Danico Pidlaoan
CMSI 401
Software Engineering Laboratory

System Accidents: Lessons of ValuJet

"The Lessons of ValuJet 592" by William Langewiesche documents and analyzes the "system accident" that caused the deadly crash of ValuJet Flight 592. Rather than a "procedural" accident that refers to human error such as a pilot flying through a thunderstorm, or an "engineered" accident that might refer to an aircraft rudder malfunction do to a design flaw not tested in certain conditions, Langewiesche describes a "system" accident as an accident stemming from a complex combination of systemic and organizational issues ranging from miscommunication, the limited regulation of the industry, and complacency of multiple contractors. Many of these issues and failures are still relevant and prevalent in the software development process and the tech industry, which may lead to similar failures.

ValuJet 592 caught fire in the air shortly after taking off from Miami. The toxic fumes and smoke caused the pilots to lose control of the aircraft, which led to the aircraft plummeting down into a swamp and killing all aboard. Further investigation revealed a series of events and faults that caused the fatal accident. ValuJet placed an order to replace the emergency oxygen tanks, but used an outside company called SabreTech to do the job and provided specific procedures to do so. However, before shipping the canisters, SabreTech employees failed to place required plastic safety caps over the firing pins since they had none in supply. In a rush to complete paperwork, the mechanics signed off. A manager at SabreTech shipping and receiving department labeled the boxes as containing "empty" oxygen canisters, and included tires to the order. When delivering to Flight 592, the ValuJet ramp agent accepted the material, even though federal regulations forbade him to. The cargo was then loaded into the cargo hold in a

very unstable arrangement. The first oxygen generator ignited at some point between then and takeoff, which caused the aircraft to burn mid-flight.

Two factors that worked hand in hand were cost cutting and complacency. ValuJet opted to cut costs by hiring an outside agency to handle the oxygen tank orders, an organization that was notorious for unstable employment, few protections, and long working hours. This led to complacency among the mechanics and management, saying the product and labeling was “good enough” as they signed off on explosive and toxic canisters that were mislabeled and lacked the proper safety caps. Relating to software development, many organizations and engineering teams opt for cost-cutting options that might be below standard. This can be embodied through outsourcing teams or taking shortcuts to deploy code and software as soon as possible. This can also lead to unthorough QA testing and UA testing, as companies say the code is “good enough” and sign off on the release. Consequently, during multiple integrations of code, one part of the system may fail, such as an edge case that was not captured by the tests. This can cause a chair reaction that causes other parts of the software to fail, ending in catastrophe. Software can deal with so much sensitive data such as personal information, finances, and connections to hardware that can cause serious harm if malfunctioned. Sometimes, software engineers may recognize the risk, similar to NASA’s Challenger disaster, as the engineering teams acknowledged the risk of leaking O-rings but put aside since the agency had gotten away with launching O-rings before. However, they may see that they have gotten away with a flaw in the code design before and continuously integrate the risky design in future products, allowing for what sociologist Diane Vaughan calls “the normalization of deviance”.

Another factor is organizational systems that are so complex that correct procedures, communication, and product implementation get lost in translation. As Langewiesche says, “Some of the riskiest technologies require organizations so complex that serious failures are virtually guaranteed to occur. Those failures will occasionally combine in unforeseeable ways.... They expose hidden connections, neutralize redundancies, bypass ‘firewalls’, and exploit chance circumstances that no engineer could have planned for.” Similar to how the ordering, manufacturing, and shipment of the oxygen canisters had to go through multiple checks and procedures through different teams that are failure prone, the same must unfortunately be done with software development. The business must work with customers or clients in establishing the requirements for the software, which must be communicated to the engineering teams. Product managers and tech leads must come up with functional and technical requirements. Project managers then work with the engineers to break down tasks, schedule, and delegate to the engineering team. Then, engineers must refer to the technical and functional requirements, as well as their own skills and judgement, to build their own specific piece of the software. Their code must then be integrated with the rest of the software’s code, and heavily QA and UA tested. With so many moving parts, if one piece of this process fails, it can cause a chain reaction that can cause the entire system to break, and can have damaging ramifications to both clients and the business.

Another factor related to the above that contributes to system failures is what the author calls “engineerspeak”, or the language ValuJet used to communicate their requirements and instructions to SabreTech constituents. Langewiesche says, “The language problem lay on… the English-speaking engineers, literalists, who wrote the orders and technical manuals as if they were writing to themselves.” In the software industry, effective communication is just as important as technical prowess. Engineers must write readable code that communicates their

logic, and also write effective documentation, to ensure any other engineer who takes on the task knows their logic and reasoning. Writing technical requirements means that the writer must take into account the language they use, and make sure they tailor to the audience they are writing for. One must communicate to contractors differently than to someone within the company and team. Additionally, one must communicate directions and requirements differently to someone from the business side than someone on the engineering side of the company, since they carry out different function in building the same product. Requirements should also not be so complicated and technically thorough that it confuses the reader and promotes the complacency previously described.

One last factor the article discusses is the regulation of the airline industry. The software industry must also be regulated as well to make sure companies are staying compliant to federal standards and regulations, ensure the safety of users, and ensure ethical choices are being made. Examples of this scrutiny include Facebook and its data usage, as well as the government stepping in when a major hack of Sony or a national bank occur. Although we can provide so many different precautions to ensure the safety of the software we create and use, system failures occur from a combination of complex factors that may not be preventable. What we can do is analyze organizational pitfalls that may cause these problems to reduce the chances of these failures in the future.