Do Kwon
CS 181W
Paper 1

Lessons Gained from a Spacecraft's Loss

NASA lost a spacecraft called the Mars Global Surveyor (MGS) November 2006 when a software failure triggered a series of events that eventually led the spacecraft to lose its power. As to be expected, the loss of a spacecraft is an expensive matter. The loss of the Surveyor cost a little shy of two hundred million dollars, a substantial amount even for NASA. Many have attempted to assign this loss to bad luck, claiming that the circumstances leading to the failure could not have been foreseen. They argue that the software errors of the MGS could have been contained had the spacecraft been placed in a slightly more favorable orientation, or had atmospheric conditions been kinder. (McKee) However, discussing the probabilities of the consequences without owning up to the deep-seated structural causes of the error is not acceptable. The loss of the Mars Global Surveyor revealed far more than a one-time human mistake; it exposed an error-prone culture of negligence and ineptitude pervading many NASA projects.

Synopsis

Ironically, the Mars Global Surveyor was lost due to a misuse of (computer) memory. According to NASA, a software upgrade sent up to the spacecraft overwrote two wrong memory locations on the onboard computer. Unfortunately, the corrupted memory locations turned out to be rather important, as they governed the rotation of the craft's solar array. So when ground controllers commanded the solar array to reposition itself five months later, the first corrupted memory caused the onboard computer to rotate the craft's solar array further than its physical limit. The onboard-computer then went into a special error mode, re-orienting itself in space. In its new orientation, one of its batteries became directly exposed to the Sun, quickly overheating. (NASA Report)

The spacecraft interpreted the battery's excessive heat levels as an indication of overcharging, so it gradually cut off the charging process. The remaining battery could not provide sufficient power to maintain the craft, and both batteries ran down within twelve hours of the original problem. Ground control has not been able to communicate with the spacecraft, and the Mars Global Surveyor still (presumably) roams outer space.

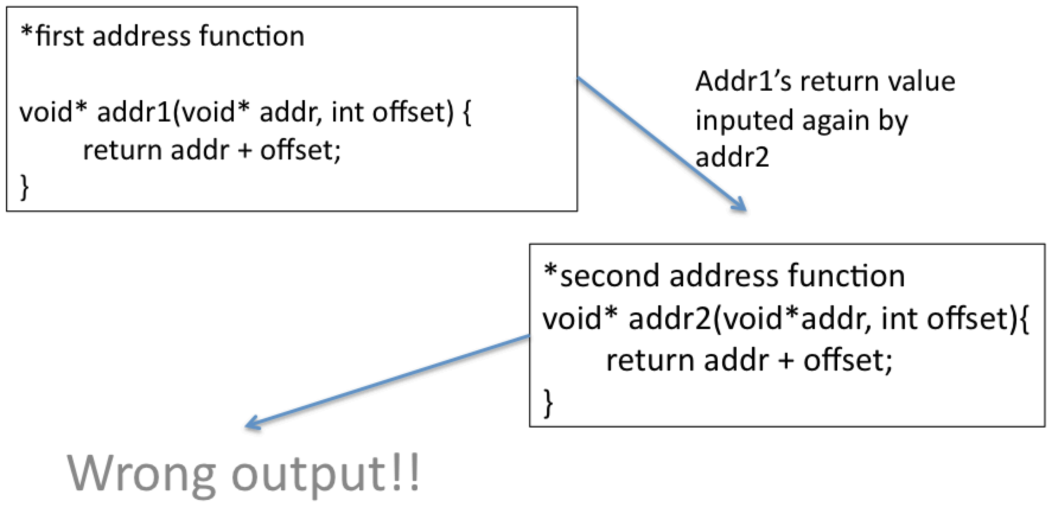
Throughout the Surveyor's failure, ground controllers were not aware of the trouble beleaguering the spacecraft, as the second memory overwrite effectively disabled on-board communications with Earth. The memory corruption caused the spacecraft's communications antenna, which is used to transmit messages with Earth, to point away from the Earth. The NASA report following the accident states, "Communication from the spacecraft from the spacecraft to the ground [became] impossible, and the unsafe thermal and power situation could not be identified by the MGS's ground controllers." (NASA Report)

Who to blame

The loss of the Mars Surveyor spelled $136 million out of the US taxpayers' pockets. Following the failed launch of Europe's Arianne 5, the Surveyor's loss marks the second most expensive software accident in history. (Leveson) That cost could become higher if the Surveyor's remains destructively collide with an operational satellite or spacecraft. If the Surveyor does not drift far away from its previous orbit, the possibility of collision is not a remote one.

The initial bug was introduced by a lack of coordination amongst the programmers maintaining the software. Two different engineers wrote different components of the code for the faulty software upload, which was intended to optimize redundant control systems for controlling the antenna in error mode. However, the combination of these engineers' code transformed an inefficient control structure into a faulting one.

Figure 1. Faulting memory calculation by redundant computations of memory addresses.



```
*first address function

void* addr1(void* addr, int offset) {
        return addr + offset;
}
```

```
*second address function
void* addr2(void*addr, int offset){
        return addr + offset;
}
```

As can be seen in Figure 1, the specifications for the software update relied on a helper function to calculate the address of an important memory location, the offset to which was stored in a constant. However, the address-getting helper function was implemented by both sections of code written by the different engineers, ultimately returning an address of base + offset * 2 instead of the correct address base + offset. (NASA Report)

NASA's post accident report revealed that the fault for the accident lied not with the Global Surveyor team, but itself. The report concluded that the Mars Global Surveyor team followed existing procedures, but that those procedures, designed and implemented by NASA, were simply not enough. NASA discovered that it had no requirements for regular code reviews, and through more than a decade of the Mars Surveyor's existence, only two partial reviews were conducted. (NASA report) NASA was also regularly in the practice of contracting software updates to firms that were not involved in the original design of the software. (Leveson) In its report NASA concluded that such inconsistency in contracting resulted in many incompatibility errors, costing heavily in the failure of many missions, including the *Ariane 5.*

A string of software failures in outer space, caused by repetitively complacent engineering and management practices, indicate an intractable culture of complacency in NATO's culture. The MGS accident is succeeded by a long chain of previous failures; the Arianne 5 launch failure (insufficient testing of floating point conversion), Mariner 1, Mars Observer, and the Mars Polar Lander are the most prominent costs of those mistakes. The report admits that the pressure of meeting the cost and schedule goals resulted in an environment of increasing risk in which too many corners were cut in applying proven engineering practices and in the checks and balances necessary for mission success. "Lack of adequate risk identification, communication, management, and mitigation compromised mission success." (NASA report)

Cheap and Quick

The Global Surveyor was in operation four times longer than its intended lifecycle, and its longevity may have contributed to its downfall. Just like any other machine, a spacecraft gets much harder to fix and maintain with its age. Every software upgrade becomes progressively challenging because it has to be tested for compatibility with source code that has been developed over many years by different programmers. (McKee) Updated libraries and system components have to be taken into question. Worst is, the coders that originally wrote the code are no longer around, so the maintenance is often left to newcomers, increasing the complexity of maintenance.

Like many other long-lived programs, the Surveyor saw much of its budget and staff slashed to ensure the economic viability of its missions. (McKee) Software maintenance was heavily reduced as a result of these cutbacks. Ironically, the long-enduring success of the Global Surveyor led to the complacency that led to its downfall. It consolidated the notion that the software on board the MGS "worked," that the wheel need not be reinvented, and that the regular maintenance of its software was no longer necessary. Success is ironically one of the progenitors of accidents when it leads to overconfidence and cutting corners or making tradeoffs that increase risk.

Recommendations

Regardless of whether budget cutting and negligence in software maintenance was directly responsible for MGS's failure, such practices certainly made it more likely. Practices that increase risk in our explorations in outer space should be discontinued. They make explorations too dangerous, and eventually racks up a heavy cost to the taxpayer that could easily have been prevented.

NASA needs to change when sending spacecrafts worth billions to outer space:

Regular reviews of software need to be conducted. The software failure aboard the MGS did not happen because a software component stopped working or did not work according to specifications, but because an interaction amongst different software components resulted in the wrong output. The only way to make sure that different software components in a highly complex system like that of a spacecraft is regular, rigorous code review. There was a five month gap from when the bug was introduced (June 2006) to the software failure aboard the MGS (November 2006); in this time there was ample opportunity for ground control to have detected the corruption in the data.

Only by implementing correct practices to mitigate risk in outer space software can we begin to be safe from needless software failures. Complexity and errors are intrinsic components of software, but we can defend against much of their harms given proper safety procedures. In very few areas is the implementation of such safety procedures more urgent than in outer space explorations, where the costs of error are too high to risk.

References

"NASA - National Aeronautics and Space Administration." *NASA*. Web. 29 Apr. 2012. <http://www.nasa.gov/mission_pages/mgs/mgs-20070413.html>.

Leveson, Nancy. *The Role of Software in Spacecraft Accidents*. Diss. MIT, 2006. Cambridge: MIT UP, 2006. Print.

McKee, Maggie. "Software 'fix' Responsible for Loss of Mars Probe." *Newscientist.com*. Web. 30 Apr. 2012. <http://www.newscientist.com/article/dn11608-software-fix-responsible-for-loss-of-mars-probe.html>.