Henno Kublin

CMSI 401

Professor Johnson

October 7th, 2020

A Plane Crashes into Software Engineering

One afternoon in May of 1996, a commercial jet was spotted by a fisherman nearby. This jet was clearly on a crash course, as it was attempting to bank as it was nearly completely horizontal, plummeting down about a mile off from the fishman. Before the fisherman even felt the initial shockwave of the proceeding explosion, every one of the over a hundred passengers aboard ValuJet 592 was dead.

In this article from The Atlantic titled "The Lessons of ValuJet 592," we begin to see how great tragedies such as this crash actually hold a link to the work of computer scientists. The beginning of the article lays the foundation for what happened, then begins to unravel the nature of the mistake that was made. The writer, William Langewiesche, outlines three different categories of failures. The first two are more standard, one being procedural failures where a simple mistake is made leading things astray from procedure, and the other being engineering failure, where something isn't tested properly and thus fails catastrophically.

Langewiesche then offers the third, more pertinent genre of failure, that of system failure. This type of failure is something that must be avoided for the future through rigorous testing and analysis of systems. He makes it clear though that "system-accident thinking does not demand that we accept our fate without a struggle, but it serves as an important caution" (Langewiesche, 1998). Just because a system has failed or will fail does not mean to throw in the towel and resign to the whims of that failure, but rather we use that failure as a cautionary tale for the future, such as we have done in aviation with this crash and many crashes like it.

Even after pinpointing the issue of the crash, rooted somewhere in the architecture of the emergency oxygen systems, there is still more to be learned from the event than just the fact that this system needed to be fixed. Infact, this story can carry out lessons beyond aviation or engineering, but into the world of software. Often when crashes such as this occur, the root of the cause can seem so mundane or elementary that it is a struggle to understand how it wasn't caught in the first place. However, aeronautical engineering comprises thousands, if not millions, of these tiny contributing parts that make up the whole of the operation. Another such example is that of the Challenger space shuttle disaster, where a single seal intended to prevent leaks failed in the cold temperatures and caused the entire spaceship to be engulfed in flames.

Single failures of an overall system can have devastating effects. Back to the world of software development, the effects are admittedly less deadly most of the time, but can still have incredible scale. A great example of this is in the release of the video game Alien: Colonial Marines back in 2013. This game initially released with horrible reviews, primarily because of the AI of the xenomorph aliens being bug-ridden and overall idiotic. Later analysis of the game code brought forth an incredibly simple bug that could have been at the root of the problem. At a certain point in the back end of the code that drives the alien AI, there was a reference to "AttachPawnToTeather." After realizing that "teather" is misspelled and actually references a bit of code that does nothing for the AI, a community modder named James Dickinson changed the reference to "AttachPawnToTether" (Lawler, 2018). This change caused a vast improvement in the alien's AI and greatly enhanced the game. This makes you wonder that if only this tiny spelling error had been corrected upon launch, then perhaps this game would have had a much stronger release and not lost the studio millions of dollars.

Such horror stories of software development may not be caused by extreme disasters or loss of life like the crash of ValuJet 592, but they are still incredibly important to understand nonetheless. The question then turns to: how could this be avoided? The simple answer is more testing and more caution, but what makes up those factors? Similar to what we are learning currently in class, a well-designed development process is incredibly important for consistency of the quality of work being generated. This carries over for both software development as well as engineering, or essentially any field in which work is being done. The systems put in place for designing key features of products, whether that be video games or commercial jets, are incredibly important to the outcome or safety of the final release.

Though software engineering may not be as high stakes as engineering in space or aviation, it still could have seriously detrimental effects if errors slip through the cracks. A prominent example would be that of emergency response systems: if a system for an earthquake or a tsunami has a back-end software system to ensure it reaches the correct people, if that system fails then lives could be at stake. Similarly, at our very own school of LMU we can find a perfect example of a software-backed system that did not work according to its mission. Two years ago, there was an on-campus shooter scare that stemmed out of the freshman dorms as a person walked by a student who seemed to be warning of an active shooter. It later came out that this was all rooted in a misunderstanding and a stupid prank, but regardless the systems in place should have responded as if it was a full scale threat. Instead, students received duplicate messages many minutes, and up to even an hour after the report came in. Most students were alarmed first by the convoy of police cars and the police helicopter, which then caused an onslaught of text messages cascading out between people and group chats. These messages were all word-of-mouth and incredibly error-prone. This was the point where the school should have sent out a official text to all students letting them know the situation, but instead we got greatly delayed responses with disjointed messaging, nothing that would have actually helped us in a real crisis.

Overall, within the realm of software development, it is extremely important to set up the systems necessary to ensure that the final product of whatever is being produced is of the highest quality. Situations like the video game release of Alien: Colonial Marines or of the active shooter threat we had here on LMU campus are prime examples of what can go wrong when even the smallest mistakes slip through the code and into the final product. Ensuring that the final code is held to the highest standard through development and even after release ensures the best quality product, and can even save lives in the end.