ValuJet 592

Part 1) Synopsis

The Atlantic's article "*The Lessons of ValuJet 592"* examines the tragic crash of ValuJet Flight 592 in May 1996 and uses it as a case study in systemic failure. The plane crash, which killed all 110 people onboard, was not the result of a single "procedural" or "engineered" failure. Instead, it was revealed that there were many small oversights that interacted in unpredictable ways under a negligent system. The crash happened shortly after takeoff from Miami when an improperly handled chemical oxygen generator, stored in the cargo hold, ignited and caused an uncontrollable fire. As a result, the aircraft lost all electrical systems and filled with smoke, leaving the pilots hopeless. Within minutes, the plane plunged into the Florida Everglades. The article emphasizes that this disaster was not simply a pilot error, or the fault of one of the mechanics. Instead, it was the product of an interconnected chain of problems. These problems included the airline (ValuJet), its maintenance subcontractor (SabreTech), and the Federal Aviation Administration (FAA). Each piece of the problem made decisions that, taken alone, might not have caused a disaster. But when these issues are combined (mislabeling hazardous materials, failing to install required safety caps, and inadequate oversight), the results are completely catastrophic.

Leading up to the crash, the FAA had already flagged concerns about ValuJet. The budget airline was exploding in popularity and expanding rapidly as a low-cost carrier, and its safety record had been questionable. SabreTech cut corners in handling oxygen generators, treating them as ordinary cargo instead of hazardous materials. ValuJet itself failed to ensure adequate controls on its subcontractor, even though it was partially responsible for what went on its airplanes. Before pointing fingers, the article brings attention to the effects of a highly competitive free market, noting that the primary goal of any airline within this market is to deliver people as quickly and cheaply as possible. Safety, while considered for obvious reasons, is "never first, and it never will be"(3). This capitalist mindset has a palpable effect on the workers in this story too, as ValuJet was notorious for using underpaid temporary labor for everything from ramp agents to mechanics and pilots. While blame can easily be pointed at the most visible targets, the article urges readers to consider the ways that the system failed these people long before any sort of catastrophic failure.

The Atlantic article also draws attention to the responses surrounding the disaster. ValuJet executives promoted confidence and reassurance to the public about their safety, even as concerns were raised internally and externally. Additionally, the FAA gave public assurances that its oversight was sufficient. Despite the lies, this tragedy revealed how organizations can create false confidence and downplay the risks instead of openly acknowledging and addressing them. After the crash, blame was shared widely. The FAA restructured their regulations for transporting hazardous materials so something like this would never happen again. ValuJet was grounded, and then eventually merged with AirTran. SabreTech faced criminal charges for their negligence. Despite this, the article's main message is that ValuJet 592 was a systems accident, a failure that came from not one glaring mistake, but from many small errors across a network of organizations.

Part 2) Our thoughts and how the situation might apply to software engineering projects

The ValuJet accident highlights the importance of high-quality work and the dangers of fast production in any type of project. The ValuJet strategy prioritized profits and efficiency over safety, which led to aggressive cost-cutting measures being taken. For software engineering projects, we can learn from the issues that arise from cutting-corners.

The article talked about Diane Vaughan's idea of "normalization of defiance," where potentially dangerous practices become considered normal due to advantages in time/cost-saving. In ValuJet's case, this came in a broad plethora of forms. For a software company, one way this could manifest as an over-reliance on AI. Since training data is pulled from all sorts of sources on the internet, AI code can prove to be unreliable for hyper-specific use cases. If AI is used too heavily, it may end up causing catastrophic problems later down the line due to lack of oversight from a qualified engineer. Additionally, many companies rely heavily on AI during their hiring processes. This creates an environment where AI resumes are picked up by AI reviewers and everyone ends up with AI slop. Even if candidates do make it to the interview stage, those have started being handled by AI as well. However, candidates themselves can simply rely on AI to give them perfect answers to any questions that pop up during the interview. These circumstances dehumanize candidates and degrade the hiring process in the name of efficiency.

ValuJet prioritized fast company growth over the quality of their airplanes. In software engineering, this way of thinking comes in the form of minimal testing, relying on external libraries/frameworks, and rushed deployment pipelines. The FAA had been tasked with both promoting growth of the airline industry and enforcing regulation. But their focus on growth led to a lax approach when it came to regulation/testing. Similarly, if the members in a software engineering project prioritize fast deployment, safety and quality of the project has less priority. Without proper regulation through automated testing or peer reviews, broken code can easily be deployed unnoticed. An example of this is the recent 2024 CrowdStrike crashes that brought computer systems around the world to a halt, costing companies billions of dollars. Without comprehensive testing, something like a faulty update can easily slip out into the world and cause enormous amounts of damage.

Another way ValuJet cut corners was through outsourcing without much oversight from SabreTech. Not only did they rely heavily on temporary workers and outsourced labor, but they failed to properly train these people. This lack of ample personnel is mirrored by the software industry today. In the past few years, massive layoffs have occurred within the software industry as companies bounce-back from COVID protocols. As a result, from anecdotal evidence gathered during internship hunting, once cushy software jobs have become overworked and understaffed positions that struggle to meet deadlines. Tightened schedules means less time to properly test, code review, and document the software. Similarly, if rushed software engineers solely rely on external libraries, frameworks, or cloud providers without research/training, problems could arise. These problems could range from poor optimization to security vulnerabilities within open-source packages or unreliable cloud providers that cause client downtime. Importantly, these issues are often compounding, with early symptoms being overlooked or “hacked” around until major issues arise.