. Anyone can come and talk to me about anything that's concerning them at any time. If there's a serious problem, we'll try and get it resolved. Sometimes they just want to get something off their chests, and it doesn't go any further.

I think those are probably the two most important things in my toolbox.

Jenny: And what about in the way that you build software in general? I know that NASA is known for being pretty process-heavy. Do you want to talk about how the process itself impacts your team and how you feel, as somebody who's working on software that's scrutinized a lot?

Peter: Well, process-heavy, I don't know—it may very well be that the manned program is far more process-heavy than we are. Actually, to be honest, in the last 10 years, we've

become fairly process-oriented at JPL. And frankly, our industry partners have been well ahead of us in that area, although I think we're beginning to gain parity with them.

But we do have certain rules we have to follow, and we have a coding standard like any good software house would. We have a design life cycle that we follow. We have some institutional standards and practices.

One of the things that you have to do as the leader of the team is to make sure, A, that everybody is aware of the standards and processes; B, that they're trained on how to interpret them and use them; and C, that they actually get used so that there's an aspect of verification that goes with that.

But I haven't generally had problems with that in the past. People are generally pretty perceptive, at least the ones that I've worked with, to accepting the limitations of being in this environment.

Jenny: Can you take a few minutes and describe the process that you guys use to build software?

Peter: Well, I have to be a little careful here because I do have to be conscious of ITAR restrictions.

Andrew: What is ITAR?

Peter: International Trafficking and Arms Regulations. It's a federal law that prevents us from disclosing details about defense-related technologies; spacecraft are classified as a defense-related technology.

So I can't give you any specifics, but we use a standard waterfall-type life cycle here. Sometimes it's iterative, so you'll have a series of builds that you'll establish and you'll build up functionality, moving towards your end goal of a fully functional system.

You have your requirements, your design, your integration phases. People are pretty familiar with that here. There have been new people that come in and become familiar with it very quickly.

Andrew: I'm kind of having a bit of an almost disappointing "I put my pants on one leg at a time" moment. I always imagined that NASA had some incredible, intricate way of building things. But it sounds like you guys just do the same things that everybody else does, except you need to be a lot more careful and maybe rigorous about it. And I guess it's because your software has to work millions of miles away, where there's no you way you can get to it.

Peter: That's true.

Andrew: How do you cope with that?

Peter: A lot of testing. We do a lot of testing, and we have a lot of systems engineering that goes into it. We have an integrated team. Frequently, you have people who are mem-

bers of both the systems engineering team for the spacecraft or the mission and also members of the software team.

Where it tends to come together is in the area of fault protection. You'll have a team of two or three or four fault protection engineers who are systems engineers that are looking at the entire system, what can go wrong with the system and what, if anything, you can do to correct it. They're also part of the software team and helping to write the specifications for the software that has to do that.

That's really where it all comes together. Fault protection and fault management is a big part of what makes our missions able to survive for extended periods of time without any direct interaction from the operators.

Andrew: I know that there are some people who are going to be reading this who aren't quite sure exactly how fault protection works, how testing works in general. Without breaching any federal laws, can you tell us a little bit about how you guys go about testing this kind of software?

Peter: Yes, I can give you the general overview. It's important to have—let's see, one, it's important to have visibility into what the software's doing, so you don't want to have any hidden commands, data, features.

Everything needs to be accessible so that you can actually see what's going on inside the software to the largest extent possible. Obviously, you're not going to publish every counter or every loop variable or every intermediate state. But to the extent that there's a hardware measurement available, you want that to be visible from the user's perspective.

The second thing is you need to have platforms that you can test on. In the last 10 years, what we do largely is a software-based simulator.

You have a software simulator that can run on a workstation and it basically models the environment that the spacecraft software operates in. You have models of the avionics systems. You have models of the vehicle dynamics; by that I mean the rotational and translational motions of the vehicle. You have models of the planets and whatever external factors are of importance to your system.

Then you can operate software in an environment that simulates what it's going to expect to see in operations. You run tests in that way and you confirm that the behavior is what you expect it to be, both nominal and off-nominal cases, so that you can identify places where fault protection is working correctly and maybe places where you need to beef it up in some way.

We also take advantage of hardware test beds. That's your highest-fidelity test bed. You actually build duplicate electronics and you put them in a hardware test bed, and then you command the software in a flight-like way. You observe how it behaves, and hopefully it behaves as you expected it to.

That's the best way to do it, but you're limited by the fact that hardware generally operates in real time. There are two problems there. One is the time it takes, and the other is

that it costs a lot of money to get the extra set of hardware. So you're really limited to only a couple of hardware test beds and can't do a whole lot on them. You rely on software test beds because they can generally run faster than real time, and you duplicate those on as many workstations as you can buy.

Andrew: First of all, thank you for the explanation; it's really interesting. It occurs to me that if you went to, say, your typical programming team building typical business applications at a company, and you expected them to put in that kind of testing, that level of quality engineering, you'd have a minor revolt on your hands.

Peter: Which is the reason why our missions to Mars cost $400 million, and your typical business application only costs $1 million to develop. You're looking at a difference in scale and in cost of failure.

The cost of failure if a business application doesn't work quite right is a few irate customers. It can be bad for the business if they really get it wrong, but chances are in the nominal cases it's going to work fine, and in the off-nominal cases they'll get some complaints and they can issue a patch.

If we get it wrong in the off-nominal cases, we may not get a second chance.

Andrew: I'm guessing that unlike a business team, the actual team members, even just the most junior programmers on the team, will have a more focused attitude towards quality, an appreciation of quality among the people who are actually building the software. And maybe even an appreciation of the project itself.

Peter: Yes, I think that people who work here generally feel privileged to be here. I know I certainly do. It's a unique place to work. There are certainly some disadvantages to being here, but there are an awful lot of advantages with respect to just the opportunity to work on the things that we work on.

I think that most people here are very conscientious and really trying to do the best they can. They understand that the things we're doing are difficult, and therefore require attention to detail.

And to the extent that we might miss something because we're all human, we have a robust test program that tries to shake out all of the major defects. We certainly aren't going to get every defect, but if we can get all the major ones, then we're doing well.

The goal is to make the spacecraft survivable so that nothing that we did during development is going to cause it to be lost. That's the main thing: to make the spacecraft robust so that it can protect itself until we can figure out what went wrong and correct it.

Andrew: I'm guessing that all of that attention to detail, that attention to quality, the awareness of the need to get it right, and just the general feeling of being privileged to

work there, that's got to have a positive effect on teams and their ability to gel quickly and actually get moving in the same direction.

Jenny: I have another question. One of the things that some of the people we've been talking to have brought up as a problem in making teams gel is that a lot of times people will have divergent goals. They'll understand the project a little bit differently from their teammates.

Do you think that that's something that you've encountered in JPL, or is that something that pretty much everybody gets? We're sending a spacecraft to Mars, and that's the goal of the project.

Peter: Yeah, I think on the flight projects that I've worked on, people are pretty well oriented towards the goal and what the project's supposed to do.

On a day-to-day basis, you occasionally have someone who heads off in the wrong direction and does something that's not exactly what you expected, but generally the high-level goal is understood and everybody understands the seriousness of what can happen if we don't get it right the first time.

Andrew: Does that have an impact on how well people work together?

Peter: I think the things that probably affect the day-to-day operations of the team are, one, the natures of the personalities, as I mentioned earlier. And two, it's not all rainbows and roses here. While we are certainly honored and privileged to be here, along with that comes a lot of responsibility.

I tend to be more laid back, but I know people who really do stress out over the fact that what they're doing is so critical.

There are often long hours for many months. We have to hit planetary launch windows that are about three weeks long and only come around once every two years or so. You really don't have an option to slip a launch.

If you slip a launch, it costs tens, if not hundreds of millions of dollars, because you've got to then keep being staffed up for the time that you are waiting for the next window to come around. You've got equipment to store. You've got additional facilities to keep running. There are all kinds of costs involved. We really do try very, very hard to meet the launch window.

There's a lot of stress involved, too, and there have been many documented cases of personal problems occurring as a result of the work schedule that we sometimes have to adhere to.

Jenny: So given that kind of stress, and given the fact that the team is under such strict, strict deadlines, do you find that that makes it more difficult for them to get along? Or do you think that that's something that brings them together?

Peter: I think that it depends on the person, and where you are in the mission timeline.

Now, frankly, the people that really can't handle the stress eventually drop off the team. They either leave for another job or they just get reassigned. There's really no room in this business for someone who can't do the work. To that extent, I think that there's a lot of cooperation and general agreement on how to do things.

As far as the teams go, the teams tend to work pretty well together. On the ones that I've been on, there hasn't been too much discord. Sometimes there are differences of opinion on how to proceed, but generally, once a direction is given, you proceed pretty well.

Where we have gotten into trouble, and this is my personal opinion, is reuse. There have been several attempts to produce reusable software architectures that can then be used to reduce costs and bring our overall project costs down and make our projects more affordable.

And frankly, I think that the lack of a solid deadline to hit on those has contributed to the fact that people tend to disagree more on which direction to go in, and they tend to not make as much progress on those particular projects.

Jenny: That's really interesting. A couple of the other people that we've interviewed for this book have also brought up the idea that reuse as a goal in and of itself can sometimes actually be an obstacle.

Peter: Yes, I think that's an accurate statement.

Jenny: The interesting thing to me is that reuse is something that people have really touted as a big goal of a lot of projects. I've worked with or heard of a lot of teams who designed entire software packages to make them reusable, especially over the last five or six years. That seemed to be happening industrywide.

In your experience, even with the kind of rigor you guys put in, do you run into trouble delivering software that's meant to have lots of layers of abstraction, where the goal is to be reusable?

Peter: I think we do. It's something we've been trying to do for 20 years. I'm not sure that we've still gotten it right, to be honest.

What tends to happen is that you get an initiative going to build some kind of reusable platform, and it's not just software. It also applies to the hardware and the avionics and various components that we use.

Then you get a bunch of smart people together, and those smart people tend to not really agree on what the best way to do things is, and so they debate and they talk about it and they make some progress. But eventually what happens is some project comes along that wants to use the system or needs to use the system, and suddenly the system becomes dedicated to that project.

That's when you start to get some progress in terms of getting to a real functional system, but it loses its reusability aspects because now the project owns the project, and doesn't want to spend project money to help the next project out. They need to spend their project money to get their project going and to make their project the best that it can be.

It's really a tough nut to crack. I haven't seen anybody successfully complete a reusable platform yet, to be honest.

Andrew: Do you think that maybe what happens is that once the common, well-understood goal of putting a spacecraft in orbit around a planet is gone, the goal becomes more amorphous? Do you think that the lack of that clear goal might be making the team less cohesive?

Peter: Yes. And you get into philosophical arguments, and once you descend into abstraction—you could think of it as ascending—but once you get into that abstract level of thinking, then it becomes much harder to justify a particular direction as the right one.

Whereas when you've got a concrete goal—"We've got to put this functionality together, and it has to be here by this date or we're going to miss our launch window"—eventually people fall in line. They say, "OK, well, this isn't the way that I would really like to see it done, but it's going to work, and it's good enough."

What you lose is that "good enough" aspect when you're going for something reusable. It's natural to do that because you're trying to find the best solution and not just the good enough one. But that tends, as you said, to be an obstacle to actually accomplishing something.

Andrew: It sounds like you fall right into that locking-horns trap that you talked about before, except in this case, there's no longer the trump card of "We have to get the mission out. Our launch window is at this date for these three weeks, and if we spend the next six months locking horns, we're going to miss our launch window." That trump card's gone and the situation just devolves into locked horns, into deadlock.

Peter: That's exactly right. One of the things that we also suffer from here at JPL, and it's sort of unique to the business we're in, is that every one of our missions is unique. I've yet to see two missions that are identical or even very similar.

I mean, even if you look at the Mars Pathfinder and the MER and the Mars Science Laboratory, which is currently in development, all three are Mars rovers. Yet each mission is very unique in the hardware that's onboard, in the methods used to arrive on Mars, and in the instrument packages that are onboard.

What you have is a situation where technology outpaces the life cycle that we're in. It takes three to five years to get a mission from concept to launch. Technology moves at roughly that same rate—even faster, actually, if you look at how quickly computers become obsolete.

We end up with a situation where even if we can agree to develop a certain platform using a certain language on a certain piece of hardware, by the time we get it ready for the first mission, technology has moved ahead. That makes the job even that much harder to actually come up with a standard, reusable platform.

We're still trying, and there are still some smart people thinking about it and trying to figure out what to do. And perhaps someday, with pluggable architectures, we'll finally come up with something that is more universal.

Jenny: A question hit me as you were talking about the testing team earlier. One of the big tenets of agile development that both Andrew and I really like a lot is the responsibility for quality that's put on developers, the fact that they're expected to unit-test their own code and that through continuous integration, it gets easier and easier to monitor changes that are made in code. So I wondered, do you guys use any of those techniques?

Peter: I am not aware of anybody using agile development here.

Jenny: Not agile development, per se, but more quality-related activities that are done by developers themselves.

Peter: Yes, we do expect our developers to unit-test their code before they deliver it for integration.

We tend not to do continuous integration. It's more along the lines of everybody gets together, we figure out the requirements and the design for the current cycle, we then produce those modules, unit-test them, and they come together at once.

Andrew: I have a question about integration testing. What you just said reminded me about when I first read about integration in general, back when I was first learning about software engineering.

If you guys didn't invent the term integration *with respect to software, you certainly were some of the first to popularize it back in the '80s and early '90s. For those of us purely working on software, where every system in the project is a software system, integration just means integrating modules together. But I'm assuming that when you say* integration, *you're using the larger, original meaning of the word, where you're integrating the development of the software and the development hardware, too. And if you go back to that meaning, the term unit—as in "unit testing"—had a specific meaning, too. So is that what you mean by "integration?" Integration of your software with a hardware component within the larger system that you're working on?*

Peter: Yes. So there are two levels of integration. We develop independent software modules, or units, which are pieces that are small enough that a single person or a course of people can actually get their arms around them and adequately implement them and test them.

Then those come together in our software integration. We put those together, and those get linked together and then we run some software integration tests on, say, one of our SoftSim test platforms that I mentioned earlier, or perhaps even on a hardware test bed if we have one available.

Then once the software is integrated and ready to go, it's declared ready by the software manager, and it gets sent to the hardware test beds for certification. It then goes to the vehicle.

One of the phases of the mission is called ATLO—Assembly Test and Launch Operations. That's also known as Phase D, which is just a letter designation for that particular time in the project life cycle. That starts from the delivery of hardware to integration on what you could call the factory floor, although we attempt to do it in a clean room.

You start to get hardware together, and the hardware gets assembled. Once you have a sufficient critical mass of hardware assembled together, you can upload the software onto the avionics system and then you can start to test it out and make sure that, A, the software correctly commands the hardware, and B, the hardware works properly when you command it.

We test as much as we can on the ground. Typically, that runs about nine to 12 months prior to delivery to the launch site. That's our ATLO phase. When a new software build is ready, it goes to the hardware test bed and then it goes to ATLO for integration with the vehicle and the testing that occurs there, so we do have several levels of integration that occur.

Andrew: So I guess for somebody who when they first heard you say "We don't do continuous integration" wasn't sure why not, that should explain it. I think it's safe to say that if your integration process involves a clean room, it probably precludes automated integration.

Peter: True. We don't have automated integration. Now, I have seen systems at some of our industry partners where they will do, for example, a nightly build of everything that has been checked into the CM system for that day. The reason they do that is to make sure that the whole build system hasn't been upset or broken by some changes that were made.

Then there is a process where at some designated point or when everybody says thumbs up, we designate that that particular build is ready for integration and it all gets pushed forward in the process.

As I understand continuous integration, I think the way you're using it, anybody can basically say, "OK, my module's ready," and just stick it in and start to work with it. We really can't do that. It's really a much more coordinated effort.

Andrew: One of the things that people really like about continuous integration or an automated integration process is that it's very efficient, and it enforces a certain quality level. People don't have to spend a lot of time coordinating with each other to build the software, and they can trust that the software does have a certain level of quality because they can depend on the system to make sure that it builds cleanly, passes its unit tests, etc. Now, you can't do that level of automation, for the reasons you just outlined. But are there other ways that you make your process more efficient? And are there ways that you make sure that the quality of the code is always high?

Peter: What we will do is we sometimes run those cycles in parallel, so you'll have build two in integration, and build three will be in the design phase. You'll have people working on different aspects of the system in different phases of the life cycle at the same time.

Yes, we do expect people to deliver quality, unit-tested code. And not only that, it has to adhere to the coding standards. It has to be well commented. They have to deliver documentation associated with it that describes what it's supposed to do, what problem reports are addressed by the current revision, what problem reports are still outstanding. We keep track of all the defects that are found in integration and tests, and we have to address all of those before launch.

We also have a NASA facility called the Independent Verification and Validation facility in Fairmont, West Virginia, that does independent audits of our processes and our code and helps us to locate any residual issues, defects, requirements, problems.

Jenny: Like an external SQA group?

Peter: Yes, exactly like an external SQA group. We have independent SQA here at JPL that is not reporting through the project management, but reports to our SQA organization. Then on top of that, we have an outside, external SQA, which is provided by NASA.

Jenny: It sounds like it's very transparent. People write code. They write documentation around that code. They go through many levels of review by themselves and then with the team, right?

Peter: Yes.

Jenny: Which sounds really great in terms of finding defects as early in the process as possible. But one question that I have about that is I've noticed in my career that a lot of developers get really, really defensive when other people look at their code, or even when people have negative comments about code, or find defects, even. Do you feel like the culture where you are is different than that?

Peter: I guess I do. The developers that I've worked with have all been fairly responsive to having people review their code and point out places where there might be problems. In fact, we go through review, a code walkthrough, of every module that we put onto our spacecraft.

There's a certain amount of professional pride, of course. People don't like to see their code picked apart, but on the other hand, if there really is a problem, I've never known anybody that wasn't willing to accept the fact that there was a problem and then fix it. In fact, a lot of times they're really happy when you say, "You've got a concurrency defect here and you could have a potential deadlock," and they say, "Oh, yeah, I guess that's true. We'll have to fix that." Haven't had any real problems with that.

Jenny: And what about the relationship between the testers and the developers? Is there a good relationship there?

Peter: Generally, yes, I think there is. In some cases, it's the same person, really. We don't always have independent tests. We have independent review test results. So a developer will do their unit tests. We have a unit-test review, and then when we get to integration, it is frequently the same developer that is writing the tests for integration and sometimes even executing it.

But we always have a second set of eyes, someone from the systems engineering organization or someone from the subsystem organization, that's responsible for that piece of equipment that the software's supposed to operate.

We always have someone from systems engineering look at the test and verify that the software looks like it's doing the right thing.

Andrew: I really can appreciate how carefully you guys have to be with your quality, and the results speak for themselves in the missions you fly.

But I can just see some of our readers reading this, maybe somebody who's used to being a down-and-dirty developer who has to fire things off really rapidly, and often in not the most rigorous environment. I could see them getting an interoffice mail package from an IV&V group or an SQA group, opening it up, and groaning because it's yet another report from someone who I don't even know who's looked through all my code and come up with all these problems, these bugs I've introduced.

Peter: Well, it's no picnic. It really is something of a burden to have to deal with these things.

On Phoenix, there were over 800 technical issue memoranda that were generated by the NASA IV&V group. I had to review every one of those technical issue memoranda to ensure that there wasn't anything in there that was critical to the health of the spacecraft.

I think there were roughly half a dozen of those that were significant enough that we needed to fix them. There were many of them that we caught in our own internal reviews or testing. Then there were some that were what you might categorize as nits where you say, "Yeah, you know, I understand what you're saying, but that's not really going to be a problem because we're either using it in a different way, our use precludes the problem from occurring, or it's just a minor point that, geez, stylistically it would look nice, but we're not worried about it."

Andrew: Do you think that if you asked everybody on the team to give their honest opinion, would the general consensus be that it was worth all of the effort or not worth all of the effort? Are these half a dozen defects things that would have actually crashed a spaceship into a moon?

Peter: None of them were things that we would call mission-critical. That is, we wouldn't have lost the mission if we had failed to fix those. But they would have been annoying. We might have lost a couple of days on the surface, maybe even a week, trying to figure out what happened and get it corrected.

It's good that we found them. Was it worth the effort? I guess I would say, yes, I think it was. It's like eating your vegetables. Sometimes you don't want to do it, but you have to because it's good for you in the long run.

Andrew: I think that's very interesting. If you were going to plot out software and quality practices on a scale of least onerous to most onerous, or from least overhead to most overhead, you guys would be at the very end of that scale. And you, the guy who actually

has to review the memoranda, still feel it's worth it. It was worth the time and effort you and your team put in, simply based on how much better it made the final product.

Peter: I think so. If you look at Phoenix, we had very, very few problems in flight. I can't think of any major or even significant software issues that we had prior to landing.

And post-landing, we had one issue, a problem that was related to reuse of a telemetry system that was designed for an orbiter, where it wasn't expected that you would be rebooting it frequently.

On the surface, we shut down and wake up several times a day, and there was a counter that wasn't managed quite right. When that counter rolled over, we had a problem that caused the spacecraft to go into what we call safe mode, a protective mode where it basically shuts all of the science down and tries to maintain a good power environment, and communicate back to Earth and wait for instructions.

Fortunately, we were able to identify the cause of that problem rather quickly; within about 24 hours, we understood what happened and got it corrected within a few days.

But, really, as spacecraft go, missions that I've been involved in, this mission had the least amount of software problems that I can remember.

Jenny: Just one more kind of general question, Peter. Is there anything in particular that characterized the best team that you've worked on? Something that you want to drop out as a pearl of wisdom for readers?

Peter: I think that the best teams that I've worked on and worked with have really had a common goal. You're working to the same goal and you appreciate each other. You have a respect for everyone's skills, and even though somebody's the project leader and somebody else might be the chief architect, everybody really understands that contributions are being made all around.

It's funny; the most satisfying projects are not necessarily the ones that do the grandest things, but the ones where the team is cohesive and everybody is respected for their contributions.