**Assignment 1**

On May 11, 1996 a terrible yet highly interesting accident occurred when ValuJet 592 crashed in southern Florida. Not long after taking off, a regularly scheduled flight from Miami to Atlanta from ValuJet Airlines caught on fire and crashed, killing everyone onboard. The article tells the story of this event and explains in depth how the failure that caused it can teach us many things about safety.

The story begins with a man calling to report the accident from his boat at Everglades Holiday Park. As a pilot himself, the man was able to identify the kind of plane involved in the accident, but also due to his experience, found it very hard to believe, because in his mind planes were highly automated machines and thus very safe.

The article goes on to describe the three main kinds of airplane accidents that can occur. The first kind, and the most common, are the so-called procedural accidents. These kinds of accidents result from obvious mistakes like flying into a storm or taking off when the wings of the plane are frozen. Procedural accidents are also the easiest to understand and prevent because they consist of simple (though catastrophic) mistakes. The next kind are engineered accidents, which refer to hardware flaws in the aircraft. These include design flaws as well as things that might break during the flight. Finally, the third kind of accident are the system accidents. A system accident is what happened to ValuJet, which is also the hardest kind of accident to identify and also to solve. The author explains that system accidents arise from the confusion that lies within the complex organizations with which we manage our dangerous technologies.

Although it seemed at the beginning that a fire had caused the crash, it took a few days of exhaustive investigation to really understand the root of the problem. It turned out that a series of events, driven by trying to make the flight as cheap as possible when fixing the emergency oxygen supply, somehow caused the problem to occur.

The final conclusion of the article summarizes that there really is not a single person that can be blamed for this accident. Instead, the article suggests that the accident was actually arguable un avoidable at that point since whenever there are many elements involved, those elements are linked in multiple and often unpredictable ways. The argument follows that the failure of one part may coincide with the failure of an entirely different part, and this unforeseeable combination may cause the failure of other parts, leading to an accident. Therefore, If the system is large, the possible combinations of failures are practically infinite, and chaos is bound to unleash.

In the light of the above, there are many ways in which software can contribute to a system accident. In fact, software these days solves increasingly complex problems, and is integrated in larger and larger workflows. Take the example of Facebook. Currently, Facebook is the single largest social network in the web. The service is composed of hundreds and hundreds of features, and the platform grows each day. Even though Facebook prides itself for having the best software developers in the word, we all know that accidents are not uncommon in this platform. We need not look too long back to find the time of the famous Cambridge Analytica scandal, in which features of the platform were abused to undermine the electoral system of the largest economy in the world, the United States. Clearly, this was not a single person’s fault, and even though we might get angry at Mark Zuckerberg for this dilemma, he is only one of the factors that contributed to this highly complex series of events that lead to the "accident."

As software becomes a bigger and more important part of our lives, and as the software solutions we build become bigger and more complex, new paradigms of development arise to help guide the process in a systematic way. Test driven development is considered good practice and should help develop minimize risk when implementing software that has multiple parts. Say for example that google wants to add a new feature to it google maps API which allows users to track other people’s location in real time. In order to avoid abuse of the service (as happened in the Facebook example) google should begin the development of this new feature with a set of tests, which during development, will help to make sure that everything that the engineers work on aligns exactly with the plan. Tests serve also as documentation, providing examples of how the API can and should be used once available.

As I have discussed, the lessons of ValuJet are highly relevant in a software environment. Perhaps the most important lesson of the ValuJet accident is the fact that complexity produces errors, and errors produce accidents. As developers we should strive to remove complexity whenever possible from our designs. One interesting idea comes from the Agile project management methodology, under which teams are encouraged to design what are called MVP's (minimum viable products) that focus only on those features that provide the most value to the customer, and trims everything else that is not essential. Under this way of thinking, development occurs much faster, giving more time for testing. The final product is also simpler and leaves less room for failure.

The lessons of ValuJet also include ethical dilemmas. As an engineer it is easy to forget the impact that our products make in the world. It is thus of outmost importance that we consider every ethical question that might arise in our work. For example, the flight engineers who fixed the plane's emergency oxygen supply should have considered the ethical problems that would arise from focusing too much on reducing the cost of the product. Likewise, as software developers, our products might negatively impact our users, even if we are trying to do the right thing. Just like ValuJet should not have grown so fast, nor should we rush into development whenever we think have a great idea for the product. Instead we must pause to consider the consequences of this feature and be cautious as to the ethical implications of our work.