In a scene from the movie *Office Space*, a trio of disgruntled programmers discover that a software glitch will expose their money laundering scheme:

PETER: You said the thing was supposed to work.
MICHAEL: Well, technically it did work.
PETER: No it didn't!
SAMIR: It did not work, Michael, ok?!
MICHAEL: Ok! Ok!
SAMIR: Ok?!
MICHAEL: Ok! Ok! I must have, I must have put a decimal point in the wrong place or something. Shit. I always do that. I always mess up some mundane detail.
PETER: Oh! Well, this is not a mundane detail, Michael!

Unfortunately, major consequences stemming from small software problems are real. On February 25, 1991, during Operation Desert Storm, an Iraqi missile exploded in Saudi Arabia, killing 28 U.S. army members. This kind of event wasn't supposed to happen because the United States had armed itself with the MIM-104 Patriot surface-to-air missile, capable of intercepting and wiping out ballistic missiles midair. The system failed because a software bug (using a 24-bit floating point number to represent the fraction 1/10) resulted in a very small roundoff error in the missile's clock (0.000000095 seconds). The small error accumulated ten times per second, snowballing into a major problem. At the time of the attack, the roundoff error had become 0.34 seconds - enough to get a missile trajectory wrong by almost half a mile.[1]

Today, the Materials Project electronic structure database has registered 9000 users. More importantly, by tracking citations we can see that they are actually using the data in their research and publications. This is certainly what we were hoping for, and we hope it continues. Yet, every time I see a Materials Project data point, phase diagram, or band structure in a paper, I cross my fingers that the results we've passed on are correct.

I'm paranoid probably because I (and others on the Materials Project team) spend large chunks of time fixing problems in the Materials

Project data. A search for the word “bug” in my email gives ~500 results in the past year (and there are additional "issues", "problems", and "errors"). Trying to exterminate the Materials Project's bugs can be somewhat maddening - the past few years have demonstrated that the infestation *always* returns, usually based on something that appears innocent at first glance. For example, on multiple occasions, code that incorrectly set (or failed to set) a *single* input tag ruined tens of thousands of dollars worth of computing and several weeks of work. Currently, we're struggling to find out whether old bugs in a crystal structure matching code may have affected what we've computed and potentially any of the reported results; there have been hundreds of thousands of comparisons made using this code, and the results were used to set up later calculations and analyses.

We're not alone in our problems; as summarized by Jeff Atwood and Steve Krug[2] for programming projects in general - *you can find more problems in half a day than you can fix in a month*. However, in high-throughput calculations bugs are almost guaranteed to multiply because each piece of code is used to set up thousands of calculations.

To help combat software errors, organizations such as *Software Carpentry* have started banging the drum for better software practices in the sciences. These efforts are timely: recently, 5 high-profile papers (including 3 *Science* papers) were retracted to due to a small bug in a computer program used to determine biological crystal structures.[3] I wish *Software Carpentry* luck and applaud their efforts to hold workshops, provide resources, and start a conversation around this topic. At the same time, I wonder if they'll actually reach their target audience. In particular, the bullet-pointed strategies they've published[4] ("put everything that has been created manually in version control"; “every piece of data must have a single authoritative representation in the system") can ring like parental nagging ("eat your vegetables!"; "brush your teeth twice a day!"). Perhaps a better technique is to encourage good software practices by making it easy to do the right thing. For example, the web site Github has made it fun to use the complicated *git* version control

system and furthermore makes it easy to adopt practices like issue tracking in a more natural and fluid way than explicit instruction. Perhaps more importantly, many problems stem from miscommunication, not from failure to use the right tools or procedures. Going back to the Patriot missile software (presumably developed by well-funded software professionals, not scientists), the story doesn't end with the round-off bug. Rather, the software was intended and tested for use in mobile applications and for tracking slower-moving targets in which the clock bug was not a factor.

However, the software was later repurposed (inappropriately) for a more demanding application.[5] Similarly, for Materials Project, one of the major dangers is not necessarily the correctness of the data itself but whether our users understand the limitations of computed data when applying it to their problems.

Similarly, developing and launching code under time pressure is responsible for many of the small mistakes that accumulate over time. Most organizations only learn the hard way that catastrophes are often rooted in peccadilloes. Astronaut Chris Hadfield summarizes it well[6] when describing historical tragedies in the space program - that the bigger the ambition of your project, the more important it is to pay attention to the smallest details:

"But when astronauts are killed on the job, the reason is almost always an overlooked detail that seemed unimportant at the time... The Russians began wearing pressure suits for launch and landing only after a ventilation valve came loose and a Soyuz depressurized during re-entry in 1971, killing all three cosmonauts on board, likely within seconds. Shuttle astronauts started wearing pressure suits only after Challenger exploded during launch in 1986. In the case of both Challenger and Columbia, seemingly tiny details— a cracked O-ring, a dislodged piece of foam— caused terrible disasters. This is why, individually and organizationally, we have the patience to sweat the small stuff even when— actually, especially when— pursuing major goals. We've learned the hardest way possible just how much little things matter."

Footnotes:

[1] I'm getting most of information on the Patriot missile from this site.

[2] Jeff Atwood suggests prioritizing these bugs based on user complaints.

[3] The journal Nature has even started a "Digital Toolbox" section; an article summarizing the coding errors is here.

[4] See the paper "Best Practices for Scientific Computing" by Wilson et al.

[5] In addition to the Patriot missile example, the Computational science: ...Error article from footnote #3 also includes an example where a scientific code is used outside of its intended parameter range, thereby leading to erroneous results being published.

[6] This excerpt is from Chris Hadfield's book, "An Astronaut's Guide to Life on Earth: What Going to Space Taught Me About Ingenuity, Determination, and Being Prepared for Anything".