Geoff Colman

CMSI 401

“The Lessons of ValuJet 592”

A commercial jet bound for Atlanta from Miami crashed in the Florida Everglades in May 1996. Testimony and recordings of the pilots’ conversations with Miami control show the increasing distress of, and difficulty the pilots had controlling, the aircraft due to a fire onboard. A convoluted sequence of systemic events led to the tragic incident. The airline had a poor safety record due to a history of aggressive of cost-cutting with poor training and outside contracting; it had recently contracted workers to replace the emergency oxygen generators in 3 aircraft. Through a confluence of improbable failures attributed to such predictable human causes as incompetence (the mechanics mislabelled the canisters and neither the ramp agent nor the co-pilot recognized the hazards); laziness (the mechanics did not properly secure the canisters with the safety caps, but signed off on the work nonetheless); and a broader moral hazard (a lesser likelihood one will respect rules in the absence of direct personal consequences), the hazardous, improperly-prepared oxygen generators were placed on the ill-fated plane - ultimately resulting in the catastrophic fire.

The NTSB investigation into the accident rapidly zeroed in on the fire in the cargo area. They placed culpability with the outside contractor (SabreTech) that mishandled the canisters, ValuJet itself for its systematic disregard for safety (demonstrated on numerous prior occasions, and specifically here in the confusing and inconsistent language even their own mechanics couldn’t follow), and a criminally complicit FAA that had been infiltrated by industry insiders. While the aftermath saw a positive outcome in a housecleaning of the FAA, harsh punishments for ValuJet, and new safety measures, the author cautions that this accident was one with multiple distal causes distributed across a complex system that happened to coalesce into a catastrophic failure - what he calls a “system accident.” In his view, these accidents inevitably arise from the inherent complexity of the systems controlling dangerous technologies.

A few takeaways from this article stand out to me. The first, and most obvious, is something that should be relatively intuitive: the more complex a system or product/service becomes, the less likely any one agent or collection of agents will be able to effectively manage it. In the specific context of software development/engineering, the Pilone and Miles book references this indirectly at several points in the chapters on team and project management: keeping every member of a team on the same page becomes increasingly difficult with each additional team member. Indeed, this is the underlying logic of the old aphorism “too many cooks spoil the broth,” and presumably one of the driving motivations behind such tools as Slack and Git. Not only does this phenomenon dramatically increase the likelihood of breakdowns in communication at each step of the development process, it also seems to imply that a team is only as productive as its least effective member: if other team members have to hand-hold a new or inexperienced member on top of their own tasks, it is all the more difficult for the team as a whole to assure quality at every step.

Closely related to this (and similarly intuitive) is the idea that, if a system becomes too unwieldy, it can end up encouraging exactly the problems its safeguards aim to prevent. Put another way: the more complex a system is, the more individual points there are (in both space and time) at which things could possibly go wrong. The old engineering maxim “KISS” (“Keep It Simple, Stupid”) informally expresses more or less this same idea. It isn’t difficult at all to imagine a software project so bloated with classes and files (and nested classes and directories) that it becomes exceedingly difficult to debug, and there are certainly myriad examples thereof. One of my favorite examples of a software horror story occurred when the Motorola Droid phone was first released in 2009. After a month or two, a bug came to light that was causing the camera’s autofocus feature to alternately silently fail and work normally on a 24.5-day cycle due to a rounding error having to do with a system timestamp.[[1]](#footnote-0) The author of the referenced article makes a joke about how troubleshooting that ticket must have been like “finding a needle in a haystack.”

This idea is often conflated with Murphy’s Law: “if something can go wrong, it will.” Like most people, Mr. Langewiesche equates these concepts, but he also draws a separate conclusion that can be interpreted as a corollary thereto. If a failure or error occurs - even something important, to a catastrophic degree - the probability that it will immediately result in a catastrophe is comparatively small. Often these failures are relatively minor in the system as a whole, and/or precautions exist along the way to catch or or prevent them from becoming worse; the best practices of software engineering, for example, specifically demand graceful error handling. Since the consequences of the error or failure are not immediately felt in these circumstances, observers perceive that the system can “get by” with less stringent design and maintenance, and a lesser standard is normal and acceptable. This is closely related to the aforementioned moral hazard: without an obvious incentive for greater rigor, agents become lax in their decision-making and maintenance regimes - and conceivably even their designs. This can work well for an arbitrary length of time - until that one time it does not, and (in an absolute worst-case) people die as a result. The example of the bug in the Raytheon missile defence system comes to mind: while the system was not specifically designed to be used in the manner in which it was, it worked “well enough” as (improperly) deployed for a while. The commanders were evidently complacent with their implementation, in spite of it being counter to recommendations, and people died as a result.

Following as a logical consequence from this, humans are frequently the weakest links in the chains of systems. In the specific example of ValuJet 592, there were *numerous* points in the chronology of the oxygen canisters (the mechanics, the shipping clerk, the ramp agent, the co-pilot) at which a reasonable, conscientious individual simply performing their job well and with pride could have prevented the tragedy from occurring. The system ostensibly had provisions (the author’s criticisms thereof notwithstanding) to ensure that such incidents shouldn’t arise, but - at almost every step of the chain of custody - incompetent and/or lax employees circumvented them. Examples of these sorts of failures are downright plentiful. A similar case of user error stemming from laziness was behind the x-ray technician crushing the elderly patient on the imaging platform. In 1983, Korean Airlines flight 007 from Anchorage to Seoul inadvertently strayed into Soviet airspace (which resulted in its being shot down by a Russian fighter pilot) because the flight crew weren’t paying close enough attention to which of two primary autopilot system settings were engaged.

Engineering software on a large-scale comes with a variety of dangers associated with complexity. While (fortunately for all involved) most aren’t “life or death,” some are; it is for this reason that it is extremely important to critically consider potential use-cases and user stories at all stages of development.

1. https://www.itworld.com/article/2765407/personal-technology/bizarre-droid-auto-focus-bug-revealed.html [↑](#footnote-ref-0)