Francesca Kang
CMSI 401
Due: 2 Oct 2019

ValuJet 592

In May of 1996, Valujet 592 crashed, killing 110 people on impact. In the search for what went wrong, investigators realized the real story is why things didn't go wrong sooner. The fatal crash was the result of multiple factors coming together to cause a system error. Procedural errors are blamed on individuals not following necessary steps such as avoiding thunderstorms. Engineered errors are attributed to physical parts that fail to meet a performance standard. System errors are failures on a larger scale, where pieces and parts of a larger network fail at the same time resulting in a noticeable, detrimental error. ValuJet's values placed priority in a corner-cutting form of efficiency. Without proper "checks" and tests, their (literal) downfall was inevitable. In a previous version of this essay, I focused on the overall fact of trying to "better" the system leading to failure. In this paper, I will focus on the importance of location when discussing: lapses in communication between the pilots and control tower during critical moments, highly flammable active oxygen-generating canisters, necessary and absent special safety caps, involved but uncommitted employees, damaged important parts of wires and electrical panels in the plane. ValuJet's company as a software would consist of methods and classes that work to use information to take other data from point A to point B. A series of analogies will show how the location of proper "checks" and tests would have prevented these errors:

1. Communication lapses in software aren't always "fatal" errors that cause system crashes because like many math problems, the right answer can still be derived doing wrong work. A plane has successfully landed somewhere it's technically not supposed to with no fatalities (Hudson River, 15 Jan 2009). After recovery, those people could get back on a plane and go wherever. That airplane's intended destination was not the river, however that landing would barely be noticed in the software version. For example, John sends a message to Tim. Tim says that he never got it, so John resends it. Tim gets that message and they go about their day as usual. This could happen multiple times. Although it would be annoying, resending a message in the scope of their regular workday would not be a big deal. In the ValuJet scenario, the people crashed and died. There was no opportunity to resend a plane full of people. For example, John sends a message to Tim. The message never went through AND the contents of that message were not recoverable to be sent again. That would be a major source of frustration for Tim and John, for which they would blame the software. The software could have been set up to preserve the message no matter what. If the internet connection cuts out, the message would still be on its way to Tim. The delivery would

happen as soon as connection is reestablished. Likewise, the plane should have been able to head towards a safe landing, without communication. There were other factors that prevented that in the ValuJet situation which will be addressed next.

2. The highly flammable active oxygen-generating canisters were placed in the front cargo hold. This location error doomed the entire plane. If the flammable canisters were placed in a back cargo hold, some or all of the plane could have been saved. The important electric panels and wiring wouldn't have melted. The cockpit wouldn't have filled with smoke. The pilots would have been able to see and have physical control over the plane. The software version of this error could be a glitchy method placed at the "front" of a user's access. For example, John runs the program and opens up the main menu. She selects an option and what was supposed to be an artistically-designed fade-to-black transition ends up crashing and staying an empty screen instead of loading the new page. If that fade-to-black was placed after that page, John could still use the software. The crash would occur after she made her menu selection and used that portion of the program. It would just be a pain to reload the program in order to return to the menu each time.
3. The safety caps of the canisters were completely absent. This could be equated to a test being completely absent in the software. The test would check to make sure that the transition sends the user to a usable page, otherwise it wouldn't run. John probably wouldn't notice that the page loaded immediately after her selection without a fancy transition. The faulty transition would sit unused like flammable canisters left on the ground.
4. The managers and employees make for the easiest analogy. In a robotic fashion, they did what they were supposed to without extra thought. The boxes of canisters needed to be shipped, so they were. The safety-cap requirement was checked off without a test by someone whose job was checking things off. The boxes were labelled "empty" as instructed. The boxes were loaded by designated employees. The hazardous materials bypassed the ramp agent. All the "warning" signs around the warehouse desensitized the workers. They were like methods that were made to run even with "test failures", which makes the tests useless. Having just one committed employee whose responsibility was to do any of the proper checks would have prevented the canisters from exploding in the front cargo hold. That person could have stopped the boxed canisters from ever making it into the airplane or specifically the front cargo hold, asked if the canisters were really empty before putting it in boxes without safety caps, or checked if the safety caps were given with the non-empty containers. It makes no sense to check off safety caps and then ask a system if

there are safety caps, because the answer would be, “Yes, safety caps are checked off.” Instead, the test should have come first: “Are there safety caps? No? Ok. Don’t check it off then.”

ValuJet 592 was a great example of failures occurring in fatal locations within a system. It is a tragedy that so many errors occurred at once to create a huge, unsalvageable crash. For this reason, in both software and real-life systems, it is never a good idea to assume that everything is okay because things haven’t crashed and burned yet.