They Write the Right Stuff

As the 120-ton space shuttle sits surrounded by almost 4 million pounds of rocket fuel, exhaling noxious fumes, visibly impatient to defy gravity, its on-board computers take command.

The right stuff kicks in at T-minus 31 seconds.

As the 120-ton space shuttle sits surrounded by almost 4 million pounds of rocket fuel, exhaling noxious fumes, visibly impatient to defy gravity, its on-board computers take command. Four identical machines, running identical software, pull information from thousands of sensors, make hundreds of milli-second decisions, vote on every decision, check with each other 250 times a second. A fifth computer, with different software, stands by to take control should the other four malfunction.

At T-minus 6.6 seconds, if the pressures, pumps, and temperatures are

nominal, the computers give the order to light the shuttle main engines — each of the three engines firing off precisely 160 milliseconds apart, tons of super-cooled liquid fuel pouring into combustion chambers, the ship rocking on its launch pad, held to the ground only by bolts. As the main engines come to one million pounds of thrust, their exhausts tighten into blue diamonds of flame.

Then and only then at T-minus zero seconds, if the computers are satisfied that the engines are running true, they give the order to light the solid rocket boosters. In less than one second, they achieve 6.6 million pounds of thrust. And at that exact same moment, the computers give the order for the explosive bolts to blow, and 4.5 million pounds of spacecraft lifts majestically off its launch pad.

It's an awesome display of hardware prowess. But no human pushes a button to make it happen, no astronaut jockeys a joy stick to settle the shuttle into orbit.

The right stuff is the software. The software gives the orders to gimbal the main engines, executing the dramatic belly roll the shuttle does soon after it clears the tower. The software throttles the engines to make sure the craft doesn't accelerate too fast. It keeps track of where the shuttle is, orders the solid rocket boosters to fall away, makes minor course corrections, and after about 10 minutes, directs the shuttle into orbit more than 100 miles up. When the software is satisfied with the shuttle's position in space, it orders the main engines to shut down — weightlessness begins and everything starts to float.

But how much work the software does is not what makes it remarkable. What makes it remarkable is how well the software works. This software never crashes. It never needs to be re-booted. This software is bug-free. It is perfect, as perfect as human beings have achieved. Consider these stats : the last three versions of the program — each 420,000 lines long-had just one error each. The last 11 versions of this software had a total of 17 errors.

Commercial programs of equivalent complexity would have 5,000 errors.

This software is the work of 260 women and men based in an anonymous office building across the street from the Johnson Space Center in Clear Lake, Texas, southeast of Houston. They work for the “on-board shuttle group,” a branch of Lockheed Martin Corps space mission systems division, and their prowess is world renowned: the shuttle software group is one of just four outfits in the world to win the coveted Level 5 ranking of the federal governments Software Engineering Institute (SEI) a measure of the sophistication and reliability of the way they do their work. In fact, the SEI based it standards in part from watching the on-board shuttle group do its work.

The group writes software this good because that’s how good it has to be. Every time it fires up the shuttle, their software is controlling a $4 billion piece of equipment, the lives of a half-dozen astronauts, and the dreams of the nation. Even the smallest error in space can have enormous consequences: the orbiting space shuttle travels at 17,500 miles per hour; a bug that causes a timing problem of just two-thirds of a second puts the space shuttle three miles off course.

NASA knows how good the software has to be. Before every flight, Ted Keller, the senior technical manager of the on-board shuttle group, flies to Florida where he signs a document certifying that the software will not endanger the shuttle. If Keller can’t go, a formal line of succession dictates who can sign in his place.

Bill Pate, who’s worked on the space flight software over the last 22 years, [/url]says the group understands the stakes: “If the software isn’t perfect, some of the people we go to meetings with might die.

Software is everything. (It also sucks.)

In the history of human technology, nothing has become as essential as fast

as software.

Virtually everything — from the international monetary system and major power plants to blenders and microwave ovens — runs on software. In office buildings, the elevators, the lights, the water, the air conditioning are all controlled by software. In cars, the transmission, the ignition timing, the air bag, even the door locks are controlled by software. In most cities so are the traffic lights. Almost every written communication that's more complicated than a postcard depends on software; every phone conversation and every overnight package delivery requires it.

Software is everything. It also sucks.

"It's like pre-Sumerian civilization," says Brad Cox, who wrote the software for Steve Jobs NeXT computer and is a professor at George Mason University. "The way we build software is in the hunter-gatherer stage."

John Munson, a software engineer and professor of computer science at the University of Idaho, is not quite so generous. "Cave art," he says. "It's primitive. We supposedly teach computer science. There's no science here at all."

Software may power the post-industrial world, but the creation of software remains a pre-industrial trade. According to SEI's studies, nearly 70% of software organizations are stuck in the first two levels of SEI's scale of sophistication: chaos, and slightly better than chaos. The situation is so severe, a few software pioneers from companies such as Microsoft have broken away to teach the art of software creation (see "Drop and Code me Twenty!")

Mark Paulk, a senior member of the SEI technical, says the success of software makes its weaknesses all the more dramatic. "We've developed software products that are enormously complex and enormously powerful. We're critically dependent on it," says Paulk. "Yet everyone complains how

bad software is, with all the defects. If you bought a car with 5,000 defects, you'd be very upset."

In this software morass, the on-board shuttle group stands out as an exception. Ten years ago the shuttle group was considered world-class. Since then, it has cut its own error rate by 90%.

To be this good, the on-board shuttle group has to be very different — the antithesis of the up-all-night, pizza-and-roller-hockey software coders who have captured the public imagination. To be this good, the on-board shuttle group has to be very ordinary — indistinguishable from any focused, disciplined, and methodically managed creative enterprise.

In fact, the group offers a set of textbook lessons that applies equally to programmers, in particular, and producers, in general. A look at the culture they have built and the process they have perfected shows what software-writing must become if software is to realize its promise, and illustrates what almost any team-based operation can do to boost its performance to achieve near-perfect results.

Software for Grown-Ups

"Shipping hell continued today. Grind, grind, grind. We'll never make it. Have I said that already? Why do we always underestimate our shipping schedules? I just don't understand. In at 9:30 AM; out at 11:30 PM Dominos for dinner. And three diet Cokes."

No, it's not the on-board shuttle group. It's Douglas Coupland's "Microserf's," a true-to-life fictional account of life in the software-fast-lane. And it's the dominant image of the software development world: Gen-Xers sporting T-shirts and distracted looks, squeezing too much heroic code writing into too little time; rollerblades and mountain bikes tucked in corners; pizza boxes and Starbucks cups discarded in conference rooms; dueling tunes from Smashing Pumpkins, Alanis Morrisette and the Fugees. Its the

world made famous, romantic, even inevitable by stories out of Sun Microsystems, Microsoft, and Netscape.

It's not the story of the on-board shuttle group. Their quarters are a study in white-collar pedestrian. The most striking thing is how ordinary they look. Other than the occasional bit of shuttle memorabilia, you could be in the offices of any small company or government agency. Everyone has his or her own small office, and the offices have desks, PCs, and sparse personal artifacts. People wear moderately dressy clothes to work, neat but nothing flashy, certainly nothing grungy.

It's strictly an 8-to-5 kind of place — there are late nights, but they're the exception. The programmers are intense, but low-key. Many of them have put in years of work either for IBM (which owned the shuttle group until 1994), or directly on the shuttle software. They're adults, with spouses and kids and lives beyond their remarkable software program.

That's the culture: the on-board shuttle group produces grown-up software, and the way they do it is by being grown-ups. It may not be sexy, it may not be a coding ego-trip — but it is the future of software. When you're ready to take the next step — when you have to write perfect software instead of software that's just good enough — then it's time to grow up.

Ted Keller, 48, the group's senior technical manager, looks and sounds like the headmaster of a small private high school. It's Keller's job to make sure the software gets delivered on time, with all its capabilities, without regard to turf battles. He's a compact, bald man, a little officious and persnickety, qualities any astronaut would find reassuring. He has a gentle, geeky sense of humor, not so much with outsiders, but with his crowd.

It comes out in a meeting between members of the software group and their NASA counterparts. It's held in a small conference room overstuffed with 22 people and an overhead projector. Several times, from the back of the room, Keller issues a wry remark about the speed of code delivery, or the detail of

some specifications, and the room lightens with laughter.

Otherwise, the hour-long meeting is sober and revealing, a brief window on the culture. For one thing, 12 of the 22 people in the room are women, many of them senior managers or senior technical staff. The on-board shuttle group, with its stability and professionalism, seems particularly appealing to women programmers.

For another, it's an exercise in order, detail, and methodical reiteration. The meeting is a classic NASA performance — a rehearsal for an almost identical meeting several weeks away. It consists of walking through an enormous packet of data and view — graphs which describe the progress and status of the software line by line. Except for Keller's occasional asides, the tone is businesslike, almost formal, the view – graphs flashing past as quickly as they can be read, a blur of acronyms, graphs, and charts.

What's going on here is the kind of nuts-and-bolts work that defines the drive for group perfection — a drive that is aggressively intolerant of ego-driven hotshots. In the shuttle group's culture, there are no superstar programmers. The whole approach to developing software is intentionally designed not to rely on any particular person.

And the culture is equally intolerant of creativity, the individual coding flourishes and styles that are the signature of the all-night software world. "People ask, doesn't this process stifle creativity? You have to do exactly what the manual says, and you've got someone looking over your shoulder," says Keller. "The answer is, yes, the process does stifle creativity."

And that is precisely the point — you can't have people freelancing their way through software code that flies a spaceship, and then, with peoples lives depending on it, try to patch it once its in orbit. "Houston, we have a problem," may make for a good movie; it's no way to write software. "People have to channel their creativity into changing the process," says Keller, "not changing the software."

The tight strictures the group practices can make the siren song of the rock n roll software world hard to resist. Quinn Larson, 34, had worked on shuttle software for seven years when he left last January to go to work for Micron Technology in Boise, Idaho, automating the manufacture of Microns memory chips.

At Micron, Larson was given the task to automate the saws that cut finished chip wafers to the right size. Screw up the program, you destroy the valuable wafers.

"It was up to me to decide what to do," says Larson. "There were no meetings, there was no record-keeping." He had freedom; it was a real kick. But to Larson's way of thinking, the culture didn't focus on, well, the right stuff. "Speed there was the biggest thing," he says. "The engineers would say, these are our top priorities, and we need to get'em as fast as we can." But the impression Larson got was that engineers didn't seem too concerned about how well the finished software actually worked. "Basically, they wanted quick software — just put it out the door."

Larson started back at the shuttle group in mid-August. "The people here are just of the highest caliber," he said on his first day back in Clear Lake.

It's the Process

How do they write the right stuff?

The answer is, it's the process. The group's most important creation is not the perfect software they write — it's the process they invented that writes the perfect software.

It's the process that allows them to live normal lives, to set deadlines they actually meet, to stay on budget, to deliver software that does exactly what it promises. It's the process that defines what these coders in the flat plains of southeast suburban Houston know that everyone else in the software world

is still groping for. It's the process that offers a template for any creative enterprise that's looking for a method to produce consistent – and consistently improving — quality.

The process can be reduced to four simple propositions:

1. The product is only as good as the plan for the product. At the on-board shuttle group, about one-third of the process of writing software happens before anyone writes a line of code. NASA and the Lockheed Martin group agree in the most minute detail about everything the new code is supposed to do — and they commit that understanding to paper, with the kind of specificity and precision usually found in blueprints. Nothing in the specs is changed without agreement and understanding from both sides. And no coder changes a single line of code without specs carefully outlining the change. Take the upgrade of the software to permit the shuttle to navigate with Global Positioning Satellites, a change that involves just 1.5% of the program, or 6,366 lines of code. The specs for that one change run 2,500 pages, a volume thicker than a phone book. The specs for the current program fill 30 volumes and run 40,000 pages.

"Our requirements are almost pseudo-code," says William R. Pruett, who manages the software project for NASA. "They say, you must do exactly this, do it exactly this way, given this condition and this circumstance."

This careful design process alone is enough to put the shuttle organization in a class by itself, says John Munson of the University of Idaho. Most organizations launch into even big projects without planning what the software must do in blueprint-like detail. So after coders have already started writing a program, the customer is busily changing its design. The result is chaotic, costly programming where code is constantly being changed and infected with errors, even as it is being designed.

"Most people choose to spend their money at the wrong end of the process," says Munson. "In the modern software environment, 80% of the

cost of the software is spent after the software is written the first time — they don't get it right the first time, so they spend time flogging it. In shuttle, they do it right the first time. And they don't change the software without changing the blueprint. That's why their software is so perfect."

2. The best teamwork is a healthy rivalry. Within the software group, there are subgroups and subcultures. But what could be divisive office politics in other organizations is actually a critical part of the process.

The central group breaks down into two key teams: the coders – the people who sit and write code — and the verifiers — the people who try to find flaws in the code. The two outfits report to separate bosses and function under opposing marching orders. The development group is supposed to deliver completely error-free code, so perfect that the testers find no flaws at all. The testing group is supposed to pummel away at the code with flight scenarios and simulations that reveal as many flaws as possible. The result is what Tom Peterson calls "a friendly adversarial relationship."

"They're in competition for who's going to find the errors," says Keller. "Sometimes they fight like cats and dogs. The developers want to catch all their own errors. The verifiers get mad, 'Hey, give it up! You're taking away from our time to test the software!'"

The developers have even begun their own formal inspections of the code in carefully moderated sessions, a rigorous proof-reading they hope will confound the testers. The verifiers, in turn, argue that they deserve credit for some errors found before they even start testing. "From the verification group's perspective," says Pat McLellan, a senior manager, "we're aware that if there was no independent verification group, the developers would tend to be more lax. Just the presence of our group makes them more careful."

The results of this friendly rivalry: the shuttle group now finds 85% of its errors before formal testing begins, and 99.9% before the program is delivered to NASA.

3. The database is the software base. There is the software. And then there are the databases beneath the software, two enormous databases, encyclopedic in their comprehensiveness.

One is the history of the code itself — with every line annotated, showing every time it was changed, why it was changed, when it was changed, what the purpose of the change was, what specifications documents detail the change. Everything that happens to the program is recorded in its master history. The genealogy of every line of code — the reason it is the way it is — is instantly available to everyone.

The other database — the error database — stands as a kind of monument to the way the on-board shuttle group goes about its work. Here is recorded every single error ever made while writing or working on the software, going back almost 20 years. For every one of those errors, the database records when the error was discovered; what set of commands revealed the error; who discovered it; what activity was going on when it was discovered — testing, training, or flight. It tracks how the error was introduced into the program; how the error managed to slip past the filters set up at every stage to catch errors — why wasn't it caught during design? during development inspections? during verification? Finally, the database records how the error was corrected, and whether similar errors might have slipped through the same holes.

The group has so much data accumulated about how it does its work that it has written software programs that model the code-writing process. Like computer models predicting the weather, the coding models predict how many errors the group should make in writing each new version of the software. True to form, if the coders and testers find too few errors, everyone works the process until reality and the predictions match.

"We never let anything go," says Patti Thornton, a senior manager. "We do just the opposite: we let everything bother us."

4. Don’t just fix the mistakes — fix whatever permitted the mistake in the first place.

The process is so pervasive, it gets the blame for any error — if there is a flaw in the software, there must be something wrong with the way its being written, something that can be corrected. Any error not found at the planning stage has slipped through at least some checks. Why? Is there something wrong with the inspection process? Does a question need to be added to a checklist?

Importantly, the group avoids blaming people for errors. The process assumes blame – and it’s the process that is analyzed to discover why and how an error got through. At the same time, accountability is a team concept: no one person is ever solely responsible for writing or inspecting code. “You don’t get punished for making errors,” says Marjorie Seiter, a senior member of the technical staff. “If I make a mistake, and others reviewed my work, then I’m not alone. I’m not being blamed for this.”

Ted Keller offers an example of the payoff of the approach, involving the shuttles remote manipulator arm. “We delivered software for crew training,” says Keller, “that allows the astronauts to manipulate the arm, and handle the payload. When the arm got to a certain point, it simply stopped moving.”

The software was confused because of a programming error. As the wrist of the remote arm approached a complete 360-degree rotation, flawed calculations caused the software to think the arm had gone past a complete rotation — which the software knew was incorrect. The problem had to do with rounding off the answer to an ordinary math problem, but it revealed a cascade of other problems.

“Even though this was not critical,” says Keller, “we went back and asked what other lines of code might have exactly the same kind of problem.” They found eight such situations in the code, and in seven of them, the rounding off function was not a problem. “One of them involved the high-gain antenna

pointing routine," says Keller. "That's the main antenna. If it had developed this problem, it could have interrupted communications with the ground at a critical time. That's a lot more serious."

The way the process works, it not only finds errors in the software. The process finds errors in the process.

Just a Software Problem

The B-2 bomber wouldn't fly on its maiden flight — but it was just a software problem. The new Denver airport was months late opening and millions of dollars over budget because its baggage handling system didn't work right — but it was just a software problem. This spring, the European Space Agency's new Ariane 5 rocket blew up on its maiden launch because of a little software problem. The federal government's major agencies – from the IRS to the National Weather Service — are beset with projects that are years late and hundreds of millions of dollars over budget, often because of simple software problems. Software is getting more and more common and more and more important, but it doesn't seem to be getting more and more reliable.

As the rest of the world struggles with the basics, the on-board shuttle group edges ever closer to perfect software. Admittedly they have a lot of advantages over the rest of the software world. They have a single product: one program that flies one spaceship. They understand their software intimately, and they get more familiar with it all the time. The group has one customer, a smart one. And money is not the critical constraint: the groups $35 million per year budget is a trivial slice of the NASA pie, but on a dollars-per-line basis, it makes the group among the nation's most expensive software organizations.

And that's the point: the shuttle process is so extreme, the drive for perfection is so focused, that it reveals what's required to achieve relentless execution. The most important things the shuttle group does — carefully

planning the software in advance, writing no code until the design is complete, making no changes without supporting blueprints, keeping a completely accurate record of the code — are not expensive. The process isn't even rocket science. Its standard practice in almost every engineering discipline except software engineering.

Plastered on a conference room wall, an informal slogan of the on-board shuttle group captures the essence of keeping focused on the process: "The sooner you fall behind, the more time you will have to catch up."

Charles Fishman (fish@nando.net) is a writer based in Raleigh, North Carolina.