The Ganssle Group

Perfecting the Art of Building Embedded Systems

The Ganssle Group
PO Box 38346, Baltimore, MD 21231
Email info@ganssle.com

Disasters, and what we can learn from them.

Published in ESP, November 2000

Crash and Burn

We're not terribly good at learning from our successes; smug with the satisfaction of a job well done most of us proceed immediately to the next task at hand. It's a shame that we can't look at a job well done and then dig deeply into what happened how when, to suck the educational content of the project dry.

Ah, but failures are indeed a different story. High profile disasters inevitably produce an investigation, calls for Congress to "do something", and in the best of circumstances a change in the way things are built so the accident does not get repeated.

Isn't it astonishing that airplane travel is so reliable? That we can zip around the sky at 600 knots, 7 miles up, in an ineffably complex device created by flawed people? Perhaps aviation's impressive safety record is a by-product of the way the industry manages failures. Every crash is investigated; each yields new training requirements, new design mods, or other system changes to eliminate or reduce the probability of such a disaster striking again.

Though crashes are rare, they do occur, so airliners carry expensive flight data recorders whose sole purpose is to produce post-accident clues to the safety board. What a shame that we firmware folks don't have a similar attitude. Mostly we're astonished when our systems break or a bug surfaces. I hope that in the future we learn to write code proactively, expecting bugs and problems but finding or trapping them early, and leaving a trail of clues as to what went wrong.

I believe we should examine disasters, our own and others, as so many embedded systems crash in similar ways. I collect embedded disaster stories, not from morbid fascination but because I think they offer universal lessons. Here's a few that are instructive.

NEAR

December 20, 1998, the Near Earth Asteroid Rendezvous spacecraft, after three years enroute to 433 Eros, executed a main engine burn intended to place the vehicle in orbit about the asteroid. The planned 15 minute burn aborted almost immediately; firmware put the spacecraft into a safe mode, as planned in case of such a contingency. But then NEAR unexpectedly went silent. 27 hours later communications resumed, but ground controllers found that it had dumped most of the mission's fuel.

Controllers spent a few days analyzing data to understand what happened, and then initiated a series of burns that will ultimately lead to NEAR's successful rendezvous with the asteroid. But two thirds of the spacecraft's fuel had been dumped, using all of the mission's reserves. The good news is that there's enough fuel – barely – to complete the original goals of the NEAR mission. But reduced fuel means things happen more slowly, so NEAR's rendezvous with 433 Eros will be 13 months later than planned.

Like so many real-world failures, a series of events, each not terribly critical, led to the fuel dump.

Immediately after the engine fired up for the planned 15 minute burn, accelerometers detected a lateral acceleration that exceed a limit programmed into the firmware. This momentary under-one-second transient was in fact not out of bounds for the mechanical configuration of the spacecraft. But the propulsion unit is cantilevered from the base of the spacecraft, creating a bending response that, according to the report (see reference 1) "was not appreciated". Quoting further "In retrospect, the correct thing for the G&C software to have done would have been to ignore (blank out) the accelerometer readings during the brief transient period". In other words, though the transient wasn't anticipated, the software was too aggressive in qualifying accelerometer inputs.

With the software figuring lateral movement exceeded a pre-programmed limit, it shut the motor down and put the spacecraft into a safe mode. The firmware used thrusters to rotate NEAR to an earth-safe attitude. Code then ran a script designed to change over from thrusters to reaction wheels (heavy spinning wheels that absorb or impart spin to the spacecraft) for attitude control. According to the report "Due to insufficient review and testing of the clean-up script, the commands needed to make a graceful transition to attitude control using reaction wheels were missing." Wow!

Excessive spacecraft momentum meant that the reaction wheels just weren't up to the task of putting NEAR into the earth-safe mode. The firmware did try, for the programmed 300 seconds, but then gave up and started warming up thrusters, which offer much more kick than the momentum wheels. Now the only chance to save the spacecraft was to go to the lowest level save mode, "sun-safe", where it spun slowly around an axis pointing towards the sun. This would keep the batteries charged till ground intervention could help out.

Seven minutes later an error in a data structure (i.e., a parameter stored in the firmware) led to the system thinking a momentum wheel that was running a its maximum speed was stopped. A series of race conditions, exacerbated by low batteries, led to some 7900 seconds of thruster firing over the course of many hours. Eventually NEAR did stabilize in sun-safe mode, though now missing 29 kg of critical propellant.

So NEAR's troubles stem ultimately from a transient due to an odd vibration mode – something the firmware design team could not have anticipated. This rather small transient revealed flaws in the firmware that, in large part, led to a near-catastrophe (pun intended).

The review board inspected some, but not all, of the system's 80,000 lines of code (C, ADA, and assembly). They uncovered 9 software bugs and 8 data structure errors. Bugs included poorly designed exception handlers and critical variables that could be erroneously overwritten.

Hindsight is certainly a powerful microscope, especially when zooming in on a specific problem that causes a mishap. But I can't help but wonder why the post-failure review board's firmware review was so much more effective than those – if any – performed during original design. The report's recommendation 1c insists that from now on all command scripts must be tested, especially those critical to spacecraft safety – including abort cases. Well, duh!

You'd think configuration management would be a no-brainer for a mission costing many megabucks. Turns out the flight software was version 1.11… but two different version 1.11s existed. The one not flying had the proper command script to handle the thruster to reaction wheel changeover. Astonished? I sure was. From the report: "Flight code was stored on a network server in an uncontrolled environment." Version control is not rocket science!

Clementine

NEAR is by no means the only space probe to suffer from software issues; recent failed Mars missions come immediately to mind. Another asteroid-rendezvous spacecraft experienced a somewhat similar failure in 1994. Clementine, which very successfully mapped much of the moon from lunar orbit, was supposed to autonomously rendezvous with near-Earth asteroid 1620 Geographos. A software error caused a series of events that depleted the supply of hydrazine propellant, leaving the spacecraft spinning and unable to complete its mission.

A sequencing error triggered an opening of valves for four of the vehicle's 12 attitude control thrusters, using up all of the propellant. No fuel, no go.

Unfortunately, I've been unable to obtain more detailed information about the nature of the software error. However, there's an enticing – and as yet unavailable – reference in the appendix of the NEAR report to a memo called "*How Clementine Really Failed and What NEAR can Learn*". Is it possible that NEAR's software failure had been anticipated 4 years earlier?

NASA published a report on the mission (reference 2) that mentions the failure but does not delve into root causes. Clementine was a technology demonstrator operated by the Ballistic Missile Defense Organization; NASA was a partner, not the main force behind the mission. Reference 2, though short on firmware details, does delve into the human price of a schedule that's too tight. Here are a few quotes; there's not much one can add!

"The tight time schedule forced swift decisions and lowered costs, but also took a human toll. The stringent budget and the firm limitations on reserves guaranteed that the mission would be relatively inexpensive, but surely reduced the mission's capability, may have made it less cost-effective, and perhaps ultimately led to the loss of the spacecraft before the completion of the asteroid flyby component of the mission."

"The mission operations phase of the Clementine project appears to have been as much a triumph of human dedication and motivation as that of deliberate organization. The inadequate schedule… ensured that the spacecraft was launched without all of the software having been written and tested."

"Further, the spacecraft performance was marred by numerous computer crashes. It is no surprise that the team was exhausted buy the end of the lunar mapping phase."

Ariane 5

In May of 1998 I described the 1996 failure of Ariane 5, the large launch vehicle that tumbled and destroyed itself 40 seconds after blast-off. Since then more information has come to my attention (see reference 3).

Shortly after launch the Inertial Reference System (SRI, the apparently scrambled acronym a result of translation from French to English) detected an overflow when converting a 64 bit floating point number to 16 bit signed integer. An exception handler noted the problem and shut the SRI down. Due to the incredible expense of these missions (this maiden flight itself had two commercial spacecraft aboard, each valued at about $100 million) a back-up SRI stood ready to take over in case of the primary's failure. SRI number 2 did indeed assume navigation responsibility… but it ran identical code, encountered the same error, and shut down as well.

Why did the overflow occur? This code had been ported from the much smaller Ariane 4. According to the report "…it is important to note that it was jointly agreed [between project partners at several contractual levels] not to include the Ariane 5 trajectory data in the SRI requirements and specification." Clearly, a decision doomed to failure. Here's a case where the firmware was in fact perfect – if perfection is measured by how well the code meets the spec. Again, "The supplier of the SRI was only following the specification given it."

As with NEAR, Ariane's crash resulted from a series of coupled events rather than any single problem.

The exception was largely a result of poor specification. But designers did realize that some variables might go out of range; in fact they specifically wrote code to monitor four of the seven critical variables. Why were three left exposed? An assumption was made that physical limits made it impossible for these three to overflow (an assumption that proved expensively faulty). Further, a target of 80% processor loading meant checking all calculations would be prohibitively expensive.

But the exception itself didn't cause Ariane's crash. When both SRIs failed, they did so gracefully, and even returned diagnostic data to the vehicle's main computer that indicated the flight data was invalid. But the main computer ignored the diagnostic bit, assumed the data was valid, and used this incorrect information to guide the vehicle.

As a result of trying to use bad data, the computer commanded the engine nozzles to hard-over deflection, resulting in the tumbling and destruction of the rocket.

To complicate the picture further, the floating point operation that overflowed was a calculation not even required for normal flight operations. It was left-over code, a relict of the firmware's Ariane 4 heritage, code that had meaning only before lift-off.

The review board also noted that, though testing of the SRI is hard, it's quite possible and (gasp!) maybe even a good idea. "Had such a test been performed by the supplier or as part of the acceptance test the failure mechanism would have been exposed."

To summarize: poorly tested code that should not have been running caused a floating point conversion error because the spec didn't call for an understanding of real flight dynamics. In an effort to keep processor loading low the variables involved weren't monitored, though others were. Two redundant SRIs running the same code performed identically and shut down. The main computer ignored the SRI "bad data" bit and try to fly using corrupt information.

Another interesting tidbit from the report: "… the view had been taken that software should be considered correct until it is shown to be at fault." This is the rationale behind using identical code on redundant SRIs. It does beg the question of why insufficient testing to isolate those potential software faults occurred.

Conclusion

My embedded disaster collection grows daily. I expect, as embedded systems become ever more pervasive, that there's no end in sight to the firmware crisis we'll all experience.

Several common threads run through many of these stories. The first is that of error handling. Look at Ariane: when the software failed, it properly set a diagnostic bit that meant "ignore this data". Yet the main CPU blithely carried on, instead ignoring the error bit.

Inadequate testing, too, appears repeatedly as a theme in disasters. The NEAR team had simulators and prototypes, but these test platforms worked poorly. Their fidelity was suspect, leaving the engineers to wonder, when problems surfaced, if the simulator or the code was at fault. Ariane, too, had poor simulators and thus only partially tested software. On Clementine it appears that some code was not tested at all.

Interprocessor communications is a constant source of trouble. Though I'm a great believer in using multiple CPUs to reduce software complexity and workload, when too much comm is required problems result. NEAR's computers ran into race conditions. Ariane's error bit was unrecognized.

Those who don't learn from the past are sentenced to repeat it.

Reference 1: *The NEAR Rendezvous Burn Anomaly of December 1998*
http://near.jhuapl.edu/anom/index.html

Reference 2: *Lessons Learned from the Clementine Mission*, NASA/CR report 97-207442.

Reference 3: *Ariane 5, Flight 501 Failure, Report by the Inquiry Board*.
rk.gsfc.nasa.gov:80/richcontent/Reports/Failure_reports/Ariane501.htm