

1 Main Notes

Contents

1	Main Notes	1
2	Fault Tolerant Systems	2
2.1	Introduction	2
2.2	Examples	2
2.3	How to make a system more fault tolerant	3
2.3.1	Hardware	3
2.3.2	Software	3
3	Sensors	3

2 Fault Tolerant Systems

2.1 Introduction

So, what does it mean for a system to be fault tolerant, what does it mean for a system to be reliable? How do we know if a system is reliable? How can we make it more reliable? These concepts will be explored in this chapter.



Definition — Fault Tolerance

A system is fault tolerant if

the system continues to operate CORRECTLY and SAFELY in the presence of faults

Simple enough, but what is a fault?



Definition — Fault

A fault is an unexpected irregularity within the input or output of the system.

We should note that it is impossible to make something completely fault proof, there are too many environmental factors to make this possible. With these basic definitions under our belt, we will explore some systems that proved not fault tolerant, and some that proved to be and why.

But first, let's list some particular industries where fault tolerance is not a nice-to-have, it is a necessity. Then we will look at specific examples.

2.2 Examples

1. Transportation

Can be broken down further into specific industries like cars, boats, trains, planes, etc

The flops

Big flop is Boeing. The issue was that they had made a new derivative of the 737, the 737 MAX. Boeing had made several changes to the 737 MAX, and one of these changes was a system to warn the pilots if the plane was stalling using an angle of attack sensor. The 737 MAX had two, however the software system only used one, and never checked the measured values against other sensors that could be used to complement the reading for accuracy. Furthermore, because pilots were never told about this new feature, they were not ready to handle it. In short, the fleet was grounded because the new system had a **single point of failure**. And when this went, everything else did too.

2. Banking

3. Medical Devices

4. Space

5. Energy

6. Communications



Definition — Reliability

Reliability has a specific definition for us. While we know what it MEANS to be reliable or not reliable, reliability is:

the probability that a system will continue operation correctly during a period of time



Definition — Availability

Availability is defined as:

the probability that at a given moment, the system is working correctly and is available.



When we say correctly, we mean it!

Note that this is for continuous reliability, if you say that system has 0.99 percent reliability during a 5 hour interval, there cannot be these consistent spikes in instability after the first hour, that means the system is not 0.99 percent reliable during this time period.

As with most things, engineers and scientists cannot resist modelling stuff with equations. Reliability is no exception. We generally say that reliability is modelled as:

$$R(t) = e^{-\lambda t} \quad (1)$$

where t is defined as the time, and the output is the reliability of the system at t . So if we wanted our .99 reliability at 5 hours, we would need for

2.3 How to make a system more fault tolerant

The main point of this section can be summed up as **Redundancy**. Redundancy is the best way to achieve a more fault tolerant system. While having things such as high quality batteries, ECC memory, or well designed and implemented software, redundancy is what will let you make the most out of such equipment.

2.3.1 Hardware

2.3.2 Software

Software is an interesting case, since as long as the hardware doesn't give way, software will do exactly what you tell it to do, that and only that. So much thought must go into how you make the software and verify it.

However, when you are running software, redundancy is still important. The way that redundancy in software is achieved is by running multiple copies of the software. The way that this can achieve redundancy is that if there is some mission critical variable (such as angle of attack!), after each program has calculated its value for such variable, they can vote (especially if it is discrete), or take a mean, median, mode, whatever. If they take all the values and average them out, hopefully they will smooth out any error and come to as close of an accurate value as possible.

Multiple Implementations

Given some set of requirements designed by engineers, you could have multiple teams implement the same set of requirements. By having this, your goal is to minimize the amount of faults that can occur through bugs in implementation.



Definition — N Version Programming

When you have N teams implement the same set of requirements.

3 Sensors