ValuJet Reading Synopsis and Response

The article regarding the ValuJet 592 crash details the causes and people to blame for the eventual crash of the airplane. It first describes the initial moments of people’s reactions to the crash. It describes how the plane crashed into a swamp in Florida, and was reported seen by a few bystanders. The article takes a moment to discuss the moments just before the crash as the pilots contact the controller to give them a chance at a safe landing, but ultimately fail. The controller has a hard time understanding the pilots as they struggle through failing equipment to explain their condition, until the plane finally crashes. The article goes on to discuss the aftermath of the crash, including the responders and the investigators of the crash. An investigation begins to try to figure out what caused the crash. What ensues is a web of culpability, as blame for the incident is passed along to different groups of people. The article describes the actual physical cause, which were oxygen generators that then caught fire which then possibly ignited tires which then also caught fire and possibly exploded. Their were supposed to be safety caps added to these oxygen generators, but this process was neglected by SabreTech. ValuJet contracted ValuJet to handle maintenance, and in this case they were supposed to handle the oxygen generators correctly. The article then shows how it is difficult to assign blame, as it is the fault of ValuJet for contracting SabreTech, and the employees of SabreTech were also at fault for not performing what was necessary of them. At the same time SabreTech never reassured that every employee was behaving correctly. The FAA is also brought into blame as it should have inspected ValuJet more thoroughly and ensured that every plane was up to standards.

One way that software could have led to a disaster similar to the one in the article would be any software designed to speed up the process of logging information. For example, in the case of the oxygen generators missing the caps, if the workers were supposed to log whether they did or did not put the safety caps on, and they mark that they did not put the caps on, there is a possibility that the software makes a mistake and puts down that the caps were added correctly. This, of course, would be quite a big mistake to make, and though it would be a much less likely mistake to make, it is still a possibility. On the other hand, not introducing technology in this case would maintain culpability in the hands of the person marking off whether they did or did not put the safety caps on. In this case, the solution that removes unnecessary complexity, no matter how convenient, seems to be the right solution. Another type of failure that could result in a disaster similar to this is any software dealing with hazard detection. In the case of the plane, a hazard detection device could have noticed the smoke as soon as it started and alerted someone. However, if in fact it malfunctioned and determined the smoke to benign, it could result in the events that occurred in the plane crash. This would be similar to the tesla accident in which the self driving tesla believed a white truck to be a cloud and ran into it, killing the person inside it.

The main takeaway from the article is that in a complicated network of systems, trying to reduce error becomes difficult. The article suggests that the natural tendency is to make more regulations and make more complicated systems to handle all possible errors. This of course is a natural inclination, as the more procedures are in place to handle errors, the less possibility for errors to slip through the cracks. However, this is the wrong approach, according to the article, as making the system more complex will make it harder to manage, and thus more prone to errors. Instead, one should rely on a more simple system that can then be regulated in a proper fashion. On the software side of things, this seems like good advice, as a programmers first instinct is to make a complex solution that will allow them to exercise their deep thinking abilities. However, these solutions are not always necessary, and being able to take a step back and find the right kind of solution for the right situation is an important skill to have. The unix philosophy dictates a similar idea in that one should try to find simple working solutions to problems first, rather than faster more complex solutions. The reason for this, as the article states, is that solutions that are simple are far less error prone, and often times a complex solution does not give enough benefits to warrant its use. In this same fashion, the article suggests that a system could be regulated by multiple external sources to ensure that errors are reduced as much as possible. One way to regulate a system to reduce errors is from an economic standpoint, in which the airlines are regulated in order to keep competition down so that cheap airlines do not skip out on safety. This idea could easily apply to software companies, as competing companies may skip out on properly testing code to reach deadlines earlier. However, removing the strict competition could stop this kind of behavior, and allow companies to perform the proper checks that are necessary. The other way the article suggests is by using an operational approach, such as the FAA in regards to airlines. In this case, software companies hugely benefit from a regulating force, as it allows them to be sure that their software is up to proper standards. With this, the stress of having to regulate it themselves is no longer an issue, and the company can turn their focus toward other issues. Otherwise they would run the risk that ValuJet did, which could greatly hurt the company image.