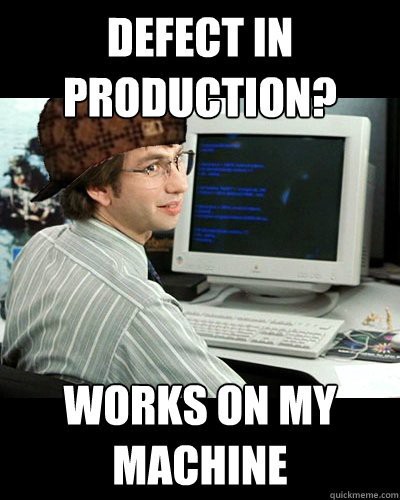
4 Interesting Cases of Software Failures and Consequences

Did you ever get an email from the hospital saying you are dead? 8,000 people did… and that’s one example.



There 2 factors, or considerations, in the area of Software Quality Assurance:

1. We bake a quality design, which will preclude problems from ever happening, right into the structure and arrangement of our Program Modules/Classes. (Example: Having only 1 data dictionary in the entire application which all the modules need to reference. They don’t keep their own local copies of the data/ which have different dimensions or go out of sync).
2. I write my methods to be easily testable.

If we do these 2 things: It will

The beauty of software development is that with just a computer and access to the internet amazing things can be created.

The role of software is apparent in multiple areas of our lives: educations, finance, healthcare, communication, and more. As a software engineer myself, I can appreciate the power and complexity involved in many of the software systems I use daily. However, the more I learn about software and its development process, the more I learn about their weaknesses and potential threats.

Although software systems are effective at calculating large and complex data, they have one main weakness: humans create these systems. And we humans make mistakes… lots of them. Therefore, it is natural that the software systems we build contain errors and are prone to failure.

Software systems have become such an essential part of our economy that whenever they fail, there are economic consequences. A research study done by software testing company Tricentis revealed that in the year 2017 software failure affected 3.6 billion people and caused $1.7 trillion in financial losses [1].

To give you an idea of possible consequences that may result from software failure, in this article, I will be presenting cases of software failure and its effects.

**Case #1: St. Mary’s Mercy Hospital**

Imagine waking up one day, checking your mailbox an receiving a letter from your hospital saying you died. Well, that is precisely what happened to 8500 people who received treatment between Oct 25 and Dec 11 at St. Mary’s Mercy Hospital. So what happened? It turns out the hospital had recently upgraded its patient-management software system. However, a mapping error in the software resulted in the system assigning a code of 20 (which means “expired”) instead of 01 which meant the patient had been discharged. But that is not all. The erroneous data was not only sent to the patients but also to insurance companies and the local Social Security Office. It is not clear how [2]

**What is Failure Pattern? Different modules of the Software Application did not reference a standard Dictionary of Definitions.**

**Case #2: National Health Service**

I don’t know what is worse: Not taking your medicines at all or taking the wrong medication. Either way, at least 300,000 heart patients were given the wrong drug or advise as a result of a software fault. So, what happened? In the year 2016, it was discovered that the clinical computer system SystemOne had an error that since 2009 had been miscalculating patient’s risk of heart attack. As a result, many patients suffered heart attacks or strokes since they were told they were at low-risk, while other suffered from the side-effects of taking unnecessary medication [3].

**Case #3: Air Traffic Control in LA Airport**

The air traffic control has the important responsibility of informing aircraft pilots about relevant information regarding weather, routes, the distance between other airplanes, and more. Failing to communicate with aircraft pilots promptly could result in catastrophe. On September 14, 2004, at 5 P.M. air traffic control at the LA airport lost voice communication with approximately 400 airplanes being tracked in the southwestern United States and many planes were headed towards each other. So what happened?

The primary voice communication system shut down unexpectedly. To top it off the backup system failed a few minutes after it was turned on. The cause of the error was that the communication system had an internal timer that ticks off in milliseconds. After it reached zero, it could not time itself so it would shut down. The outage affected 800 flights across the country [4].

**Pattern of the Failure:** When the Test Team Designed the Traceability Matrix, and the method unit tests, they did not drill enough into the EDGE Case of the Timer reaching Zero.

SOLUTION: Design a well-specified state table for their test cases – so they could have a blueprint to drill into the EDGE CASES (there are always edge cases when a variable hits ZERO, or transitions across zero – positive to negative).

**Case #4: Toyota**

In the mid-2000’s many Toyota drivers were reporting that their car was accelerating without them touching the pedal. After a series of accidents, which lead to investigations, investigators discovered that software errors were the cause of the unintended acceleration. In this case, there was a series of things wrong with the software installed in Toyota cars: Memory corruption, wrong memory handling, disabling safety systems, systems with single points of failure, and thousands of global variables. Toyota recalled millions of vehicles and Toyota’s stock price decreased 20% a month after the cause of the problem was discovered. This case demonstrates the consequences of not giving enough attention to good programming practices and testing as a result of wanting to launch the product.

**Conclusion**

In this article, we examined various cases of software failure and their consequences. These cases demonstrate that our society has a high dependency level on software and that whenever it fails, not only economic consequences can arise.

As long as humans are involved in the development process, software systems will contain errors and will be prone to failure. As software developers, our responsibility is to ensure that the systems we built are thoroughly tested in different and realistic conditions. It is to ensure that the software we are promoting is actually capable of helping and not harming its users. In many cases, competition and the desire to be the first on the market are the motivators for launching an untested and unfinished product. As software users, our responsibility is to use our software tools as a support for our activities and not blindly accept their results or suggestions.

If you enjoyed this article, please recommend and share. Don’t forget to  
subscribe and follow me on Twitter to stay up-to-date with my latest posts. See you in the next article.

**References**

[1] <https://www.techrepublic.com/article/report-software-failure-caused-1-7-trillion-in-financial-losses-in-2017/>  
[2] <http://www.baselinemag.com/c/a/Projects-Networks-and-Storage/Hospital-Revives-Its-QTEDeadQTE-Patients>  
[3] <http://www.dailymail.co.uk/health/article-3585149/Up-300-000-heart-patients-given-wrong-drugs-advice-major-NHS-blunder.html>  
[4] <http://www.cse.psu.edu/~gxt29/bug/softwarebug.html>