Alexia Filler
CMSI 401
ValuJet Article Summary and Analysis (Paper 1)
10/2/19

"The Lessons of ValuJet 592," an article written for the *Atlantic* by William Langewiesche, discusses events that led to the disastrous end of this 1996 ValuJet flight. This collection of events and general protocol oversights caused what Langewiesche refers to as a "system accident," meaning that the crash occurred as a result of several smaller problems within the airline safety process. The accident occurred due to a fire originating in the cargo hold of the plane, which caused electrical failures in addition to producing what was probably a thick, black, potentially toxic stream of smoke that crept into the cockpit and cabin. Through the process of investigating the scene of the crash as well as interviewing members of the extensive team associated with putting the plane in the air initially, the National Transportation Safety Board (NTSB) uncovered many factors that contributed to the accident. To start, ValuJet contracted mechanics from other companies, since temporary, contracted mechanics cost less than permanent mechanics. These mechanics were instructed to put safety caps on oxygen tanks that were being transported in the cargo hold of this flight, but the caps were nowhere to be found, and the reason for having them in the first place was lost in the somewhat confusing language of the protocol manual. The mechanics signed off on the tanks being transported without caps, and those in charge of giving the packages clearance to be put on the plane did not recognize the tanks as potentially hazardous. This combined with Federal Aviation Administration's (FAA) seeming blindness regarding some of ValuJet's prior safety infractions outlines just how complicated and multi-leveled the aviation safety regulation system is and how this complication leaves room for errors that might seem impossible within such a rigid system.

This article concludes by saying that accidents are inevitable, especially since people have gotten more lenient with guidelines and “let things slide” if they see an instruction they do not understand. This is exactly what happened with ValuJet, when the mechanics signed off on the oxygen tank caps being attached when they were not, or when anyone and everyone involved in the shipping process failed to recognize potentially dangerous materials as they had been trained to do. When such little deviations from regulation can cause such destruction, it seems that constant, active attention to detail is required to prevent such disasters. However, there is never any way for an entire, multi-level system to ensure that such meticulousness exists within every level. This seems to be the point of the article: the larger and more complex a system or set of instructions is, the more potential for misreads or mistakes there is.

This concept has several connections and applications to the software development process, even though it may deal mostly with chemical and hardware shortcomings. Software development requires a multi-layer process not unlike the process of prepping a plane to fly. From the Needs Analysis to the Software Development Plan, all the way through to the launch and maintenance of a software, there are many steps to go through before a deliverable is produced. This requires the customer to lay out exactly what they want in acute detail, and then the software team must make sure that every specific request is met. In every step of the process, there is room for misstep. A developer could see a listed requirement as unnecessary, or he/she could sign off on a code review without really looking at it or giving constructive comments because he/she trusts that his/her fellow developer has done it right. But this is exactly the problem that occurred in the ValuJet disaster; each level of the system assumed that those before and after them had done their job correctly. The same risk and potential for accidents exist in the software development process for exactly the same reason.

Accidents in code come about *all the time*. Clicking on a particular button on a website might crash the page, or an update to an operating system might cause unexpected glitches. Even when code seems completely clean and is well-tested, there may be some very specific action that causes some part of it to break. The hope is that thorough testing and a rigid code review process can narrow the bugs down to the minimum possible, but, inevitably, a few will slip through. With as many bugs as can be witnessed in seemingly professional software and web services, it seems impossible to say the contrary! However, a benefit is that, in the software development process, many of the required steps are carried out by the same people, which decreases the chance of some important step being lost in translation (such as one team misunderstanding the jargon of another team, as was demonstrated to some degree in the ValuJet scenario). Also, even though the process of software development is multi-faceted, each step is generally quite clearly outlined (in the development style we are being exposed to in this class, anyway).

While, at this stage in our careers, we not exactly dealing with software scenarios that could result in life-threatening consequences as the result of uncaught bugs, the importance of being accountable in our work remains. The article emphasizes the risks associated with making a safety system *too* complex, as might happen when attempting to account for every possible error. It would be great if we could learn from our mistakes without having to make the mistakes in the first place, but that is simply not possible. The ValuJet accident helped ValuJet, the NTSB, and the FAA rethink their safety regulations and prevent other similar disasters from occurring, but only at the loss of 110 lives. This is certainly an expensive mistake to have made, and the way that we structure the process of developing software attempts to combat the chance of making such mistakes by carefully outlining the expectations and consequences of the project. The key seems to be to remain active and alert at all major stages of the project, so that catchable errors do not go unnoticed, especially since such errors could have considerable consequences as we grow into full software developers.

Word Count: 1021

Note to BJ: This was a cool article! It made me think about how the Chernobyl nuclear disaster was definitely also a result of system errors (have your read up on that? I'd recommend reading through the Chernobyl Wikipedia page... It's almost comical how every single decision that was made regarding the disaster was absolutely the wrong decision to make and relied on so many previous bad decisions). 😊